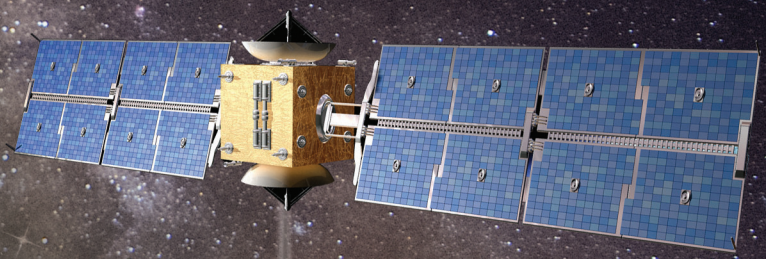
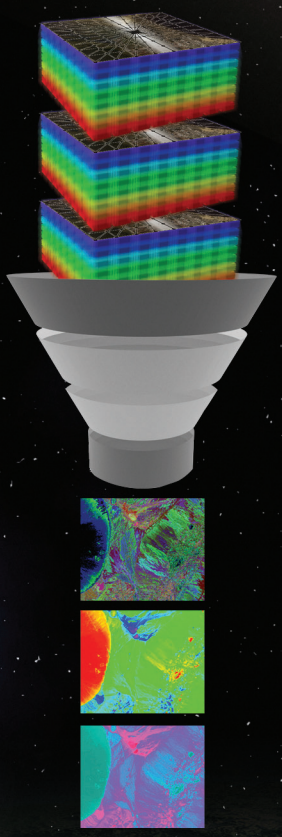
SPIE.





OPTICAL SATELLITE

Data Compression and Implementation

Shen-En Qian

OPTICAL SATELLITE

Data Compression and Implementation

OPTICAL SATELLITE
Data Compression and Implementation

Shen-En Qian

SPIE PRESS
Bellingham, Washington USA

Library of Congress Cataloging-in-Publication Data

Qian, Shen-En.
Optical satellite data compression and implementation / Shen-En Qian.
pages cm
Includes bibliographical references and index.
ISBN 978-0-8194-9787-1
1. Data compression (Computer science). 2. Imaging systems Image quality.
3. Signal processing. 4. Coding theory. 5. Optical images. I. Title.
QA76.9.D33 2013
629.43'7 dc23

2013944363

Published by

SPIE The International Society for Optical Engineering
P.O. Box 10
Bellingham, Washington 98227-0010 USA
Phone: +1 360 676 3290
Fax: +1 360 647 1445
Email: spie@spie.org
Web: http://spie.org

Printed in the United States of America.
First printing

Contents

Preface

Satellite data compression has been an important subject since the beginning of satellites in orbit, and it has become an even more active research topic. Following technological advancements, the trend of new satellites has led to an increase in spatial, spectral, and radiometric resolution, an extension in wavelength range, and a widening of ground swath to better serve the needs of the user community and decision makers. Data compression is often used as a sound solution to overcome the challenges of handling a tremendous amount of data. I have been working in this area since I was pursing my Ph. D. thesis almost 30 years ago.

Over the last two decades, I—as a senior research scientist and technical authority with the Canadian Space Agency—have led and carried out research and development of innovative data compression technology for optical satellites in collaboration with my colleagues at the agency, other government departments, my postdoctoral visiting fellows, internship students, and engineers at Canadian space industry. I invented and patented two series of near-lossless satellite data compression techniques and led the Canadian industry teams who implemented the techniques and built the onboard near-lossless compressors. I also led a multidisciplinary user team to assess the impact of the near-lossless compression techniques on ultimate satellite data applications. As the representative of Canada, I am an active member of the CCSDS working group for developing international data-compression standards for satellite data systems. Three international satellite data compression standards have been developed by the working group and published by the International Organization for Standardization (ISO). In collaborating with experts in this area in the world, I have co-chaired an SPIE conference on satellite data compression, communication, and signal processing since 2005. I have published over sixty papers and currently hold six U. S. patents, two European patents, and several pending patents in the subjects of satellite data compression and implementation. I feel that I have acquired sufficient knowledge and accumulated plenty experience in this area, and it is worth the effort to systematically organize them and put them into a book.

This book is my attempt to provide an end-to-end treatment of optical satellite data compression and implementation based on 30 years of firsthand

experience and research outcomes. (It is a companion text to my book *Optical Satellite Signal Processing and Enhancement*, published by SPIE Press.) The contents of the book consist of nine chapters that cover a wide range of topics in this field. It serves as an introduction for readers who are willing to learn the basics and the evolution of data compression, and a guide for those working on onboard and ground satellite data compression, data handling and manipulation, and deployment of data-compression subsystems. The material is written to provide clear definitions and precise descriptions for advanced researchers and expert practitioners as well as for beginners. Chapters open with a brief introduction of the subject matter, followed by a review of previous approaches and their shortcomings, a presentation of recent techniques with improved performance, and finally a report on experimental results in order to assess their effectiveness and to provide conclusions.

Chapter 1 is the introduction to the book that describes the rationale and needs for satellite data compression and introduces a set of image quality metrics for assessing compressed satellite images. Chapter 2 presents a review of satellite lossless-data-compression techniques, considering both prediction-based and transform-based methods. Chapter 3 summarizes three international satellite-data-compression standards developed by CCSDS from the perspective of applying the standards. Chapter 4 describes vector quantization (VQ) based data-compression techniques that I have developed for compressing hyperspectral data. The focus of the research was to significantly reduce the computational complexity of conventional VQ algorithms in order for them to effectively compress hyperspectral datacubes. Many innovative yet practical solutions have been developed, including two of my granted patents: Successive Approximation Multi-stage Vector Quantization (SAMVQ) and Hierarchical Self-Organizing Cluster Vector Quantization (HSOCVQ). Chapter 5 describes how both of these techniques solve the blocking effect when applied to compressing continuous data flow generated aboard satellites and how they restrict the compression error to a level lower than that of the intrinsic noise of the original data to achieve so-called near-lossless compression. Chapter 6 addresses the optimization and implementation aspects of onboard data compression; aspects include the effect of anomalies of input data on compression performance, the location in the onboard data-processing chain where the compressor should be deployed, and the techniques to enhance error resilience in the data downlink transmission channel. Chapter 7 describes the hardware implementation of compression engines and onboard compressors that are based on SAMVQ and HSOCVQ. Chapter 8 reports a multidisciplinary user-acceptance study that assessed the impact of the compression techniques on various hyperspectral data applications to address the users' concern about possible information loss due to the lossy compression nature of SAMVQ and HSOCVQ. Chapter 9

describes the Hyperspectral Image Browser (HIBR) system, which is capable of remotely displaying large hyperspectral datacubes via the Internet and of quickly processing the datacubes directly on the compressed form for users to identify the interested data, whose richness comes mostly from the spectral information.

There are many people I would like to thank for their contributions to the works included in this book. I would like to thank the Canadian Space Agency, where I have been working for the last 20 years; my colleagues Allan Hollinger, Martin Bergeron, Michael Maszkiewicz, Ian Cunningham, and Davinder Manak for their participation in data compression projects; my postdoctoral visiting fellows Pirouz Zarrinkhat and Charles Serele; and over forty intern students who have each left their mark. I would like to thank Robert Neville (retired), Karl Staenz (now at the University of Lethbridge), and Lixin Sun at the Canada Centre for Remote Sensing for collaborating on the Canadian hyperspectral program; Josée Lévesque and Jean-Pierre Ardouin at the Defence Research and Development Canada for their collaboration on assessing the impact of data compression. I thank David Goodenough at the Pacific Forestry Centre; John Miller and Baoxin Hu at York University for providing datasets and for actively collaborating on the data-compression user acceptability study; and Bormin Huang of the Cooperative Institute for Meteorological Satellite Studies at the University of Wisconsin-Madison for his discussion on satellite data compression.

I would also like to thank the participants in the user acceptability study: Andrew Dyk at the Pacific Forestry Centre; Jing Chen at the University of Toronto; Harold Zwick, Dan Williams, Chris Nadeau, and Gordon Jolly at MacDonald Dettwiler Associates; and Benoit Rivard and Jilu Feng at the University of Alberta. I thank Luc Gagnon, William Harvey, Bob Barrette, and Colin Black at MacDonald Dettwiler Associates (former EMS Technologies) for the development and fabrication of onboard compressor prototypes; and Melanie Dutkiewicz and Herbal Tsang for the development of a hyperspectral browser. I thank Valec Szwarc and Mario Caron at the Communication Research Centre (Canada) for discussions on enhancing resilience to bit errors of compressed data in the downlink channel; and Peter Oswald and Ron Buckingham for their discussion on onboard data compression. I would also like to thank Penshu Yeh at the NASA Goddard Space Flight Center, Aaron Kiely at the Jet Propulsion Laboratory, Carole Thiebaut and Gilles Moury at the French Space Agency (CNES), and Raffaele Vitulli at the European Space Agency for the collaboration within the CCSDS in developing international spacecraft-data standards and for their contributions to the CCSDS work included in this book.

I would also like to thank the three anonymous manuscript reviewers for their tireless work and strong endorsement of this book, their careful and meticulous chapter-by-chapter review on behalf of SPIE Press, and their

detailed comments leading to the improvement and final results of the book in its current form. Many thanks as well to Tim Lamkins, Scott McNeill, and Dara Burrows at SPIE Press for turning my manuscript into this book.

Finally, I would like to thank my wife Nancy and daughter Cynthia for their help and support. They provided great encouragement and assistance during the period I wrote this book. The credit of this book should go to them.

Shen-En Qian (**錢神恩**)
Senior Scientist, Canadian Space Agency
Montreal, Canada
September 2013

List of Terms and Acronyms

%E	Percentage error
%SE	Percentage standard error
3D CB-EZW	Three-dimensional context-based embedded zerotrees of wavelet transform
3D-SPECK	Three-dimensional set-partitioned embedded block
AAC	Adaptive arithmetic coding
AC	Arithmetic coding
ACE-FTS	Atmospheric Chemistry Experiment-Fourier Transform Spectrometer
AIRS	Atmospheric infrared sounder
ALADIN	Atmospheric Laser Doppler Lidar Instrument
ALI	Advanced Land Imager
ALOS	Advanced Land-Observing Satellite
AMEE	Automated morphological end-member extraction
AOS	Advanced orbital system
AOTF	Acousto-optical tunable filter
APD	Avalanche photodiode
APRAD	Average percent relative absolute difference
APSICL	Adjacent pixel spectra in a cross-track line
A-RMSE	Absolute root mean square error
ARSIS	Amélioration de la résolution spatial par injection de structures
ARTEMIS	Advanced Responsive Tactically Effective Military Imaging Spectrometer
ASIC	Application-specific integrated circuit
ATGP	Automatic target generation process
AVIRIS	Airborne visible/infrared imaging spectrometer
AVNIR	Advanced visible and near-infrared radiometer
AWGN	Additive white Gaussian noise
BCM	Band correlation minimization
BDM	Band dependence minimization
BER	Bit-error rate
BIP	Band interleaved by pixel

BIPLGC	Binary-input power-limited Gaussian channel
BP	Belief propagation
BPE	Bit-plane encoder
BPOC	Base-bit plus overflow-bit coding
bpppb	Bits per pixel per band
BPSK	Binary phase shift keying
BRDF	Bidirectional reflectance distribution function
BSQ	Band sequential
CALIOP	Cloud-Aerosol Lidar with Orthogonal Polarization
CASI	Compact airborne spectrographic imager
CBERS	China-Brazil Earth Resources Satellite
CC	Correlation codevector
CCD	Charge-coupled device
CCSDS	Consultative Committee for Space Data Systems
CDS	Coded dataset
CE	Compression engine
CEM	Constraint energy minimization
CEOS	Committee on Earth Observation Satellites
CFDP	CCSDS File Delivery Protocol
CGT	Codebook generation time
CHRIS	Compact high-resolution imaging spectrometer
CR	Compression ratio
CrIS	Cross-track Infrared Sounder
CRISM	Compact Reconnaissance Imaging Spectrometer for Mars
CRT	Complex ridgelet transform
CSCI	Component software-configurable item
CT	Coding time
CT	Computation time
CV	Codevector
CVQ	Correlation vector quantizer
CZT	Cadmium-zinc-telluride
DAAC	Distributed active archive center
DC	Digital count
DCT	Discrete cosine transform
DCWG	Data Compression Working Group
DIV	Difference in variance
DLP	Diagonal linear projection
DLS	Diagonal linear shrinker
DMA	Direct memory access
DN	Digital number
DPCM	Differential pulse code modulation
DSP	Digital signal processor
DT	Decoding time

DTCWT	Dual-tree complex wavelet transform
DWT	Discrete wavelet transform
EDU	Engineering demonstration unit
EM	Endmember
EnMAP	Envrionmental Mapping Analysis
EOS	Earth Observing System
ETF	Electronically tunable filter
ETM	Enhanced thematic mapper
ETM+	Enhanced thematic mapper plus
EUMETSAT	European Organization for the Exploitation of Meterological Satellites
EV	Earth view
EZW	Embedded zerotrees of wavelet transforms
FCLSLU	Fully constrained least-squares linear unmixing
FER	Frame-error rate
FFT	Fast Fourier transform
FIFO	First-in first-out
FIPPI	Fast iterative pixel purity index
FIR	Far-infrared
FOV	Field of view
F-P filter	Fabry-Pérot filter
FPA	Focal plane array
FPGA	Field programmable gate array
FPR	False positive rate
FPVQ	Fast precomputed vector quantization
FR	Full reference
FRIT	Finite ridgelet transform
FTHSI	Fourier Transform Hyperspectral Imager
FTS	Fourier transform spectrometer
FWHM	Full width at half maximum
GIFTS	Geosynchronous Imaging Fourier Transform Spectrometer
GLA	Generalized Lloyd algorithm
GLAS	Geoscience Laser Altimeter System
GPO2	Golomb power-of-two coding
GSD	Ground sample distance
GUI	Graphical user interface
HIBR	Hyperspectral image browser
HIS	Intensity-hue-saturation
HPF	High-pass filter
HRG	High-Resolution Geometrical
HRV	High-Resolution Visible
HRVIR	High-Resolution Visible and Infrared
HS	Histogram-based segmentation

HSOCVQ	Hierarchical self-organizing cluster vector quantization
HVS	Human visual system
HYDICE	Hyperspectral Digital Image Collection Experiment
IARR	Internal average relative reflectance
IASI	Infrared atmospheric sounding interferometer
IBP	Iterative back-projection
IC	Isoclustering
IC	Integrated circuit
ICESat	Ice, Cloud, and Land Elevation Satellite
IEA	Iterative error analysis
IFOV	Instantaneous field of view
IFTS	Imaging Fourier transform spectrometer
IIR	Imaging Infrared Radiometer
IRMSS	Infrared Multispectral Scanner
ISO	International Organization for Standardization
ISRO	Indian Space Research Organization
IWT	Integer wavelet transform
JAXA	Japan Aerospace Exploration Agency
JPL	Jet Propulsion Laboratory
KLT	Karhunen–Loéve transform
LAI	Leaf area index
LAIS	Locally averaged interband scaling
LBG	Linde–Buzo–Gray
LCMV-CBS	Linearly constrained minimum variance constrained band selection
LCTF	Liquid crystal tunable filter
LDC	Lossless data compression
LDCM	Landsat Data Continuity Mission
LDPC	Low-density parity check
LITE	Lidar In-space Technology Experiment
LLE	Locally linear embedding
LOCO	Low-complexity lossless compression
LOLA	Lunar Orbiter Laser Altimeter
LOS	Line of sight
LRO	Lunar Reconnaissance Orbiter
LSU	Linear spectral unmixing
LUT	Lookup table
M3	Moon Mineralogy Mapper
MAD	Maximum absolute difference
MAE	Mean absolute error
MC 3D-EZBC	Motion-controlled three-dimensional embedded zeroblock coding
MCT	Mercury–cadmium–telluride

MDD	Minimum distance detection
MDP	Minimum distance partition
MDS	Minimal distance selector
MEI	Morphological eccentricity index
MERIS	Medium-Resolution Imaging Spectrometer
MGS	Mars Global Surveyor
MHS	Modified histogram-based segmentation
MIR	Middle-infrared
MISR	Multi-angle imaging spectroradiometer
MLA	Mercury Laser Altimeter
MNF	Minimum noise fraction
M-NVQ	Mean-normalized vector quantization
MODIS	Moderate-resolution imaging spectroradiometer
MOLA	Mars Orbiter Laser Altimeter
MOMS	Modular optoelectronic multispectral scanner
MOS	Modular optoelectronic scanner
MPS	Mean-distance-order partial search
MRO	Mars Reconnaissance Orbiter
MS	Multispectral
MSA	Maximum spectral angle
MSCA	Multiple-subcodebook algorithm
MSE	Mean square error
MSID	Maximum spectral information divergence
MSS	Multispectral Scanner
MSSE	Mean square spectral error
MSX	Midcourse Space Experiment
MT	Multi-thresholding
MTF	Modulation transfer function
NDVI	Normalized difference vegetation index
NEΔT	Noise-equivalent change in temperature
NGST	Next-Generation Space Telescope
NIR	Near-infrared
NIST	National Institute of Standards and Technology
NN	Nearest neighbor
NNP	Nearest-neighbor predictor
NPS	Nearest partition set
NR	No reference
NR	Noisy radiance
NRR	Noise-removed radiance
NWP	Numerical weather prediction
OPD	Optical path difference
OSP	Orthogonal subspace projection
PALSAR	Phased Array-type L-band Synthetic Aperture Radar

PCA	Principal component analysis
PCB	Print circuit board
PD	Probability of detection
PDS	Partial distance search
PDS	Planetary Data System
PFA	Probability of false alarm
PMAD	Percentage maximum absolute difference
PPI	Pixel purity index
PRISM	Panchromatic Remote-sensing Instrument for Stereo Mapping
PROBA	Project for Onboard Autonomy
PSF	Point spread function
PSNR	Peak signal-to-noise ratio
PT	Processing time
QLUT	Quantized-index lookup table
RBV	Return Beam Vidicon
RDCT	Reversible discrete cosine transform
RE	Ratio enhancement
REP	Red-edge position
ReRMSE	Relative root mean square error
RF	Radio frequency
RGB	Red-green-blue
RMSE	Root mean square error
RMSSE	Root mean square spectral error
ROC	Receiver operating characteristic
ROI	Region of interest
RR	Reduced reference
RTDLT	Reversible time-domain lapped transform
SA	Spectral angle
SAM	Spectral angle mapper
SAMVQ	Successive approximation multistage vector quantization
SAR	Synthetic aperture radar
SC	Spectral correlation
ScaRaB	Scanner for radiation budget
SCPS	Space Communications Protocol Specifications
SDD	Standard deviation difference
SeaWiFS	Sea-viewing wide-field-of-view sensor
SEU	Single-event upset
SFBBC	Spectral-feature-based binary code
SFF	Spectral feature fitting
SFFS	Sequential forward-floating selection
S-FMP	Spectral fuzzy-matching pursuits
SFS	Sequential forward selection

SFSI	Short-Wave Infrared Full-Spectrum Imager
SGA	Simplex growing algorithm
SID	Sub-identity
SLA	Shuttle Laser Altimeter
SLSQ	Spectrum-oriented least squares
SNR	Signal-to-noise ratio
SOAD/SOSD	Sum of absolute/squared distance
SOFM	Self-organizing feature map
SPIHT	Set partitioning in hierarchical trees
SPIM	Spectrographic imager
SPOT	Système Pour l'Observation de la Terre
SRBC	Solar-radiation-based calibration
SRF	Spectral response function
S-RLP	Spectral relaxation-labeled prediction
SSE	Sum of squared error
SSIM	Structural similarity
SSR	Solid state recorder
SV	Spectral vector
SVM	Support vector machine
SVR	Synthetic variable ratio
SWIR	Short-wavelength infrared
TC	Telecommand
TDLT	Time-domain lapped transform
TDM	Time-division multiplex
TERM	Triangular elementary reversible matrix
TES	Tropospheric Emission Spectrometer
TIE	Triangle inequality elimination
TM	Thematic Mapper
TMC	Thematic Mapper calibrator
TOA	Top of atmosphere
TPR	True positive rate
USES	Universal source encoder for space
UVISI	Ultraviolet and Visible Imagers and Spectrographic Imagers
VA	Vector accumulator
VCA	Vertex component analysis
VD	Virtual dimensionality
VHDL	Very high-speed integrated-circuit hardware description language
VI	Vegetation index
VIF	Visual information fidelity
VLSI	Very large scale integration
VM	Verification model
VNIR	Visible and near-infrared

VQ	Vector quantization
WER	Word-error rate
WFC	Wide-Field Camera
WGCV	Working Group on Calibration and Validation
WPT	Wavelet-package transform
WT	Wavelet transform
XML	Extensible markup language
ZC	Zero crossing

Chapter 1
Needs for Data Compression and Image Quality Metrics

1.1 Needs for Satellite Data Compression

In modern times, satellites have become part of our daily life. Thousands of satellites have been launched into orbit around the Earth and other planets for a number of purposes, such as communications, weather forecasting, navigation, Earth observation, and scientific research. Optical satellites measure reflective light in wavelength ranges from the ultraviolet to the infrared (including near-infrared, middle-infrared, and thermal infrared) to observe the Earth and other planets to acquire information that cannot be easily obtained by other means.

Optical satellites, based on their functions, normally carry the following five types of payloads:

1. Panchromatic,
2. Multispectral,
3. Hyperspectral,
4. Fourier transform spectroscopy (FTS), and
5. Light detection and ranging (lidar).

With advancements in sensor technology, the trend of optical satellite development has led to an increase in the number of spectral bands, spectral and spatial resolution, swath (i.e., wider cross-track line on the ground), and radiometric precision to better serve the needs of the user community and decision makers. For example, in the SPOT (Systéme Pour l'Observation de la Terre) satellite series, the spatial resolution (sometimes also referred to as ground sample distance, or GSD) of the panchromatic image is increased from 10 m for SPOT 1–4 to 5 m for SPOT 5,[1] and further increased to 1.5 m for SPOT 6–7. Hyperspectral imagers, such as Hyperion[2] aboard NASA's EO-1 satellite, and imaging Fourier transform spectrometers (also known as ultraspectral sounders) acquire over hundreds to a few thousand spectral bands of images of a scene and generate enormous amounts of data.

The Hyperion sensor generates 220 spectral bands covering a spectral region from the visible/near-infrared to the short-wavelength infrared (400–2500 nm) with a spectral resolution of 10 nm for each ground pixel of 30-m GSD in a cross-track line of 255 pixels in approximately every 4.4 ms. The datarate generated onboard is 220 bands $\times$ 255 pixels $\times$ 4.4 ms $\times$ 12 bits $=$ 151 Mb/s. Advanced Responsive Tactically Effective Military Imaging Spectrometer (ARTEMIS) is a U. S. Department of Defense (DoD) hyperspectral sensor aboard the TacSat-3 satellite.[3] By increasing the spectral resolution to 5 nm, GSD to 4 m, and the number of pixels in a cross-track line to 1000 pixels, the datarate generated by ARTEMIS increases to 400 bands $\times$ 1000 pixels $\times$ 4.4 ms $\times$ 16 bits $=$ 1.44 Gb/s, assuming that the integration time remains the same and a 16-bit digitization.

Consider the atmospheric infrared sounder (AIRS)[4] aboard NASA's Aqua satellite as an example of an ultraspectral sounder. The AIRS data has 2378 spectral channels covering a wavelength range in the infrared region of 3.74–15.4 μm. A day's worth of AIRS data consists of 240 3D granules (datacubes), each with a six-minute duration. Each granule contains 135 scan lines with 90 cross-track footprints per scan line; thus, there are a total of $135 \times 90 = 12{,}150$ footprints per granule. The 16-bit raw radiances are converted into the brightness temperatures and then scaled as 16-bit unsigned integers. The data volume of 240 3D granules is 110 Gb.

There is an increasing need for satellite data compression. Compression is an efficient way to help reduce the amount of data onboard, to effectively transmit the data from space to ground, and to archive and process the data on the ground. In information theory and computer science, data compression, source coding, or bitrate reduction involves encoding information data using fewer bits than the original representation.

In the case of the SPOT satellite series, onboard data compression has been used since the first SPOT satellite to help balance the datarate generated onboard and the data downlink telemetry-channel capability. SPOT 1–4 benefited from a compression ratio of 1.33 to match the onboard datarate and the transmission bitrate of the telemetry channel capability. When the spatial resolution of the panchromatic and multispectral sensors of SPOT 5 are doubled, the onboard datarate is quadrupled.[5] On the other hand, the transmission bitrate of the telemetry channel is unchanged at 50 Mb/s to maintain strict compatibility with SPOT-satellite ground-receiving stations spread across the world. To accommodate the improved spatial resolution, two telemetry channels (instead of one on SPOT 1–4) are used, which simultaneously provide a total data transmission bitrate of 100 Mb/s. With this downlink-channel transmission capacity, it is still not adequate to handle the datarate generated by SPOT 5, even though the transmission bitrate has been doubled. It took a discrete-cosine-transform-based compressor aboard SPOT 5 that yielded a compression ratio $2.4 \sim 3.4$ to solve this problem.

Mars-Express, launched by the European Space Agency in 2003, is an interplanetary exploration mission. It is aimed at visible and near-IR

observation of the surface and atmosphere on Mars. The hyperspectral imagery captured by the imaging spectrometer OMEGA (Observatoire pour la Minéralogie, l'Eau, les Glaces, et l'Activité) is compressed using a wavelet-based algorithm to package it into highly miniaturized datacube.[50] The Mars Reconnaissance mission is another exploration mission launched by NASA in 2005, and it carries the Compact Reconnaissance Imaging Spectrometer for Mars (CRISM) payload to study the mineralogy and atmosphere of the red planet via a hyperspectral imager. Lossless data compression is applied by decorrelating the data using a DPCM method followed by the CCSDS-121 as an entropy encoder.[51] The Third World Satellite (TWSat), launched in 2008, carries the hyperspectral imager HySI-T, which features another example of satellite data compression. The onboard compression is performed using the JPEG2000 algorithm with a compression ratio of 3.4:1.[52]

The Medium-Resolution Imaging Spectrometer (MERIS) instrument aboard ESA's ENVISAT satellite is a hyperspectral sensor used to observe the color of the ocean, both in the open ocean and in coastal zones, to study the oceanic component of the global carbon cycle and the productivity of these regions, amongst other applications.[6] By design, MERIS could record 390 spectral bands covering a wavelength range from the visible to the near-infrared (412–900 nm) with a spectral interval of 1.25 nm. However, MERIS is restricted by its downlink channel transmission capability and transmits only 15 channels, where each channel is an average taken over 8–10 fine spectral bands. Table 1.1 tabulates the bandwidth of the 15 channels and their main applications. MERIS would have transmitted all 390 fine spectral-resolution bands if an onboard data compressor with a compression ratio of 26:1 had been deployed. Qian carried out a study within his space agency right after the launch of MERIS in early 2000.[7] In the study, the author evaluated the advantages and disadvantages of onboard data compression versus band-averaging, and compared the information volume carried by the data received by band-averaging and by onboard compression

Table 1.1 Specifications of the 15 channels of the MERIS sensor.

Channel Number	Bandwidth (nm)	Application
1	402.5 422.5	Yellow substance and detrial pigments
2	432.5 452.5	Chlorophyll absorption maximum
3	480 500	Chlorophyll and other pigments
4	500 520	Suspended sediment, red tides
5	550 570	Chlorophyll absorption minimum
6	610 630	Suspended sediment
7	655 675	Chlorophyll absorption and fluorescence reference
8	673.25 688.75	Chlorophyll fluorescence peak
9	698.75 718.75	Fluorescence reference, atmospheric corrections
10	746.25 761.25	Vegetation, clouds
11	756.88 746.38	Oxygen absorption R branch
12	763.75 793.75	Atmosphere corrections
13	845 885	Vegetation, water vapor reference
14	875 895	Atmosphere corrections
15	890 910	Water vapor, land

under the same downlink channel transmission capability. The study concluded that compression is more beneficial because all 390 fine spectral bands are transmitted to the ground despite the data containing some errors after compression with a ratio of 26. The compressed data provides more information than the transmitted 15 broadband channels.

Satellite data compression is a tradeoff between processing capabilities and data volume (whether it is storage or transmission). Before selecting the compression techniques, it is important to understand the context in which they operate and the constraints that led to their selection. There is not much in common between the requirements for compressing data onboard a satellite and compressing data on the ground. In the former case, computational power is limited and error is unrecoverable, while in the latter case, compression is used to speed up network transfer or processing, but the whole data can be transmitted, if necessary.

Satellite data compression has been an important subject since the beginning of satellites in orbit, and it has become an even more active research topic. Under the same downlink channel transmission capability, onboard compression allows transmitting more data to the ground or increasing the ground coverage if the possible information losses due to compression are well controlled.[8]

1.2 Quality Metrics of Satellite Images

There is a need to assess the quality of satellite data and images. When a satellite produces data products for a particular application, the quality must be verified before the data is delivered to the user community. On the user side, after applying application algorithms to the data or images, the processed images need to be assessed to ensure that the derived intermediate or final products meet the requirements of the application. This chapter describes image quality metrics used to assess the original satellite data and the products derived from the original data or processed data.

The reason for defining a set of comprehensive image quality metrics is obvious. An optical satellite sensor suffers from degradations in the acquisition process related to instrument characteristics, for example, the radiometric noise and modulation transfer function (MTF). Different degradations introduced by the acquisition system cause a loss of image quality. The first degradation of the produced image products is radiometric noise caused mainly by photonic effects in the photon detection process, by electronic devices, and by quantization. This noise can often be assimilated to white noise even if some correlation exists between different bands. A quality metric called signal-to-noise ratio (SNR) is often used to quantify how much the signal has been corrupted by noise.

Other degradations are due to the optical characteristics of the spectrographs. The point spread function (PSF) can cause a smoothing effect along the spatial dimension. The dispersion element of the spectrometer and the characteristics of the detector array can produce a smoothing effect along the spectral dimension. During the characterization of an optical satellite sensor,

properly measuring the image quality related to specific application needs by using quality metrics can help enhance the performance of the sensor by focusing the crucial characteristics to be improved.

A set of spectral band images produced by a multispectral sensor can be affected by registration problems. The misregistration leads to a bad alignment between the spectral band images. An image or a datacube generated by a hyperspectral sensor can be affected by spectral distortion and spatial distortion problems. Spatial distortion prevents a given ground sample from being imaged on the same column of the 2D detector array after the radiance light is dispersed and causes the imaged detector elements to shift. The spectral and spatial distortions of an optical sensor need to be measured using a well-defined metric.

When lossy data compression is applied onboard, i.e., before the data transmission from space to ground, information losses due to compression will be irrecoverable. Some loss of information may occur after compression in order to reduce the datarate to match the downlink telemetry channel capability or transmit more images. It is important to assess the quality after compression. Image quality criteria or distortion measures need to be defined to properly quantify the information lost due to data compression. Quality metrics can be used to ensure that no critical information has been lost during the compression process and that the scientific value of the original data is well preserved.

Image quality metrics can be classified into three categories based on the reference used:

1. Full-reference (FR) metrics,
2. Reduced-reference (RR) metrics, and
3. No-reference (NR) metrics.

The FR metrics measure the quality of a test image based on a reference image (the reference image often refers to the original image). This kind of metric provides more-accurate assessment because the reference or the original image contains all of the information to be assessed. However, sometimes the reference or the original image is not available. Instead of using the entire reference image, the RR metrics use only the reduced representation of the reference image (e.g., mean, variance, reduced spatial representation, etc.) to assess the quality of a test image, which is compared to either the reduced representation of the test image or the entire test image. The NR metrics assess a test image without using any information of a reference image. These metrics exploit distortion based on *a priori* knowledge of the test image.

1.3 Full-Reference Metrics

There are many FR quality metrics for assessing the image quality of satellite images. These metrics are based on the calculation of errors between the reference and the test image, such as mean square error (MSE), root MSE,

SNR, etc. These metrics do not take into account the perceived visual quality of the test images; however, a set of FR metrics that consider perceived visual quality of the test images has been proposed.[9 13] One of these metrics is referred to as the Universal Image Quality Index (also called Q index). This metric is based on the philosophy that structural information is extracted from the viewing field of human eyes. As a consequence, a switch from error measurement to structural distortion measurement is performed. Many tested images have proven that the Q index can be profitably used to assess the quality of distorted images.[12] In this section, the FR metrics that do and do not take into account the perceived visual quality will be described in two separate subsections.

A test image can be a 2D panchromatic image, a set of multispectral images, or a 3D datacube. Let us assume that a test image is $\mathbf{I}(X, Y, \Lambda)$, and its reference image is $\mathbf{I}_r$ (x, y, Λ), each of which is a 3D datacube with a spatial size of N_x columns and N_y rows, and with N_λ spectral bands. If a test image is a single 2D image, N_λ is equal to 1. The datacubes (images) can also be written in a matrix form, where $I(x, y, \lambda)$ denotes the value of a pixel at x column and y row in the spectral band λ of the test datacube, and $I_r(x, y, \lambda)$ denotes the value of a pixel of the reference image at the same spatial location and the same spectral band as the test datacube. A spectrum profile $\mathbf{I}(x, y, \cdot)$, corresponding to a ground sample at location (x, y) of the datacube, is defined as

$$\mathbf{I}(x, y, \cdot) = \{I(x, y, \lambda) | 1 \leq \lambda \leq N_\lambda\}. \tag{1.1}$$

This spectrum profile is often referred to as a spectral vector whose length is N_λ. An image $\mathbf{I}(\cdot, \cdot, \lambda)$ at band λ of the datacube $\mathbf{I}(X, Y, \Lambda)$ is defined as

$$\mathbf{I}(\cdot, \cdot, \lambda) = \{I(x, y, \lambda) | 1 \leq x \leq N_x; 1 \leq y \leq N_y\}. \tag{1.2}$$

1.3.1 Conventional full-reference metrics

The following 13 metrics are widely used conventional FR metrics.

1.3.1.1 Mean-square error (MSE)

$$MSE = \frac{1}{N_x N_y N_\lambda} \sum_{x\ 1}^{N_x} \sum_{y\ 1}^{N_y} \sum_{\lambda\ 1}^{N_\lambda} [I(x, y, \lambda) \quad I_r(x, y, \lambda)]^2. \tag{1.3}$$

Root-mean-square error (RMSE)

$$RMSE = \sqrt{\frac{1}{N_x N_y N_\lambda} \sum_{x\ 1}^{N_x} \sum_{y\ 1}^{N_y} \sum_{\lambda\ 1}^{N_\lambda} [I(x, y, \lambda) \quad I_r(x, y, \lambda)]^2}. \tag{1.4}$$

1.3.1.2 Relative-mean-square error (ReMSE)

$$ReMSE = \frac{1}{N_x N_y N_\lambda} \sum_{x\ 1}^{N_x} \sum_{y\ 1}^{N_y} \sum_{\lambda\ 1}^{N_\lambda} \left[\frac{I(x,y,\lambda) \quad I_r(x,y,\lambda)}{I_r(x,y,\lambda)} \right]^2. \tag{1.5}$$

Root relative-mean-square error (RReMSE)

$$RReMSE = \sqrt{\frac{1}{N_x N_y N_\lambda} \sum_{x\ 1}^{N_x} \sum_{y\ 1}^{N_y} \sum_{\lambda\ 1}^{N_\lambda} \left[\frac{I(x,y,\lambda) \quad I_r(x,y,\lambda)}{I_r(x,y,\lambda)} \right]^2}. \tag{1.6}$$

1.3.1.3 Signal-to-noise ratio (SNR)

$$SNR = 10 \log_{10} \frac{Signal\text{-}power}{MSE}, \tag{1.7}$$

where

$$Signal\text{-}power = \frac{1}{N_x N_y N_\lambda} \sum_{x\ 1}^{N_x} \sum_{y\ 1}^{N_y} \sum_{\lambda\ 1}^{N_\lambda} I_r(x,y,\lambda)^2. \tag{1.8}$$

1.3.1.4 Peak signal-to-noise ratio (PSNR)

$$PSNR = 10 \log_{10} \frac{(Peak\text{-}value)^2}{MSE}, \tag{1.9}$$

where *Peak-value* is the maximum value of the reference image. The value of the upper limit of the quantization is also used sometimes, such as 255 for 8-bit quantization, 4095 for 12-bit quantization, and 65535 for 16-bit quantization, which is not recommended. This is because the maximum value of the real image data can often be much smaller than the upper limit of the quantization dynamic range for the purpose of preventing saturation caused by bright signal. If this is the case, the use of the upper-limit value would result in an artificially high PSNR.

1.3.1.5 Maximum absolute difference (MAD)

$$MAD = \max_{(x,y,\lambda)} \left\{ |I(x,y,\lambda) \quad I_r(x,y,\lambda)| \right\}. \tag{1.10}$$

MAD can be used to bind the error limits of any value of a test image related to the reference image. This property can be very useful in the case of local errors.

1.3.1.6 Percentage maximum absolute difference (PMAD)

$$PMAD = \max_{(x,y,\lambda)} \left\{ \frac{|I(x,y,\lambda) \quad I_r(x,y,\lambda)|}{I_r(x,y,\lambda)} \right\} \times 100\%. \tag{1.11}$$

PMAD provides more tolerance to individual larger errors of the test image due to the introduction of the normalization term.

1.3.1.7 Mean absolute error (MAE)

$$MAE = \frac{1}{N_x N_y N_\lambda} \sum_{x\ 1}^{N_x} \sum_{y\ 1}^{N_y} \sum_{\lambda\ 1}^{N_\lambda} \left| I(x,y,\lambda) \quad I_r(x,y,\lambda) \right|. \tag{1.12}$$

1.3.1.8 Correlation coefficient (CC)

$$CC(\lambda) = \frac{\sum_{x\ 1}^{N_x} \sum_{y\ 1}^{N_y} \left[I(x,y,\lambda) - \mu_{I(\cdot,\cdot,\lambda)} \right] \left[I_r(x,y,\lambda) - \mu_{I_r(\cdot,\cdot,\lambda)} \right]}{\sqrt{\sum_{x\ 1}^{N_x} \sum_{y\ 1}^{N_y} \left[I(x,y,\lambda) - \mu_{I(\cdot,\cdot,\lambda)} \right]^2} \sqrt{\sum_{x\ 1}^{N_x} \sum_{y\ 1}^{N_y} \left[I_r(x,y,\lambda) - \mu_{I_r(\cdot,\cdot,\lambda)} \right]^2}}, \tag{1.13}$$

where $\mu_{I(\ ,\ ,\lambda)}$ and $\mu_{I_r(\ ,\ ,\lambda)}$ are the means of band images $\mathbf{I}(\cdot,\cdot,\lambda)$ and $\mathbf{I}_r(\cdot,\cdot,\lambda)$ at band λ, respectively, and are defined as

$$\mu_{I(\ ,\ ,\lambda)} = \frac{1}{N_x N_y} \sum_{x\ 1}^{N_x} \sum_{y\ 1}^{N_y} I(x,y,\lambda), \tag{1.14}$$

$$\mu_{I_r(\ ,\ ,\lambda)} = \frac{1}{N_x N_y} \sum_{x\ 1}^{N_x} \sum_{y\ 1}^{N_y} I_r(x,y,\lambda). \tag{1.15}$$

$CC(\lambda)$ is the correlation coefficient between a band image at wavelength λ and its reference band image. The overall CC between a test datacube and its reference datacube is

$$CC = \frac{1}{N_\lambda} \sum_{\lambda\ 1}^{N_\lambda} |CC(\lambda)|. \tag{1.16}$$

For hyperspectral images, it is necessary to measure the spectral distortion. The following metrics are particular to hyperspectral images.

1.3.1.9 Mean-square spectral error (MSSE)

$$MSSE_{x,y} = \sqrt{\frac{1}{N_\lambda}\sum_{\lambda\ 1}^{N_\lambda}\left[I(x,y,\lambda) \quad I_r(x,y,\lambda)\right]^2}. \tag{1.17}$$

MSSE is the MSE between a spectrum of a test datacube at location (x,y) and the spectrum at the same location of the reference datacube.

Root-mean-square spectral error (RMSSE)

$$RMSSE_{x,y} = \sqrt{\frac{1}{N_\lambda}\sum_{\lambda\ 1}^{N_\lambda}\left[I(x,y,\lambda) \quad I_r(x,y,\lambda)\right]^2}. \tag{1.18}$$

1.3.1.10 Spectral correlation (SC)

The spectral correlation between a spectral vector $\mathbf{I}(x,y,\cdot)$ and its reference spectral vector $\mathbf{I}_r(x,y,\cdot)$ is

$$SC_{x,y} = \frac{\sigma_{I(x,y,\cdot)I_r(x,y,\cdot)}}{\sigma_{I(x,y,\cdot)}\sigma_{I_r(x,y,\cdot)}} = \frac{\sum_{\lambda=1}^{N_\lambda}\left[I(x,y,\lambda)-\mu_{I(x,y,\cdot)}\right]\left[I_r(x,y,\lambda)-\mu_{I_r(x,y,\cdot)}\right]}{(N_\lambda-1)\sigma_{I(x,y,\cdot)}\sigma_{I_r(x,y,\cdot)}}, \tag{1.19}$$

where $\mu_{I(x,y,\ )}$ and $\sigma^2_{I(x,y,\ )}$ are the mean and variance of the vector $I(x,y,\cdot)$, respectively, and are defined as

$$\mu_{I(x,y,\ )} = \frac{1}{N_\lambda}\sum_{\lambda\ 1}^{N_\lambda} I(x,y,\lambda), \tag{1.20}$$

$$\sigma^2_{I(x,y,\ )} = \frac{1}{N_\lambda}\sum_{\lambda\ 1}^{N_\lambda}\left[I(x,y,\lambda) \quad \mu_{I(x,y,\ )}\right]^2. \tag{1.21}$$

1.3.1.11 Spectral angle (SA)

A spectral angle represents the angle between two spectral vectors $(\mathbf{v}_1,\mathbf{v}_2)$ in an N_λ-dimensional space:

$$SA(\mathbf{v}_1,\mathbf{v}_2) = \cos^{\ 1}\left(\frac{\mathbf{v}_1\cdot\mathbf{v}_2}{\mathbf{v}_1\cdot\mathbf{v}_2}\right) = \cos^{\ 1}\left(\frac{\sum_{\lambda=1}^{N_\lambda}\left[v_1(\lambda)\cdot v_2(\lambda)\right]}{\sqrt{\sum_{\lambda=1}^{N_\lambda}v_1(\lambda)^2\sum_{\lambda=1}^{N_\lambda}v_2(\lambda)^2}}\right). \tag{1.22}$$

To measure the spectral angle between a spectrum $\mathbf{I}(x, y, \cdot) = \{I(x, y, \lambda) \mid 1 \leq \lambda \leq N_\lambda\}$ of a test datacube and the spectrum $\mathbf{I}_r(x, y, \cdot) = \{I_r(x, y, \lambda) \mid 1 \leq \lambda \leq N_\lambda\}$ at the same location of the reference datacube, Eq. (1.22) becomes

$$SA[I(x, y, \cdot), I_r(x, y, \cdot)] = \cos^{1}\left(\frac{\sum_{\lambda=1}^{N_\lambda}\left[I(x, y, \lambda) \cdot I_r(x, y, \lambda)\right]}{\sqrt{\sum_{\lambda=1}^{N_\lambda} I(x, y, \lambda)^2 \sum_{\lambda=1}^{N_\lambda} I_r(x, y, \lambda)^2}}\right). \quad (1.23)$$

A resulting value of Eq. (1.23), equal to zero, denotes no spectral distortion but possible radiometric distortion (e.g., the two spectral vectors are parallel but have different lengths). Spectral angles are measured in either degrees or radians and are usually averaged over the whole datacube to produce a global measurement of spectral distortion.

The maximum spectral angle (MSA) within a datacube is

$$MSA = \max_{x, y}\left\{SA[I(x, y, \cdot),\ I_r(x, y, \cdot)]\right\}. \quad (1.24)$$

The spectral angle mapper (SAM) is formed by calculating spectral angles for all of the spectra between a test datacube and the reference datacube. It is a 2D image:[14]

$$SAM = SA[I(x, y, \cdot), I_r(x, y, \cdot)] \mid 1 \leq x \leq N_x; 1 \leq y \leq N_y. \quad (1.25)$$

1.3.1.12 Maximum spectral information divergence (MSID)

$$MSID = \max_{x, y}\left\{\sum_{\lambda\ \ 1}^{N_\lambda}(\rho_{r\lambda} \quad \rho_\lambda)\mathrm{In}\left(\frac{\rho_{r\lambda}}{\rho_\lambda}\right)\right\}, \quad (1.26)$$

where

$$\rho_{r\lambda} = \frac{I_r(x, y, \lambda)}{\sum_{\lambda\ \ 1}^{N_\lambda}|I_r(x, y, \lambda)|} \text{ and} \quad (1.27)$$

$$\rho_\lambda = \frac{I(x, y, \lambda)}{\sum_{\lambda\ \ 1}^{N_\lambda}|I(x, y, \lambda)|}. \quad (1.28)$$

This metric is based on the Kullback–Leibler distance, which measures the distance between two spectra viewed as distributions.[15]

1.3.1.13 ERGAS for multispectral image after pan-sharpening

A metric has been proposed to measure the image quality of multispectral (MS) images after pan-sharpening. It is referred to as ERGAS (derived from its French name); it means "relative global error in synthesis " and is defined as[16]

$$ERGAS = 100\frac{d_h}{d_l}\sqrt{\frac{1}{N_\lambda}\sum_{\lambda\ 1}^{N_\lambda}\left[\frac{RMSE(\lambda)}{\mu_{I_r(\ ,\ ,\lambda)}}\right]^2}, \tag{1.29}$$

where d_h/d_l is the ratio between pixel sizes of the multispectral and pan images, e.g., 4:1 for IKONOS-2 and QuickBird images, $\mu_{I_r(\ ,\ ,\lambda)}$ is the mean of the λth band image of the reference, $RMSE(\lambda)$ is the RMSE between the λth pan-sharpened MS image and its reference image, and N_λ is the number of multispectral bands. An ERGAS value larger than 3 corresponds to fused products of low quality, whereas an ERGAS value smaller than 3 denotes a product of satisfactory quality or better.[16] ERGAS is a notable effort to integrate several measurements in a unique number; however, it does not consider CC, spectral distortion, or radiometric distortion.

1.3.2 Perceived-visual-quality-based full-reference metrics

1.3.2.1 Universal image-quality index

An objective image-quality index applicable to various image-processing applications was reported.[9] This image quality metric is designed by modeling any 2D image distortion as a combination of three factors: loss of correlation, luminance distortion, and contrast distortion. Let $\mathbf{v} = \{v_i \mid i = 1, 2, \ldots, N\}$ and $\mathbf{u} = \{u_i \mid i = 1, 2, \ldots, N\}$ be the test image and the reference image, respectively. The Q index is defined as

$$Q = \frac{4 \cdot \sigma_{uv} \cdot \bar{u} \cdot \bar{v}}{(\sigma_u^2 + \sigma_v^2)(\bar{u}^2 + \bar{v}^2)}, \tag{1.30}$$

in which σ_{uv} denotes the covariance between images $\mathbf{u}$ and $\mathbf{v}$, $\bar{u}$ and $\bar{v}$ are the means of images $\mathbf{u}$ and $\mathbf{v}$, and σ_u^2 and σ_v^2 are the variances of images $\mathbf{u}$ and $\mathbf{v}$. The dynamic range of Q is [–1, 1]. The best value of 1 is achieved if $\mathbf{u} = \mathbf{v}$, i.e., a test image is equal to the reference image for all pixels. The lowest value of 1 occurs when $v_i = 2\bar{u} \quad u_i$ for all $i = 1, 2, \ldots, N$. Equation (1.30) can be rewritten as the product of three components:

$$Q = \frac{\sigma_{uv}}{\sigma_u \cdot \sigma_v} \times \frac{2 \cdot \bar{u} \cdot \bar{v}}{\bar{u}^2 + \bar{v}^2} \times \frac{2 \cdot \sigma_u \cdot \sigma_v}{\sigma_u^2 + \sigma_v^2}. \tag{1.31}$$

The first component is the CC between the images $\mathbf{v}$ and $\mathbf{u}$. The second component is always less than or equal to 1 and sensitive to bias in the mean of $\mathbf{v}$ with respect to $\mathbf{u}$. The third component is also less than or equal to 1 and accounts for relative changes in contrast between $\mathbf{u}$ and $\mathbf{v}$. Relative change means that the contrast will not change if the two contrasts σ_u and σ_v are both multiplied by the same constant. To increase the discrimination capability of the three components of the Q metric, all statistics are calculated on suitable

$N \times N$ image blocks, and the resulting values of Q are averaged over the whole image to produce a global score. In this way the spatial variability of the test image can be better accounted for.

1.3.2.2 Multispectral image-quality index

The Q index has been widely used to assess the image quality of 2D images. It does not take into account spectral distortion when assessing the image quality of multispectral images after pan-sharpening. To overcome this limitation, an image-quality index designed for MS imagery that have four spectral bands has been developed based on the Q index.[17] It uses the theory of hypercomplex numbers, or quaternions.[18] This metric is referred to as $Q4$; it encapsulates both spectral and radiometric distortion measurements in a unique measurement, simultaneously accounting for local mean bias, changes in contrast, and loss of correlation of individual bands, together with spectral distortion. The $Q4$ index has been used to assess the pan-sharpened MS images.

Before describing the $Q4$ index, the fundamentals of the theory of a quaternion are briefly reviewed. A quaternion is a hypercomplex number that is represented in the following form:

$$z = a + \mathbf{i}b + \mathbf{j}c + \mathbf{k}d, \tag{1.32}$$

where a, b, c, and d are real numbers, and $\mathbf{i}$, $\mathbf{j}$, and $\mathbf{k}$ are imaginary units, such that

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = 1. \tag{1.33}$$

The noncommutative nature of a quaternion stems from the following further relationships among the imaginary units:

$$\begin{aligned} \mathbf{jk} &= \mathbf{i}; \quad \mathbf{kj} = \quad \mathbf{i}; \\ \mathbf{ki} &= \mathbf{j}; \quad \mathbf{ik} = \quad \mathbf{j}; \\ \mathbf{ij} &= \mathbf{k}; \quad \mathbf{ji} = \quad \mathbf{k}. \end{aligned} \tag{1.34}$$

Similar to complex numbers, the conjugate z^* of a quaternion z is defined by

$$z^* = a \quad \mathbf{i}b \quad \mathbf{j}c \quad \mathbf{k}d, \tag{1.35}$$

and the modulus by

$$|z| = \sqrt{z \cdot z^*} = \sqrt{a^2 + b^2 + c^2 + d^2}. \tag{1.36}$$

Given two quaternion random variables z_1 and z_2, the same as the complex variance, quaternion variance of z_1 and z_2 can be defined as[19]

$$\sigma_{z_1} = E\left[(z_1 \quad \overline{z}_1)^2\right] = E\left[{z_1}^2\right] \quad {\overline{z}_1}^2, \tag{1.37}$$

$$\sigma_{z_2} = E\left[(z_2 \quad \overline{z}_2)^2\right] = E\left[{z_2}^2\right] \quad {\overline{z}_2}^2. \tag{1.38}$$

Like complex covariance, quaternion covariance between z_1 and z_2 can be defined as

$$\sigma_{z_1 z_2} = E\left[(z_1 \quad \overline{z}_1)(z_2 \quad \overline{z}_2)^*\right] = E\left[z_1 z_2^*\right] \quad \overline{z}_1 \overline{z}_2^*, \tag{1.39}$$

where $\overline{z}_1 = E[z_1]$, $\overline{z}_2 = E[z_2]$, and $\overline{z}_2^* = (\overline{z}_2^*) = E[z_2^*]$. The quaternion CC between two variables z_1 and z_2 can be defined as the normalized covariance:

$$CC(z_1, z_2) = \frac{\sigma_{z_1 z_2}}{\sigma_{z_1} \cdot \sigma_{z_2}}. \tag{1.40}$$

The module of $CC(z_1, z_2)$ is a real value and represents an extension of CC suitable for assessing multivariable data having four components, such as four-band multispectral images. In the case of three components (as for a color image with three RGB bands), the real part of one quaternion can be set equal to zero.[20]

For MS imagery with four spectral bands (typically acquired in the R, G, B, and NIR wavelengths), let $\mathbf{I}(\cdot,\cdot,1)$, $\mathbf{I}(\cdot,\cdot,2)$, $\mathbf{I}(\cdot,\cdot,3)$, and $\mathbf{I}(\cdot,\cdot,4)$ denote the four band images. Similar to the Q index, $Q4$ consists of different factors accounting for the correlation, mean bias, and contrast variation of each band image. Therefore, its low value may detect when radiometric distortion is accompanied by spectral distortion. Both radiometric and spectral distortions may thus be encapsulated in a unique parameter. Let

$$\mathbf{z} = I(\cdot, \cdot, 1) + \mathbf{i}I(\cdot, \cdot, 2) + \mathbf{j}I(\cdot, \cdot, 3) + \mathbf{k}I(\cdot, \cdot, 4) \tag{1.41}$$

and

$$\mathbf{z}_r = I_r(\cdot, \cdot, 1) + \mathbf{i}I_r(\cdot, \cdot, 2) + \mathbf{j}I_r(\cdot, \cdot, 3) + \mathbf{k}I_r(\cdot, \cdot, 4) \tag{1.42}$$

denote the four-band test MS imagery and its reference imagery in quaternion variable form, respectively. The $Q4$ index is defined as

$$Q4 = \frac{4 \cdot |\sigma_{zz_r}| \cdot |\overline{z}| \cdot |\overline{z}_r|}{(\sigma_z^2 + \sigma_{z_r}^2)(\overline{z}^2 + \overline{z}_r^2)}. \tag{1.43}$$

The $Q4$ is a real value in the interval [0, 1], with 1 being the best value. It is an extension of Q suitable for assessing multivariable data having four components. Similar to Eq. (1.31), Eq. (1.43) can be written as the product of three components:

$$Q4 = \frac{|\sigma_{zz_r}|}{\sigma_z \cdot \sigma_{z_r}} \times \frac{2 \cdot |\overline{z}| \cdot |\overline{z}_r|}{\overline{z}^2 + \overline{z}_r^2} \times \frac{2 \cdot \sigma_z \cdot \sigma_{z_r}}{\sigma_z^2 + \sigma_{z_r}^2}. \tag{1.44}$$

The first component is the modulus of the hypercomplex CC between z and z_r. It is sensitive to both loss of correlation and spectral distortion between the

test MS imagery and its reference. The second and third components, respectively, measure mean bias and contrast changes on all four bands simultaneously. Again, like Eq. (1.31), ensemble expectations are calculated as averages on $N \times N$ image blocks. $Q4$ will thus depend on N as well. The $Q4$ index, together with the specific block size N, is denoted as $Q4_N$. $Q4_N$ is eventually averaged over the whole image to yield the *global* score index. Alternatively, the minimum attained over the whole image can represent a measure of *local* quality. The values taken for $Q4$ by Eq. (1.43) are invariant to permutation [$\mathbf{I}(\cdot,\cdot,1)$, $\mathbf{I}(\cdot,\cdot,2)$, $\mathbf{I}(\cdot,\cdot,3)$, and $\mathbf{I}(\cdot,\cdot,4)$] in both z and z_r in Eqs. (1.41) and (1.42). This happens because the order of components does not affect the magnitude of Eq. (1.40).

1.3.2.3 Quality index for multi- or hyperspectral images

The image-quality metric Q index was developed for monoband images that have been extended by the $Q4$ index. However, the $Q4$ index can be applied to assess multispectral images having only four-band images. An image quality index referred to as $Q2^n$, suitable to assess the quality of images having an arbitrary number of spectral bands, has been proposed.[21] The $Q2^n$ index is also derived from the theory of hypercomplex numbers,[18] particularly of 2^n-ons.[22] The new index consists of different factors to take into account correlation, the mean of each spectral band, intraband local variance, and the spectral angle. Thus, both intra- and interband (spectral) distortions are considered by a single index $Q2^n$. The index can be easily calculated and applied to assess the quality of multispectral imagery with any number of spectral bands and hyperspectral datacubes when their reference datacubes are available.

The 2^n-ons can be defined recursively in terms of the 2^{n-1}-ons. A 2^n-on is a hypercomplex number that can be represented as

$$\mathbf{z} = z_0 + z_1 i_1 + z_2 i_2 + \ldots + z_{2^n-1} i_{2^n-1}, \tag{1.45}$$

where $z_0, z_1, z_2, \ldots, z_{2^n-1}$ are real numbers, and $i_1, i_2, \ldots, i_{2^n-1}$ are hypercomplex unit vectors. Analogous to complex numbers, the conjugate $\mathbf{z}^*$ is given by

$$\mathbf{z}^* = z_0 - z_1 i_1 - z_2 i_2 - \ldots - z_{2^n-1} i_{2^n-1}, \tag{1.46}$$

and the modulus by

$$|\mathbf{z}| = \sqrt{z_0^2 + z_1^2 + z_2^2 + \ldots + z_{2^n-1}^2}. \tag{1.47}$$

Given two 2^n-on hypercomplex random variables $\mathbf{z}$ and $\mathbf{z}_r$, the hypercomplex variances σ_z and σ_{z_r}, the covariance σ_{zz_r} between $\mathbf{z}$ and $\mathbf{z}_r$, and the correlation coefficient $CC(z, z_r)$ between $\mathbf{z}$ and $\mathbf{z}_r$ can be similarly defined as in Eqs. (1.37)–(1.40).

The $Q2^n$ index between a test datacube and its reference datacube calculated in a block of spatial size $N \times N$ is given by

$$Q2^n_{N\times N} = \frac{|\sigma_{zz_r}|}{\sigma_z \cdot \sigma_{z_r}} \times \frac{2 \cdot |\bar{z}| \cdot |\bar{z}_r|}{\bar{z}^2 + \bar{z}_r^2} \times \frac{2 \cdot \sigma_z \cdot \sigma_{z_r}}{\sigma_z^2 + \sigma_{z_r}^2}. \tag{1.48}$$

$Q2^n$ is obtained by averaging the magnitudes of all $Q2^n_{N\times N}$ over the entire spatial area of the datacube:

$$Q2^n = E\left[|Q2^n_{N\times N}|\right]. \tag{1.49}$$

The more a $Q2^n$ value approaches unity, the higher the radiometric and spectral quality of the fused datacube becomes. As an extension of the $Q4$ index, the $Q2^n$ index also produces a real value in the interval [0, 1], with 1 being the best value. Both spatial and spectral distortions are assessed by a single $Q2^n$ index. The correlation, the mean of each spectral band, and the intraband local variance are also taken into account for each band, as shown in Eq. (1.48). The spectral angles between a test datacube and the reference are also assessed by the $Q2^n$ index, which is reflected by the modulus of hypercomplex CC of multivariate data.

1.3.2.4 Structural similarity index

Under the assumption that human visual perception is highly adapted for extracting structural information from an image, an image-quality metric based on the degradation of structural information has been developed. This quality metric is referred to as the structural similarity (SSIM) index.[23]

The MSE between a test image and its reference image is the simplest and most widely used distortion measure. But two test images with the same MSE related to their references may have very different types of errors, some of which are much more visible than others. Most perceptual image-quality assessment approaches proposed in the literature attempt to weight different aspects of the error signal according to their visibility, as determined by psychophysical measurements in humans or physiological measurements in animals.[24–27]

The SSIM index considers the structural similarity measurement between a test image $\mathbf{u}$ and its reference $\mathbf{v}$ in three components—luminance, contrast, and structure—and combines them to yield an overall similarity measure. The luminance function is based on the means of the images μ_u and μ_v, and is defined as

$$l(\mathbf{u}, \mathbf{v}) = \frac{2\mu_u\mu_v + C_1}{\mu_u^2 + \mu_v^2 + C_1}, \tag{1.50}$$

where the constant $\alpha > 0$ is included to avoid instability when $\mu_u^2 + \mu_v^2$ is very close to zero. $C_1 = (K_1 L)^2$, where L is the dynamic range of the pixels' values (255 for 8-bit quantization), and $K_1 \ll 1$ is a small constant. Similar

considerations also apply to contrast and structure components that are described later.

The contrast function takes a similarity from

$$c(\mathbf{u},\mathbf{v}) = \frac{2\sigma_u\sigma_v + C_2}{\sigma_u^2 + \sigma_v^2 + C_2}, \tag{1.51}$$

where $C_2 = (K_2L)^2$, and $K_2 \ll 1$. The structure function is defined as follows:

$$s(\mathbf{u},\mathbf{v}) = \frac{\sigma_{uv} + C_3}{\sigma_u^2\sigma_v^2 + C_3}. \tag{1.52}$$

As in the luminance and contrast measures, a small constant C_3 is included. Finally, the three components of Eqs. (1.50)–(1.51) are combined to form the structural similarity measure, the SSIM index, between a test image $\mathbf{v}$ and its reference image $\mathbf{u}$:

$$SSIM = [l(u,v)]^{\alpha} \cdot [c(u,v)]^{\beta} \cdot [s(u,v)]^{\gamma}, \tag{1.53}$$

where $\alpha > 0$, $\beta > 0$, and $\gamma > 0$ are parameters used to adjust the relative importance of the three components. In order to simplify the expression, it is set to $\alpha = \beta = \gamma = 1$ and $C_3 = C_2/2$. This results in a specific form of the SSIM index:

$$SSIM(\mathbf{u},\mathbf{v}) = \frac{(2\mu_u\mu_v + C_1)(2\sigma_{uv} + C_2)}{(\mu_u^2 + \mu_v^2 + C_1)(\sigma_u^2 + \sigma_v^2 + C_2)}. \tag{1.54}$$

The Q index defined in Eq. (1.31) corresponds to the special case that $C_1 = C_2 = 0$, which produces unstable results when either $(\mu_u^2 + \mu_v^2)$ or $(\sigma_u^2 + \sigma_v^2)$ is very close to zero.

For image quality assessment, it is useful to apply the SSIM index locally rather than globally. This is because (1) image statistical features are usually highly spatially nonstationary; (2) image distortions, which may or may not depend on the local image statistics, may also be space-variant; (3) at typical viewing distances, only a local area in the image can be perceived with high resolution by the human observer at one time instance; and (4) localized quality measurement can provide a spatially varying quality map of the image, which delivers more information about the quality degradation of the image and may be useful in some applications. As suggested in the literature,[9,23] the local statistics μ_u, σ_u, and σ_{uv} are computed within a local 8×8 square window that moves pixel-by-pixel over the entire image.

The overall quality measure of the entire image is obtained by averaging the local SSIMs of the blocks, which is referred to as the mean SSIM (MSSIM) index:

$$MSSIM(\mathbf{u},\mathbf{v}) = \frac{1}{M}\sum_{k\ 1}^{M} SSIM(\mathbf{u}_k,\mathbf{v}_k), \tag{1.55}$$

where **u** and **v** are the test and the reference images, respectively; $\mathbf{u}_k$ and $\mathbf{v}_k$ are the image contents at the kth local block; and M is the number of local blocks of the image. Depending on the application, it is also possible to compute a weighted average of the different samples in the SSIM index map. For example, region-of-interest (ROI) image-processing systems may give different weights to different segmented regions in the image.

1.3.2.5 Visual information fidelity

Visual information fidelity (VIF) is an image quality metric that takes into account the human visual system (HVS). It assesses the visual quality of a test image in terms of the amount of image information that a human brain could extract from the test image related to the amount of information that the human brain could extract from the reference image. The metric is derived from a quantification of two mutual information quantities: the mutual information between the input and the output of the HVS channel when no distortion channel is present (referred to as reference image information), and the mutual information between the input of the distortion channel and the output of the HVS channel for a test image.[28]

VIF is defined as the ratio of the test image information to the reference image information:

$$VIF = \frac{\sum_{j \epsilon subbands} I(\vec{C}_{N,j}; \vec{F}_{N,j} | s_{N,j})}{\sum_{j \epsilon subbands} I(\vec{C}_{N,j}; \vec{E}_{N,j} | s_{N,j})}, \tag{1.56}$$

where $I(\vec{C}_{N,j}; \vec{F}_{N,j} | s_{N,j})$ and $I(\vec{C}_{N,j}; \vec{E}_{N,j} | s_{N,j})$ represent the information that could be ideally extracted by the brain from a particular subband in the test and the reference images, respectively.

$I(\vec{C}_{N,j}; \vec{E}_{N,j} | s_{N,j})$ is called the reference image information. Intuitively, visual quality should relate to the amount of image information that the brain could extract from the test image relative to the amount of information that the brain could extract from the reference image. For example, if the information that could be extracted from the test image is 2.0 bits per pixel, and if the information that could be extracted from the corresponding reference image is 2.1 bits per pixel, then the brain could recover most of the information content of the reference image from the test image. By contrast, if the corresponding reference image information was, say, 5.0 bits per pixel, then 3.0 bits of information have been lost to the distortion channel, and the visual quality of the test image should be inferior.

The VIF given in Eq. (1.56) is computed for a collection of $N \times M$ wavelet coefficients from each subband that could either represent an entire subband of an image or a spatially localized region of subband coefficients. In the former case, VIF is one number that quantifies the information fidelity for the entire image, whereas in the latter case, a sliding-window approach could be

used to compute a quality map that could visually illustrate how the visual quality of the test image varies over space.

VIF has a number of interesting features: (1) Note that it is bounded below by zero [such as when $I(\vec{C}_N; \vec{F}_N | s_N = 0)$ and $I(\vec{C}_N; \vec{F}_N | s_N \neq 0)$], which indicates that all information about the reference image has been lost in the distortion channel. (2) When the image is not distorted at all, and VIF is calculated between the reference image and its copy, VIF is exactly unity. Thus, for all practical distortion types, VIF will lie in the interval [0, 1]. (3) VIF has a distinction over traditional quality assessment methods—a linear contrast enhancement of the reference image that does not add noise to it will result in a VIF value larger than unity, thereby signifying that the enhanced image has a superior visual quality than the reference image. It is a common observation that contrast enhancement of images increases their perceptual quality unless quantization, clipping, or display nonlinearity add additional distortion. Contrast enhancement theoretically results in a higher SNR at the output of the HVS neurons, thereby allowing the brain to be better able to discriminate between objects present in the visual signal. The VIF is able to capture this improvement in visual quality. Figure 1.1 shows an example of an original image, its contrast-enhanced image, and its JPEG-compressed image.

1.4 Reduced-Reference Metrics

Reduced-reference (RR) image-quality metrics are designed to measure the image quality of test images with only partial information about the reference images. RR methods are useful in a number of applications. For example, RR metrics are used to measure the visual quality of hyperspectral images after undergoing spatial-resolution enhancement. These metrics can measure the visual quality of hyperspectral images whose FR image is not available but where the low-spatial-resolution reference image is available. A FR metric requires that a test image and its reference image have the same size. After spatial-resolution enhancement of hyperspectral images, the size of the enhanced images is larger than that of the original image. Thus, the FR metric cannot be used. A common approach in practice is to first downsample an original image to a low-resolution image and then to spatially enhance the downsampled, low-resolution image using an enhancement technique. In this way, the original image has the same size as the enhanced image, and the FR metric can be applied to both of them. However, this common approach can never directly assess the image quality of the spatially enhanced image that is produced directly from the original image.

Another application involves real-time visual communication systems, where a RR metric is used to track image quality degradations and control the streaming resources. The system includes a feature extraction process at the

(a)

(b)

(c)

Figure 1.1 An example of an image after linear contrast enhancement and compression: (a) original image (VIF=1.0), (b) contrast-enhanced image (VIF=1.12), and (c) JPEG-compressed image with a compression ratio of 118:1 (VIF=0.15).

sender side, and a feature-extraction and RR quality-analysis process at the receiver side. The image quality at the receiver side is evaluated using the RR metric. The extracted RR features usually have a much-lower datarate than the image data and are typically transmitted to the receiver through an ancillary channel.

1.4.1 Four RR metrics for spatial-resolution-enhanced images

The PSNR, Q index, MSSIM, and VIF are widely used FR metrics in image processing, as described in Sections 1.3.2.1 and 1.3.2.2. These four FR metrics have been selected to derive their corresponding RR metrics to assess the image quality of a spatial-resolution-enhanced image.[29] They are derived as follows:

Let the size of the low-spatial-resolution image **f** be $P \times Q$, and the size of the corresponding spatial-resolution-enhanced image **g** be $2P \times 2Q$. This means that the spatial resolution of image **f** is enhanced at a factor of 2×2. The following four images (downsampled at a factor of 2×2) can be defined as

$$g_{11} = g(1:2:2P,\ 1:2:2Q), \tag{1.57}$$

$$g_{12} = g(1:2:2P,\ 2:2:2Q), \tag{1.58}$$

$$g_{21} = g(2:2:2P,\ 1:2:2Q), \tag{1.59}$$

$$g_{22} = g(2:2:2P,\ 2:2:2Q), \tag{1.60}$$

where $g(i:2:2P, j:2:2Q)$, $(i=1,2;\ j=1,2)$ is a matrix that starts at the pixel (i,j) of image **g** and extracts every other pixel in **g** along both the x and the y directions with a step of 2. Because the low-spatial-resolution image **f** and the images $g_{i,j}$ $(i,j=1,2)$ have the same image size, any FR metrics can be used to measure the image quality between them. The following four RR metrics are defined:

$$PSNR(f;g) = \frac{1}{4}\sum_{i\ 1}^{2}\sum_{j\ 1}^{2} PSNR(f;g_{ij}), \tag{1.61}$$

$$Q(f;g) = \frac{1}{4}\sum_{i\ 1}^{2}\sum_{j\ 1}^{2} Q(f;g_{ij}), \tag{1.62}$$

$$MSSIM(f;g) = \frac{1}{4}\sum_{i\ 1}^{2}\sum_{j\ 1}^{2} MSSIM(f;g_{ij}), \tag{1.63}$$

$$VIF(f;g) = \frac{1}{4}\sum_{i\ 1}^{2}\sum_{j\ 1}^{2} VIF(f;g_{ij}). \tag{1.64}$$

The previous four RR metrics are derived for a particular spatial-resolution enhancement factor 2×2; it is easy to extend it to another spatial-resolution enhancement factor $M \times N$, where both M and N are positive integers.

A number of experiments were conducted to demonstrate the feasibility of the proposed RR metrics.[29] Three hyperspectral datacubes were tested, and the 2D band images of the datacubes were used to test the proposed RR metrics. The first hyperspectral datacube was acquired using AVIRIS[30] in the Cuprite mining district of Nevada in 1997. The second hyperspectral datacube was acquired using the airborne SWIR Full-Spectrum Imager II (SFSI-II).[31] The datacube was collected over Key Lake in northern Saskatchewan, Canada to study the capability of imaging spectrometers in identifying uranium mines and associated activities. The datacube was acquired with a ground sample distance (GSD) of 3.19 m $\times$ 3.13 m. The size of the datacube is 1090 lines by 496 pixels by 240 bands. The third hyperspectral datacube was also collected using SFSI-II to study target detection from SWIR hyperspectral imagery. The GSD of the datacube is 2.20 m $\times$ 1.85 m, and the size of the datacube is 140 lines by 496 pixels by 240 bands. Synthetic targets with different materials and sizes were deployed in a mixture of sand and low-density grass cover within the scene of the datacube.

Iterative back-projection (IBP)[32,33] and bilinear interpolation were used to enhance the spatial resolution of band images #16, #50, and #13 in the experiments for the Cuprite, Key Lake, and Target datacubes, respectively. In the experiments, these four FR metrics were compared to their corresponding RR metrics. Table 1.2 lists the experimental results of the metrics applied to the spatial-resolution-enhanced images by using IBP and interpolation.

For the FR metrics, a test image is first downsampled at a factor of 2×2 and then spatially enhanced at a factor of 2×2 in order to satisfy the requirement of the processed image having the same size as the reference image. For the proposed RR metrics, an original test image is spatially enhanced at a factor of 2×2 without prior downsampling. From the table, it can be seen that the proposed RR metrics measure the image quality of the spatial-resolution-enhanced images very well, and they are consistent with the corresponding FR metrics. This indicates that the proposed RR metrics are reliable metrics for measuring the quality of the spatial-resolution-enhanced images.

Table 1.2 Experimental results of four FR and four RR metrics of the test images that are spatially enhanced by using the IBP and interpolation methods.

Datacube	Spatial Enhancement Method	Full-Reference Metrics				Reduced-Reference Metrics			
		PSNR	Q	MSSIM	VIF	PSNR	Q	MSSIM	VIF
Cuprite	IBP	36.51	0.82	0.91	0.69	43.82	0.97	0.99	0.87
	Interpolation	35.67	0.78	0.90	0.48	38.37	0.92	0.96	0.75
Key Lake	IBP	34.87	0.75	0.89	0.77	40.11	0.96	0.98	0.87
	Interpolation	32.41	0.70	0.87	0.54	34.59	0.87	0.94	0.78
Target	IBP	53.33	0.78	0.99	0.77	61.67	0.97	1.00	0.98
	Interpolation	53.14	0.74	0.99	0.65	56.35	0.89	1.00	0.88

1.4.2 RR metric using the wavelet-domain natural-image statistic model

A RR image-quality metric has been proposed[34] that is based on a natural-image statistic model in the wavelet transform domain. A Kullback–Leibler distance[35] between the marginal probability distributions of wavelet coefficients of a test image and its reference image is introduced as a measure of image distortion. A generalized Gaussian model is employed to summarize the marginal distribution of wavelet coefficients of the reference image so that only a relatively small number of RR features are required to evaluate the quality of a test image. This metric is easy to implement and computationally efficient.

The development of this RR metric was motivated by the fact that wavelet transforms provide a convenient means for locally representing signals in space and frequency simultaneously. Wavelet transforms have been widely used to model the processing of visual systems and have become the preferred form of representations for many image processing and computer vision algorithms. It was demonstrated that the histogram of the wavelet transform coefficients calculated from the horizontal subbands of an original image can be well fitted with a generalized Gaussian density model. It has been observed that the marginal histogram distribution of the wavelet coefficients of subbands changes in different ways for different types of image distortions. This histogram distribution change can be used as a clue to assess image quality.

Let $q(x)$ and $p(x)$ denote the probability density functions of the wavelet coefficients in the same subband of a test image and its reference image, respectively. Let $\mathbf{x} = \{x_1, \ldots, x_N\}$ be a set of N randomly and independently selected coefficients. The Kullback–Leibler distance between $q(x)$ and $p(x)$ are estimated as

$$\hat{d}(p||q) = d(p_m||q) \quad d(p_m||p) \tag{1.65}$$

$$= \int p_m(x)\log\frac{p(x)}{q(x)}dx, \tag{1.66}$$

where $p_m(x)$ is a two-parameter generalized Gaussian density, which can well fit the marginal distribution of the coefficients in individual wavelet subbands and is defined as

$$p_m(x) = \frac{\beta}{2\alpha\Gamma\left(\frac{1}{\beta}\right)}e^{-(|x/\alpha|)^{\beta}}. \tag{1.67}$$

In Eq. (1.67), $\Gamma(a) = \int t^{a-1}e^{-t}dt$ (for $a > 0$) is the gamma function. This model provides a very efficient means to summarize the wavelet coefficient histogram of the reference image so that only two model parameters $\{\alpha, \beta\}$ need to be transmitted to the receiver. The estimation error of $\hat{d}(p||q)$ is

$$\begin{aligned} e &= d(p||q) - \hat{d}(p||q) \\ &= d(p||q) - [d(p_m||q) - d(p_m||p)] \\ &= \int [p(x) - p_m(x)]\log\frac{p(x)}{q(x)}dx. \end{aligned} \tag{1.68}$$

This error is small when $p_m(x)$ and $p(x)$ are close, which is true for typical natural images. With the additional cost of sending one more parameter $d(p_m||p)$, Eq. (1.66) not only gives a more-accurate estimator of $d(p||q)$ but also provides the useful feature that when there is no distortion between the original and received images [which implies $p(x) = q(x)$ for all x], both the distortion measure $d(p||q)$ and the estimated distortion measure $\hat{d}(p||q)$ are exactly zero. The RR metric between a distorted test image and the reference image is defined as

$$D = \log_2\left[1 + \frac{1}{D_0}\sum_{k=1}^{K}|\hat{d}^k(p^k||q^k)|\right], \tag{1.69}$$

where K is the number of wavelet coefficient subbands, p^k and q^k are the probability density functions of the kth subbands in the reference image and test image, respectively, $\hat{d}^k$ is the estimation of the Kullback–Leibler distance between p^k and q^k, and D_0 is a constant used to control the scale of the distortion measure.

The steps for calculating the D index are as follows. For the reference image, a three-level, four-orientation, steerable pyramid wavelet transform[21] is applied to the reference image to decompose the image into 12 oriented subbands (four for each level) and a high-pass and a low-pass residual subband. Six (two for each level) of the 12 oriented subbands are selected for extracting the features (i.e., the reduced reference). Selection of a subset of all the subbands reduces the datarate of RR features, as it was observed by the

authors that selecting the other six oriented subbands or all of the 12 oriented subbands gives similar overall performance for image quality prediction. For each selected subband, the histogram of the wavelet coefficients is computed, and its feature parameters $\{\alpha, \beta, d(p_m||p)\}$ are then estimated using a gradient descent algorithm to minimize the Kullback–Leibler distance between $p_m(x)$ and $p(x)$. This results in a total of 18 extracted scalar features for the reference image that will be used for RR image-quality assessment.

For the test image, the same wavelet transform as the reference is first applied to the image, and the coefficient histograms of the corresponding subbands are computed. To evaluate $d(p_m||q)$ at each subband, the subband histogram is compared with the histogram calculated from the corresponding RR features $\{\alpha, \beta\}$ about the reference image. The third RR feature, $d(p_m||p)$, is then subtracted from this quantity [as of Eq. (1.65)] to estimate $d(p||q)$. Finally, the Kullback–Leibler distances evaluated at all subbands are combined using Eq. (1.69) to produce a single distortion measure D.

1.5 No-Reference Metrics

When the reference image is unavailable, no-reference (NR) metrics are the only choice for assessing image quality. The main philosophy of NR quality assessment is blind distortion measurement. In NR quality assessment, an assessment algorithm does not have access to the reference image, and only the test image could be processed to assess its quality. From an application perspective, NR methods are more desirable than FR methods. Unfortunately, the NR quality assessment problem is largely unsolved, with limited success achieved by restricting the scope of the algorithms to specific distortion types, such as blocking due to block-based compression algorithms, or blurring, etc. Only a handful of NR methods have been proposed in the literature, and they are mostly designed for the blocking artifact in compressed images and videos. This section describes three statistic-based NR metrics for compressed images, a NR metric for compressed images using JPEG, and NR metrics for pan-sharpened multispectral images.

1.5.1 Statistic-based methods

Three statistic-based NR metrics are used in data compression. They are entropy, coding gain, and energy compaction.

1.5.1.1 Entropy

In information theory, entropy is a measure of the uncertainty in a random variable.[36] In this context, the term usually refers to the Shannon entropy, which quantifies the expected value of the information contained in an image. Entropy is typically measured in bits.[37] Shannon entropy is the average unpredictability in a random variable, which is equivalent to its information

content.[38] Shannon entropy provides an absolute limit on the best possible lossless encoding or compression of any communication, assuming that the communication may be represented as a sequence of independent and identically distributed random variables.

Shannon's source coding theorem shows that, in the limit, the average length of the shortest possible representation to encode the messages in a given alphabet is their entropy divided by the logarithm of the number of symbols in the target alphabet. Entropy is often used to assess the effectiveness of a lossless compression algorithm and to estimate the compressibility of an image for lossy compression. For a lossless compression algorithm, the closer the bitrate of the compressed image to the entropy of the original image is, the more effective the compression algorithm. For a lossy compression algorithm, the lower the entropy of an image is, the more compressible the image.

If outputs from a discrete source (e.g., a data series, an image, a datacube) are independent and identically distributed with probabilities $p_1, p_2, \ldots, p_{(2n \quad 1)}$, then the source has entropy:

$$H = \sum_{i \quad 0}^{2^n \quad 1} p_i \log_2 p_i. \tag{1.70}$$

If the source outputs are not independent, then the entropy depends on higher-order probability distributions.

1.5.1.2 Energy compaction

Energy compaction is defined as the ratio of the arithmetic mean (AM) of the variances of an image to the geometric mean (GM) of the variances of the image:[39]

$$EC = \frac{\frac{1}{N}\sum_{i \quad 0}^{N \quad 1} \sigma_i^2}{\sqrt{\prod_{i \quad 0}^{N \quad 1} \sigma_i^2}}, \tag{1.71}$$

where σ_i^2 is the variance of the image, and N is the total number of pixels of the image. The AM/GM ratio is always greater than or equal to 1. When an image has a higher AM/GM ratio, there is a higher amount of energy compaction in the image, which is favorable for data compression.[40]

1.5.1.3 Coding gain

The coding gain is defined as the mean-squared reconstruction error in pulse code modulation (PCM) coding divided by the mean-squared reconstruction error in the transform coding $\mathbf{T}$:[41]

$$CG = \frac{\sum_{i \quad 0}^{N \quad 1} (x_i \quad x_{PCMi})^2}{\sum_{i \quad 0}^{N \quad 1} (x_i \quad x_{Ti})^2}, \tag{1.72}$$

where x_i is a pixel at location i of the original image, x_{PCMi} is the pixel at location i by PCM coding, and x_{Ti} is the pixel at location i after transform coding **T**.

1.5.2 NR metric for compressed images using JPEG

These NR algorithms[42 48] measure the blocking artifact for video signals and compressed images using JPEG. They assume that the blocking edge occurs every eight pixels and that the measure of blocking artifact is the luminance-weighted norm of the pixel difference across block boundaries. These blocking metrics measure the average gray-level slope at the block boundary adjusted by the average gray-level slope at other locations. These metrics measure several distortions present in an image blindly. These distortions include global blur based on assumed Gaussian blurs of step edges, additive white and impulse noise based on local smoothness violation, blocking artifact based on simple block-boundary detection, and ringing artifact based on anisotropic diffusion.[44,45] These blocking-artifact metrics also assess video and images in the frequency domain. These measures assume that the presence of a blocky signal will introduce spikes in the PSD function of the edge-enhanced image. The strength of these spikes quantifies the strength of the blocking artifact.[46,47]

JPEG is a block-based lossy image coding technique that uses the discrete cosine transform (DCT). It is lossy because of the quantization operation applied to the DCT coefficients in each 8×8 coding block. Both blur and blockiness can be created during quantization. The blur is mainly due to the loss of high-frequency DCT coefficients, which smoothes the image signal within each block. Blockiness occurs due to the discontinuity at block boundaries, which is generated because the quantization in JPEG is block-based, and the blocks are quantized independently.

One effective way to examine both blur and blockiness is to transform the signal into the frequency domain. The blockiness can be easily identified by the peaks at the several feature frequencies, and the blur is also characterized by the energy shifting from high-frequency to low-frequency bands. A disadvantage of the frequency domain method is the involvement of the fast Fourier transform (FFT), which has to be calculated many times for each image and is therefore expensive. FFT also requires more storage space because it cannot be computed locally.

A computationally inexpensive and memory-efficient method has been proposed to estimate the blockiness feature by calculating the difference signal within a line horizontally and between lines vertically.[48] Let $u(x, y)$ denote a test image for $x\epsilon[1, N_x]$ and $y\epsilon[1, N_y]$; a difference signal along each horizontal line of the image can be calculated as

$$d_h(x, y) = u(x + 1, y) \quad u(x, y),\ x\epsilon[1, N_x \quad 1]. \tag{1.73}$$

The blockiness is estimated as the average differences across block boundaries:

$$B_h = \frac{1}{N_y(N_x/8 \quad 1)} \sum_{j \ 1}^{N_y} \sum_{i \ 1}^{N_x/8 \ 1} |d_h(8i,j)|. \tag{1.74}$$

The activity image signal can be measured using two factors. The first activity measure is the average absolute difference between in-block image samples:

$$A_h = \frac{1}{7}\left[\frac{8}{N_y(N_x \quad 1)} \sum_{j \ 1}^{N_y} \sum_{i \ 1}^{N_x \ 1} |d_h(i,j) \quad B_h|\right]. \tag{1.75}$$

The second activity measure is the zero-crossing (ZC) rate. The horizontal zero-crossing rate can be estimated as

$$Z_h = \frac{1}{N_y(N_x \quad 2)} \sum_{j \ 1}^{N_y} \sum_{i \ 1}^{N_x \ 2} z_h(x,y), \tag{1.76}$$

where

$$z_h(x,y) = \begin{cases} 1 & \text{horizontal } ZC \text{ at } d_h(x,y) \\ 0 & \text{otherwise.} \end{cases} \tag{1.77}$$

Using similar methods for the horizontal features, the vertical features B_v, A_v, and Z_h can also be calculated. Finally, the overall features are given by

$$B = \frac{B_h \quad B_v}{2}, \quad A = \frac{A_h \quad A_v}{2}, \quad Z = \frac{Z_h \quad Z_v}{2}. \tag{1.78}$$

There are many different ways to combine the features to constitute a quality assessment model. The proposed method that provides good prediction performance is given by

$$S = \alpha + \beta B^{\gamma 1} B^{\gamma 2} Z^{\gamma 3}, \tag{1.79}$$

where α, β, γ^1, γ^2, and γ^3 are the model parameters that need be estimated with the subjective test image, such as mean opinion score.[48] Figure 1.2 shows the flowchart for calculating this S score.

1.5.3 NR metric for pan-sharpened multispectral image

A NR metric based on the Q index for a pan-sharpened multispectral image has been reported.[49] The Q index values between any couple of multispectral band images are calculated before and after fusion and used to define a measurement of spectral distortion. Similarly, Q index values between each multispectral band image and the pan image are calculated before and after

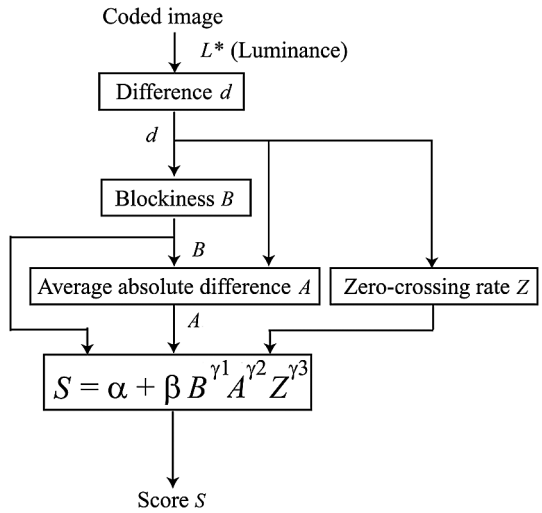


Figure 1.2 Flowchart for calculating NR metric S score.

fusion to yield a measurement of spatial distortion. The rationale is that such Q index values should be unchanged after fusion, that is, when the spectral information is translated from the coarse scale of the multispectral data to the fine scale of the pan image.

1.5.3.1 Spectral distortion index

A spectral distortion index is derived from the difference of interband Q index values calculated from the fused multispectral band images, indicated as $\{\hat{M}_\lambda\}$ $\lambda = 1, 2, \ldots, N_\lambda$, and from the original multispectral band images, $\{M_\lambda\}$ $\lambda = 1, 2, \ldots, N_\lambda$. The terms $Q(\hat{M}_i, \hat{M}_j)$ and $Q(M_i, M_j)$ can be grouped into two $N_\lambda \times N_\lambda$ matrices. The two matrices are symmetrical, and the values on the main diagonal are all equal to 1. A spectral distortion index, referred to as D_λ, is calculated as

$$D_\lambda = \left(\frac{1}{N_\lambda(N_\lambda \quad 1)} \sum_{i \ 1}^{N_\lambda} \sum_{\substack{j \ 1 \\ i/j}}^{N_\lambda} |Q(\hat{M}_i, \hat{M}_j) \quad Q(M_i, M_j)| \right)^{\frac{1}{p}}, \qquad (1.80)$$

with p being a positive integer exponent chosen to emphasize large spectral differences: for $p = 1$, all differences are equally weighted; as p increases, larger components are given more relevance. The D_λ is proportional to the p-norm of the difference matrix, being equal to 0, if and only if the two matrices are identical. If negative values of $Q(\hat{M}_i, \hat{M}_j)$ and $Q(M_i, M_j)$, caused by

anticorrelated band images, are clipped below zero, then D_λ is always lower than or equal to one.

1.5.3.2 Spatial distortion index

A spatial distortion index is calculated as

$$D_s = \left(\frac{1}{N_\lambda} \sum_{i\ 1}^{N_\lambda} |Q(\hat{M}_i, P) \quad Q(M_i, P')| \right)^{\frac{1}{q}}. \tag{1.81}$$

In Eq. (1.81), P is the panchromatic image, and P' is a spatially degraded version of the panchromatic image obtained by filtering with a low-pass filter having normalized frequency cutoff at the resolution ratio between multispectral and panchromatic, followed by decimation. Similarly, D_s is proportional to the q-norm of the difference vector, where q can be chosen so as to emphasize higher difference values. The index D_s is equal to zero, when the two vectors are identical. It is upper-bounded by one if clipping below zero of Q values is enabled.

1.5.3.3 Jointly spectral and spatial quality index

The use of two separate indices may be not sufficient to assess the fused images. In fact, D_λ and D_s respectively measure changes in spectral behavior occurring between the resampled original and the fused images, and discrepancies in spatial details originated by fusion.

A single index, referred to as Q_{NR}, combines the two indices and is the product of the complements of the spatial and spectral distortion indices. The two exponents p and q jointly determine the nonlinearity of response in the interval [0, 1] to achieve a better discrimination of the fusion results compared. The Q_{NR} is defined as

$$Q_{NR} = (1 \quad D_\lambda)^\alpha \cdot (1 \quad D_s)^\beta, \tag{1.82}$$

where α and β are the weights for spectral and spatial distortion, respectively. Thus, the highest value of Q_{NR} is 1 and is obtained when the spectral and spatial distortions are both 0. The main advantage of this index is that, in spite of the lack of a reference dataset, the global quality of a fusion product can be assessed at the full scale of a panchromatic image.

References

1. Fratter, C., M. Moulin, H. Ruiz, P. Charvet, and D. Zobler, "The SPOT5 mission," *52nd International Astronautical Congress*, Toulouse, France (Oct 2001).
2. Pearlman, J., "Overview of the Hyperion Imaging Spectrometer for the NASA EO-1 Mission," *Proc. IGARSS 2001* **7**, 3036–3038 (2001).

3. Lockwood, R. B. et al., "Advanced Responsive Tactically Effective Military Imaging Spectrometer (ARTEMIS) Design," *Proc.IGARSS 2002* (2002).
4. Aumann, H. H. and L. Strow, "AIRS, the first hyper-spectral infrared sounder for operational weather forecasting," in *Proc. IEEE Aerospace Conf. 2001* **4**, 1683–1692 (2001).
5. Lier, P., G. Moury, C. Latry, and F. Cabot, "Selection of the SPOT-5 Image Compression algorithm," *Proc. SPIE* **3439**, 541–551 (1998) [doi: 10.1117/12.325660].
6. Gortl, P. and J. P. Huot, "Overview of the Envisat MERIS and AATSR data quality, calibration and validation program," *Proc. IGARSS 2003* **3**, 1588–1590 (2003).
7. Qian, S.-E., "Evaluation of onboard data compression vs band-averaging of MERIS hyperspectral data on information volume," unpublished internal technical report, Canadian Space Agency (2002).
8. Qian, S.-E., "More efficient satellite data transmission," SPIE Newsroom, article-2940, pp.1–2 (2010) (doi: 10.1117/2.1201005.002940).
9. Wang, Z. and A. C. Bovik, "A universal image quality index," *IEEE Signal Process. Lett.* **9**(3), 81–84 (2002).
10. Wang, Z., A. C. Bovik, and L. Lu, "Why is image quality assessment so difficult?" *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.* **4**, 3313–3316 (2002).
11. Wang, Z., A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Trans. Image Process.* **13**(4), 600–612 (2004).
12. Sheikh, H. R., M. F. Sabir, and A. C. Bovik, "A statistical evaluation of recent full reference image quality assessment algorithms," *IEEE Trans. Image Process.* **15**(11), 3440–3451 (2006).
13. Sheikh, R.H. and A.C. Bovik, "Image Information and Visual Quality," *IEEE Trans. Image Process.* **15**(2), 430–444 (2006).
14. Qian, S.-E., "Hyperspectral data compression using a fast vector quantization algorithm," *IEEE Trans. Geosci. Remote Sens.* **42**(8), 1791–1798 (2004).
15. Aiazzi, B.et al., "Spectral distortion evaluation in lossy compression of hyperspectral imagery," *Proc. IGARSS 2003* **3**, 1817–1819 (2003).
16. Wald, L., T. Ranchin, and M. Mangolini, "Fusion of satellite images of different spatial resolutions: Assessing the quality of resulting images," *Photogramm. Eng. Remote Sens.* **63**(6), 691–699 (1997).
17. Alparone, L., S. Baronti, A. Garzelli, and F. Nencini, "A global quality measurement of pan-sharpened multispectral imagery," *IEEE Geosci. Remote Sens. Lett.* **1**(4), 313–317 (2004).
18. Kantor, I. L. and A. S. Solodnikov, *Hypercomplex Numbers: An Elementary Introduction to Algebras*, Springer-Verlag, Berlin (1989).

19. Moxey, C. E., S. J. Sangwine, and T. A. Ell, "Hypercomplex correlation techniques for vector images," *IEEE Trans. Signal Process.* **51**, 1941–1953 (2003).
20. Sangwine, S. J. and T. A. Ell, "Color image filters based on hypercomplex convolution," *Proc. Inst. Elect. Eng., Vis. Image Signal Process.* **147**(2), 89–93 (2000).
21. Garzelli, A. and F. Nencini, "Hypercomplex quality assessment of multi/hyperspectral images," *IEEE Geosci. Remote Sens. Lett.* **6**(4), 662–665 (2009).
22. Ebbinghaus, H. B., *Numbers*, Springer-Verlag, Berlin (1991).
23. Wang, Z. et al., "Image quality assessment: from error visibility to structural similarity," *IEEE Trans. Image Process.* **13**(4), 600–612 (2004).
24. Mannos, J. L. and D. J. Sakrison, "The effects of a visual fidelity criterion on the encoding of images," *IEEE Trans. Inform. Theory* **4**, 525–536 (1974).
25. Winkler, S., "Issues in vision modeling for perceptual video quality assessment," *Signal Process.* **78**, 231–252 (1999).
26. Pappas, T. N., R. J. Safranek, and J. Chen, "Perceptual criteria for image quality evaluation," Ch. 8.2 in *Handbook of Image and Video Processing*, 2nd Ed., A. Bovik, Ed., Academic Press, San Diego, CA (2005).
27. Wang, Z., H. R. Sheikh, and A. C. Bovik, "Objective video quality assessment," Ch. 41 in *The Handbook of Video Databases: Design and Applications*, B. Furht and O. Marques, Eds., CRC Press, Boca Raton, FL (2003).
28. Sheikh, R.H. and A.C. Bovik, "Image Information and Visual Quality," *IEEE Trans. Image Process.* **15**(2), 430–444 (2006).
29. Qian, S.-E. and G. Chen, "Four Reduced-Reference Metrics for Measuring Hyperspectral Images after Spatial Resolution Enhancement," in *International Archives of the Photogrammetry, Remote Sensing (IAPRS)*, W. Wagner and B. Székely, Eds., 204–208 (2010).
30. Green, R. O. et al., "Imaging spectroscopy and the Airborne Visible/Infrared Imaging Spectrometer (AVIRIS)," *Remote Sens. Environ.* **65**(3), 227–248 (1998).
31. Neville, R.A., N. Rowlands, R. Marois, and I. Powell, "SFSI: Canada's First Airborne SWIR Imaging Spectrometer," *Can. J. Remote Sens.* **21**, 328–336 (1995).
32. Irani, M. and S. Peleg, "Improving resolution by image registration," *Comp. Vision, Graphics, and Image Process.* **53**, 231–239 (1991).
33. Irani, M. and S. Peleg, "Motion analysis for image enhancement: resolution, occlusion and transparency," *J. Visual Comm. Image Representation* **4**, 324–335 (1993).
34. Wang, Z. and E. P. Simoncelli, "Reduced-reference image quality assessment using a wavelet-domain natural image statistic model," *Proc. SPIE* **5666**, 149–159 (2005) [doi: 10.1117/12.597306].

35. Cover, M. T. and J. A. Thomas, *Elements of Information Theory*, Wiley-Interscience, New York (1991).
36. Ihara, S., *Information Theory for Continuous Systems*, World Scientific, New York (1993).
37. Brillouin, L., *Science & Information Theory*, Dover Publications (2004).
38. Shannon, C. E., "A Mathematical Theory of Communication," *Bell System Technical Journal* **27**(3), 379–423 (1948).
39. You, Y., *Audio Coding: Theories and Applications*, Springer, Berlin (2010).
40. Sayood, K., *Introduction to Data Compression*, 2nd Ed., Morgan Kaufmann Publishers, San Francisco, CA (2000).
41. Jayant, N. S. and P. Noll, *Digital Coding of Waveforms – Principles and Applications to Speech and Video*, Prentice Hall, New York (1984).
42. Wu, H. R. and M. Yuen, "A generalized block-edge impairment metric for video coding," *IEEE Signal Process. Lett.* **4**(11), 317–320 (1997).
43. Liu, V.-M. et al., "Objective image quality measure for block-based DCT coding," *IEEE Trans. Consumer Electr.* **3**(3), 511–516 (1997).
44. Kayargadde, V. and J.-B. Martens, "Perceptual characterization of images degraded by blur and noise: model," *J OSA* **13**(6), 1178–1188 (1996).
45. Marziliano, P. et al., "Perceptual blur and ringing metrics: Application to JPEG2000," *Signal Process.: Image Comm.***19**(2), 163–172 (2004).
46. Tan, K. T. and M. Ghanbari, "Frequency domain measurement of blockiness in MPEG-2 coded video," *Proc. IEEE Int. Conf. Image Proc.* **3**, 977–980 (2000).
47. Wang, Z. et al., "Blind measurement of blocking artifacts in images," *Proc. IEEE Int. Conf. Image Proc.* **3**, 981–984 (2000).
48. Wang, Z., H. R. Sheikh, and A. C. Bovik, "No-Reference Perceptual Quality Assessment of JPEG Compressed Image," *Proc. IEEE ICIP-2002* **111**, 477–480 (2002).
49. Alparone, L. et al., "Multispectral and panchromatic data fusion assessment without reference," *Photogrammetric Eng. & Remote Sen.* **74**(2), 193–200 (2008).
50. Bibring, J. P. et al., "OMEGA: Observatoire Pour la Minéralogie, l'Eau, les Glaces et l'Activité," in *Mars Express: the Scientific Payload*, edited by A. Wilson, 37–49. ESA Publ. Division, Noordwijk (2004).
51. Murchie, S. R. et al., "Compact Reconnaissance Imaging Spectrometer for Mars (CRISM) on Mars Reconnaissance Orbiter (MRO)," *J Geophysical Res.: Planets* **112**(E5) (2007).
52. "eoPortal: Sharing Earth Observation Resources," http://directory.eoportal.org.

Chapter 2
Lossless Satellite Data Compression

2.1 Introduction

Satellite sensors produce enormous data volumes, especially hyperspectral sensors that acquire over hundreds of images of narrow-wavelength bands and produce an image cube. Thus, a lot of effort has been made to study more-efficient ways to compress satellite images or image cubes. Compression techniques can be classified into three types:

1. Lossless compression,
2. Near-lossless compression, and
3. Lossy compression.

Lossless compression techniques are reversible techniques that compress an image without loss of information. The reconstructed image is identical to the original image. Because there is no loss of information, this kind of compression technique is used for applications that cannot tolerate any difference between the original and reconstructed data. However, a lossless compression technique cannot achieve a high compression ratio, depending on the redundancy of the images. The larger the redundancy is, the higher the compression ratio that can be achieved. For optical satellite images, the lossless compression ratio is normally less than 3:1. For an image with a very smooth scene or extremely low spatial or spectral information in the data, a higher compression ratio may be achieved.

A lossy compression technique compresses an image with errors. The reconstructed image is not exactly the same as the original image. It does not bind the difference between reconstructed pixels and the original pixels. Instead, the reconstructed image is required to be similar to the original image on a mean-squared error sense. High compression ratios can be achieved. The higher the compression ratio is, the larger the compression error.

A near-lossless compression technique lies between the lossless and lossy compression techniques. The error introduced by a near-lossless compression

technique is bound by a predefined threshold, such as RMSE, the accuracy of an application product. A near-lossless compression means that it is theoretically still a lossy compression due to its irreversibility; however, the loss of information caused by the compression is designed to have negligible or minor impact on the derivation of the ultimate data products or applications. Satellite data users often do not like lossy data compression and may be willing to accept near-lossless compression by trading off the gain and cost of the compression.

For satellite data, lossy compression is normally not recommended because it will reduce the value of acquired data for their purpose. For this reason, lossy data compression is not a subject of this book. Instead, this book describes both lossless and near-lossless data compression techniques in this and following chapters.

Lossless compression techniques can be generally classified as two categories: prediction-based and transform-based. The former is based on the predictive coding paradigm, whereby a current pixel is predicted from the previous pixels, and the prediction error is then entropy coded.[1,2] Lossless compression techniques that use a lookup-table or vector-quantization method are also categorized as prediction-based methods because both the lookup-table and vector-quantization methods are used to generate prediction of the data. A vector-quantization-based lossless technique is an asymmetric compression process that is much more computationally intensive than the decompression. For prediction-based lossless compression, band-reordering techniques may also be applied before the prediction to improve the compression ratio. Transform-based methods have been more successful in lossy compression than lossless compression. This is because the transform used for lossless compression, such as the discrete cosine transform (DCT) or wavelet transform (WT), must be reversible or integer. This requirement may compromise the ability of the transform to decorrelate the data to be compressed.

Figure 2.1 shows the block diagram of a prediction-based lossless compression process. Assume that the input is a hyperspectral datacube $\mathbf{I}(x, y, \lambda)$, where x, y, and λ denote the spatial coordinates of a ground sample in the scene of the datacube and the spectral band number, respectively. If the input is a 2D image, then $\lambda = 1$. The first step is the band reordering. This step

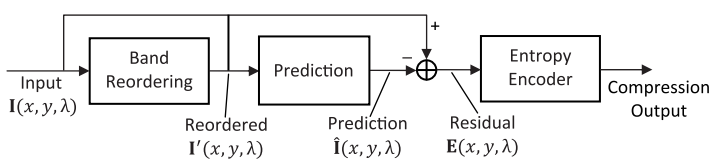


Figure 2.1 Block diagram of a prediction-based lossless compression process.

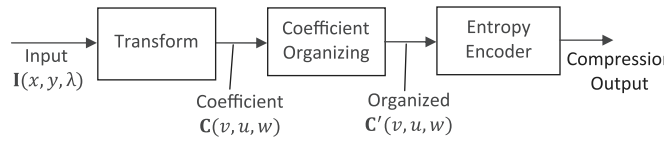


Figure 2.2 Block diagram of a transform-based lossless compression process.

may or may not be taken depending on the specific applications. Obviously, for a 2D image, this step is unnecessary. The datacube after band reordering $\mathbf{I}'(x, y, \lambda)$ is input to the prediction step to generate prediction values for each pixel of the original datacube. This step can be implemented by using conventional prediction methods, such as nearest neighbor prediction in either the spatial or spectral domain or in both the spatial and spectral domains, or using the lookup-table or vector-quantization method. The difference $\mathbf{E}(x, y, \lambda)$ (sometimes also called the residual) between the original $\mathbf{I}(x, y, \lambda)$ or $\mathbf{I}'(x, y, \lambda)$ and predictive value $\hat{\mathbf{I}}(x, y, \lambda)$ is obtained and input to the entropy encoder step for generating the compressed data or bitstreams.

Figure 2.2 presents the block diagram of a transform-based lossless compression process. No band reordering is required in this case, even if the input is a hyperspectral datacube. The transform function can be a single reversible DCT, an integer WT, or a principal component analysis (PCA) transform, or a combination of two transforms, one for spectral and another for spatial decorrelation. The transformed coefficients $\mathbf{C}(v, u, w)$ are decorrelated to facilitate removal of redundancy. The coefficients are organized in a way suitable for encoding before being sent to the entropy encoder, which generates the compressed data.

2.2 Review of Lossless Satellite Data Compression

2.2.1 Prediction-based methods

Many prediction-based lossless techniques have been reported. In hyperspectral datacubes, the interband correlation is much stronger than the intraband correlation, which is why most predictive, lossless, hyperspectral data compression techniques use prediction between bands for maximal compression performance. Unlike multispectral imagers, hyperspectral imagers generate narrow spectral bands over a continuous spectral range. This results in high correlation between the spectral band images.

A fuzzy prediction method was introduced by Aiazzi et al.[3] The predictor is switched within a predefined set based on a fuzzy logic rule. The authors improved the prediction by using analysis of edges.[4] They further developed a classified prediction for both lossless and near-lossless compression.[5] The causal neighborhoods of each pixel are clustered using fuzzy c-means

clustering. For each of the clusters, an optimal linear predictor is computed from the values and the membership degrees of those that exceed a threshold. The final estimate is computed as a weighted sum of the predictors, where the weights are the membership degrees. The spectral fuzzy matching pursuits (SFMP) method exploits a purely spectral prediction. In their paper, a method called spectral relaxation-labeled prediction was also proposed. The method partitions image bands into blocks, and a predictor is selected for prediction out of a set.

A concept of clustered-differential pulse code modulation was introduced by Mielikainen and Toivanen.[6] The spectra of a datacube are clustered into spatially homogeneous classes. A separate linear predictor minimizing the expected value of the squared prediction error is used inside each cluster. An optimal predictor is computed for each cluster, used to remove the spectral correlation and to generate the prediction error, which is coded using a range coder. They further proposed a linear prediction model.[7] The predictor is optimized for each pixel and each band in a causal neighborhood of the current pixel.

A prediction method based on context-adaptive lossless image coding is reported.[8] This method switches between intra- and interband prediction modes based on the strength of the correlation between the consecutive bands. Another multiband prediction method performs prediction using two pixels in the previous bands in the same spatial position as the current pixel.[9] The prediction coefficients are computed using an offline procedure on training data. An adaptive least-squares-optimized prediction technique called spectrum-oriented least squares (SLSQ) was presented by Rizzo et al.[10] The same prediction technique as that of Mielikainen and Toivanen[7] is used, but the entropy coder is more advanced. A SLSQ-HEU uses an offline heuristic to select between the intra- and interband compression modes. An optimal method for inter-/intracoding mode selection called SLSQ-OPT was also presented.

A block-based interband compressor prediction algorithm is proposed.[11] Each band image of an input datacube is divided into square blocks. The blocks are then predicted based on the corresponding block in the previous band. A correlation-based, conditional-average prediction method is reported.[12] This method estimates the sample mean corresponding to the current pixel in contexts that match the current pixel context. A selection is then performed based on a correlation coefficient for contexts to decide the use of the prediction or a lossless JPEG. A nonlinear prediction method for hyperspectral images is also reported.[13] The method predicts the pixel in the current band image based on the information in the causal context in the current band image and pixels co-located in the reference band image. This method is also extended into an edge-based technique, called the edge-based prediction for hyperspectral images, which classifies the pixels into edge and

non-edge pixels. Each pixel is then predicted using information from pixels in the same pixel class within the context.

The lookup-table method is one of the prediction methods.[14,15] It makes a prediction of the current pixel by searching all of the causal pixels in the current and previous band images. It is based on the idea of a nearest-neighbor search. The search is performed in reverse raster-scan order. The pixel in the previous band at the same position as the pixel in the current band is used as a predictor. The lookup tables are used to speed up the search.

Vector quantization (VQ) methods are also used to predict pixel values in lossless compression.[16 23] Ryan and Arnold identified various vector formation techniques and investigated suitable quantization parameters for lossless data compression using a VQ method. They proposed a mean-normalized vector quantization technique that produced compression performances approaching the theoretical minimum compressed image entropy of 5 bits/pixel of the AVIRIS datacubes. Images are compressed from original image entropies of 8.28–10.89 bits/pixel to 4.83–5.90 bits/pixel.[16]

Mielikainen and Toivanen used a VQ method to compress AVIRIS datacubes.[17] A codebook is first trained from the datacube to be compressed using the generalized Lloyd algorithm (GLA)[18] and then is used to compress the datacube. A reconstructed datacube is generated by decompression right after compression and is used as the predictive datacube. A residual datacube is obtained by subtracting the reconstructed datacube from the original image. The difference images between the two consecutive band images of the residual datacube are calculated and formed. Finally, each difference image is entropy-encoded. The index and codebook values are also separately entropy-encoded. This method achieved a lossless compression ratio of approximately 3:1 for the test AVIRIS datacubes.

A partitioned VQ compression method has been proposed.[19] A vector quantizer is designed that uses multiple codebooks. The spectral vectors of the hyperspectral datacube are first partitioned into two or more subvectors of different lengths, and then each subvector is individually encoded with an appropriate codebook. The indices of the subvectors are encoded using conditional entropy code.

Two VQ-based lossless compression methods for multispectral images were reported.[20,21] A VQ compression method for hyperspectral imagery using entropy-constrained, predictive trellis-coded quantization was proposed.[22] A trellis-coded quantization is another VQ method. A compression algorithm based on locally optimal partitioned vector quantization was presented.[23] It compresses hyperspectral images by applying partitioned VQ to the spectral signatures and then encoding error information with a threshold that can be varied from high-quality lossy to near-lossless to lossless. This algorithm requires more-complex encoding.

To overcome the complexity problem, a low-complexity algorithm referred to as spectrum-oriented least squares is presented.[24] It employs linear prediction targeted at spectral correlation followed by entropy coding of the prediction error.

Band reordering has also been used to obtain improved lossless compression performance.[25 27] Specifically, in band reordering, the spectral band images of a datacube are reordered in such a way as to maximize the correlation of adjacent band images, thus optimizing the performance of the subsequent compression stage. The problem of optimal band ordering for hyperspectral image compression has been solved.[25] Optimal band reordering is achieved by computing a minimum spanning tree for a directed graph containing the sizes of the encoded residual bands. A correlation-based heuristic for estimating the optimal order was proposed by Toivanen et al.[26] Another prediction method based on reordering was introduced by Zhang and Liu.[27]

2.2.2 Transform-based methods

A transform-based lossless compressed technique for multispectral images has been reported.[28] It applied PCA and the integer wavelet transform to decorrelate both spectral and spatial redundancies. This technique has been slightly modified to get better compression results for both multispectral and hyperspectral images.[29] An integer version of the PCA is applied to produce a set of base vectors that minimize the approximation error in the MSE sense. Only a small number of spectra from the image are selected for the calculation of the eigenvectors in order to reduce computational complexity. The residual image is calculated from the available approximation image and then compressed using an integer wavelet transform, which is a reversible integer-to-integer transform. It has been used in lossless color and grayscale image coding and has shown good performance.[30] The integer wavelet transform is based on the lifting scheme: different filters are derived by combining the prediction step with the update step. The integer wavelet transform is 1D in nature. In the 2D case, the 1D transform is applied to the rows and columns of the image. In the 3D case, the 1D transform is applied to the spatial and spectral domains separately. After the PCA transform, the eigenvectors are saved, and the differences between their coefficients are entropy coded. The bands of the transformed residual image are coded separately using different entropy coders, one band at time.

A transform-based, lossy-to-lossless, hyperspectral image-compression technique has been proposed by Wang et al.[31] Instead of applying a discrete wavelet transform (DWT) in the spatial domain, this technique uses a reversible discrete cosine transform (RDCT) for spatial decorrelation and a reversible-integer, low-complexity Karhunen–Loève transform

(KLT) for spectral decorrelation. The DCT has its own special advantages, such as low memory usage, flexibility at the block-by-block level, parallel processing, etc. Tran et al.[32] have designed pre- and postfilters to improve the performance of the DCT and called the combination of them a time-domain lapped transform (TDLT). The TDLT performs better than the DWT in energy compatibility and lossy compression. However, it does not perform well in lossless compression techniques where the reversible transform is required. A reversible-integer TDLT is developed and used to replace the integer WT to overcome the drawback of the TDLT. Thanks to both the DCT and KLT being reversible, the proposed method can compress hyperspectral images progressively from lossy to lossless in a single, embedded codestream file.

Baizert et al.[33] proposed a transform-based lossless compression algorithm for hyperspectral data using a VQ technique and the DCT. They demonstrated that a combination of mean-normalized vector quantization (M-NVQ) in the spatial domain and the DCT techniques in the spectral domain is the preferred compression system. The compression ratios produced by such a spectral DCT together with a spatial M-NVQ coder are 1.5–2.5 times better than the compression ratios obtained by the M-NVQ technique alone. Their work shows that, for low distortion levels, further improvements in compression ratios can be obtained by replacing the spatial M-NVQ technique with the 2D DCT.

A 3D compression algorithm based on integer wavelet transforms (IWTs) and zerotree coding is presented.[34] The algorithm for embedded zerotrees of wavelet transforms (EZWs) is extended to three dimensions, and context-based adaptive arithmetic coding is used to improve its performance. The resultant algorithm is referred to as 3D context-based embedded zerotrees of wavelet transforms (3D CB-EZWs). It efficiently encodes 3D image data by exploiting dependencies in all dimensions while enabling lossy and lossless decompression from the same bitstream. Compared with the best available 2D lossless compression techniques at that time,[35,36] the 3D CB-EZW algorithm produced average decreases of 22%, 25%, and 20% in compressed file sizes for computed tomography, magnetic resonance, and AVIRIS hyperspectral images, respectively. The progressive performance of the algorithm is also compared with other lossy progressive-coding algorithms.

A 2D-transform-based lossy-to-lossless algorithm using the IWT was proposed by Grangetto et al.[37] They first studied methods for the selection of the best factorization of wavelet filters within the lifting scheme framework. Experimental results were reported for a number of filters, test images, and for both lossless and lossy compression, showing that the obtained IWT implementation achieved compression performance very close to the real-valued DWT. They evaluated the effects of finite precision representation of

the lifting coefficients and the partial results. Their analysis revealed that a very small number of bits can be devoted to the mantissa with acceptable performance degradation.

2.3 Entropy Encoders

As described in Section 2.2, it can be seen that the efforts for lossless data compression are all focused on the development of effective predictors or selection of powerful transform methods. All of the proposed lossless compression techniques use three popular entropy encoders: arithmetic coding,[38] Golomb coding,[39] and Golomb power-of-two coding (GPO2, also referred to as Rice coding).[40] An entropy encoder is at the final stage in the lossless compression chain to produce the compressed bitstream. In essence, such entropy coders assign shorter bitstream codewords to more-frequently occurring symbols in order to maximize the compactness of the bitstream representation.

In information theory, an entropy encoder is a lossless data compression element that is independent of the specific characteristics of the medium. It creates and assigns a unique prefix-free code to each unique symbol that occurs in the input and then compresses the symbols by replacing each fixed-length input symbol with the corresponding variable-length, prefix-free output codeword. The length of each codeword is approximately proportional to the negative logarithm of the probability. Therefore, the most-common symbols use the shortest codes. This section briefly describes these entropy encoders.

2.3.1 Adaptive arithmetic coding

Arithmetic coding (AC) is a widely used entropy encoder.[38] It is a form of variable-length entropy encoding. A string of data is normally represented using a fixed number of bits per character. When a string is converted to arithmetic encoding, frequently-used characters will be stored with fewer bits, and not-so-frequently-occurring characters will be stored with more bits, resulting in fewer bits used in total. Arithmetic coding differs from other forms of entropy encoding, such as Huffman coding, in that rather than separating the input into component symbols and replacing each with a code, AC encodes the entire message in a single number, a fraction n, where $0.0 \leq n < 1.0$.

The adaptive arithmetic coding (AAC) involves changing probability (or frequency) tables while processing the data. The decoded data matches the original data as long as the probability table on the decoding side is replaced in the same way and in the same step as on the encoding side. The synchronization is usually based on a combination of symbols that occur during the encoding and decoding process. AAC significantly improves

coding efficiency compared to static methods; it may be as effective as 2–3 times better in the result.

AAC encodes a stream of symbols into a bitstream with a length very close to its theoretical minimum limit. Assume source X produces symbol i with probability p_i. The entropy of source X is defined to be

$$H(X) = \sum_i p_i \log_2 p_i, \tag{2.1}$$

where $H(X)$ has units of bits per symbol (bps). One of the fundamental tenets of information theory is that the average bitrate in bps of the most-efficient lossless (i.e., invertible) compression of source X cannot be less than $H(X)$. In practice, AAC often produces a bitrate quite close to $H(X)$ by estimating the probabilities of the source symbols with frequencies of occurrence as it encodes the symbol stream. Essentially, the better able AAC can estimate p_i, the closer it will come to $H(X)$—the lower bound on coding efficiency. The efficiency of AAC can often be improved by conditioning the encoder with known context information and maintaining separate symbol-probability estimates for each context. That is, limiting attention of AAC to a specific context usually reduces the variety of symbols, thus permitting better estimation of the probabilities within that context and producing greater coding efficiency. From a mathematical standpoint, the conditional entropy of source X with known information Y is $H(X|Y)$. Because it is well known from information theory that

$$H(X|Y) \leq H(X), \tag{2.2}$$

conditioning AAC with Y as the context will (usually) produce a bitstream with a smaller bitrate.

2.3.2 Golomb coding

Golomb coding uses a tunable parameter M to divide an input value x into two parts: q, the quotient of a division by M; and r, the remainder. The quotient is sent in unary coding, followed by the remainder in truncated binary encoding.[39] When $M = 1$, Golomb coding is equivalent to unary coding. In the case of a geometrically distributed random variable x, the appropriately selected Golomb code minimizes the expected codeword length over all possible lossless binary codes for x.

The two parts are given by the following expression, where x is a random variable being encoded:

$$q = \frac{x \quad 1}{M}, \tag{2.3}$$

$$r = x \quad qM \quad 1. \tag{2.4}$$

The final result looks like

$$(q \text{ pieces of } 1)r. \tag{2.5}$$

It is a code string of q pieces of '1' followed by $q = \log_2 M$ bits with remainder r. The parameter M is a function of the corresponding Bernoulli process, which is parameterized by $p = P(X = 0)$, the probability of success in a given Bernoulli trial. M is either the median of the distribution or the median ± 1. It can be determined by the following inequalities:

$$(1 \quad p)^M + (1 \quad p)^{M+1} \leq 1 < (1 \quad p)^{M \ 1} + (1 \quad p)^M. \tag{2.6}$$

Golomb states that, for large M, there is very little penalty for picking

$$M = round\left(\frac{1}{\log_2(1 \quad p)}\right). \tag{2.7}$$

The Golomb code for this distribution is equivalent to the Huffman code for the same probabilities, if it were possible to compute the Huffman code.

2.3.3 Exponential-Golomb coding

An exponential-Golomb (exp-Golomb) code of order k is a type of universal code, parameterized by a non-negative integer k. To encode a non-negative integer in an order-k exp-Golomb code, the following method can be used:

1. Take the number in binary, except for the last k digits, and add one to it (arithmetically). Write this down.
2. Count the bits written, subtract one, and write that number of starting zero bits preceding the previous bit string.
3. Write the last k bits in binary. For $k = 0$, the code begins:

0 => 1 => 1
1 => 10 => 010
2 => 11 => 011
3 => 100 => 00100
4 => 101 => 00101
5 => 110 => 00110
6 => 111 => 00111
7 => 1000 => 0001000
8 => 1001 => 0001001
...

2.3.4 Golomb power-of-two coding

Golomb power-of-two coding[40] uses a subset of the family of Golomb codes to produce a simpler but possibly suboptimal prefix code. A Golomb code has

when $k = 2$,
$x = 5$ (binary code: 101) ==> 0 1 01
$x = 10$ (binary code: 1010) ==> 00 1 10
$x = 20$ (binary code: 10100) ==> 00000 1 00

when $k = 3$
$x = 5$ (binary code: 101) ==> 1 101
$x = 10$ (binary code: 1010) ==> 0 1 010
$x = 20$ (binary code: 10100) ==> 00 1 100

Figure 2.3 An example of GPO2 codes.

a tunable parameter M that can be any positive value. GPO2 codes are those in which the tunable parameter is a power of two. This makes GPO2 codes convenient for use on a computer because multiplication or division by two can be implemented by shifting the bits left or right in the register of the computer.

In a GPO2 code, $M = 2^k$, where k is a non-negative integer. The codeword for a random integer variable x consists of $\lfloor x/2^k \rfloor$ zeroes followed by a '1' concatenated with the k least-significant bits of the binary representation of x. This codeword is called a GPO2 code of parameter k. An example of the GPO2 codes is shown in Fig. 2.3 when $k = 2, 3$ for three variables x.

In GPO2 coding, the key is how to select the parameter k that produces the minimum bitrate for input source variables or an image. Rice encoded a block of variables by using the best code option for the block from among several candidate codes that consisted mostly of different GPO2 codes. The Rice method does not specify how to find the best code option. The minimum bitrate parameter k is selected by exhaustively trying every optional value to pick the best one for the block. A fixed number of additional bits are used preceding the encoded block to indicate which optional code was selected. Information from previously coded blocks is not utilized.

An adaptive parameter-k selection method has been proposed by Kiely.[41] The mean of the samples of a block to be encoded is utilized. The method is based partly on a theoretical derivation of bounds on the optimum value of k as functions of the mean sample value, as shown in Fig. 2.4. These bounds are such that no more than three code choices can be optimum for a given mean sample value. For a given mean value, one of the three candidate codes is selected in a procedure that involves only integer arithmetic and table lookups. It has been shown that the value of k selected in this relatively simple procedure is always within one of the optimum k values for the source, and that the cost added by the suboptimality of the selection is never more than a half-bit per sample.

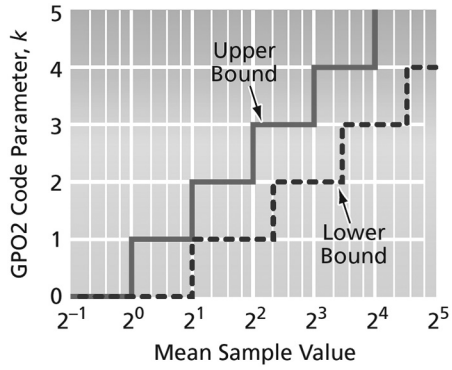


Figure 2.4 Selection of optimum code parameter *k* in a GPO2 encoder based on the mean of samples of a block (Source: NASA).

2.4 Predictors for Hyperspectral Datacubes

This section describes real-time lossless data compression of hyperspectral datacubes using prediction followed by entropy encoding. The main effort is to study predictors that can yield the best reduction of entropy of the datacubes and can be easily implemented in real time. The GPO2 encoder used by the Consultative Committee for Space Data Systems (CCSDS) is selected as the entropy encoder. Four predictor schemes have been selected for study. Three typical hyperspectral datacubes acquired by AVIRIS and three datacubes acquired by CASI were used as test data. A lossless compression system with different predictors has been simulated and tested with the test data.[42]

Lossless data compression (LDC)[43] is an international standard developed by the CCSDS for onboard satellite data compression. It is a two-step compression algorithm distinguished by the use of a predictor stage followed by a GPO2 encoder (Rice encoder).[40] This algorithm has been implemented in a chipset called USES (universal source encoder for space), which has been used in a number of space missions. The design of the predictor stage has been specified in such way that it can be customized. The USES chip is equipped with a 1D nearest-neighboring predictor (NNP). Selection of an external predictor is not adaptive and must be chosen prior to beginning the compression process. The residual between the original data and its predictor is then entropy-encoded using a GPO2 encoder. This standard uses a bank of several optional encoders, each optimized for a different entropy level; a typical implementation uses 16 encoders. The encoders operate in parallel, and the one that yields the fewest bits is selected and output. The output of the GPO2 encoder has been shown to be equivalent to Huffman encoding if the input distribution of the residual data is Laplacian. However, the throughput is much faster than Huffman encoding, as the codebook is not required, and

Table 2.1 1D nearest-neighboring predictors.

ID	Predictor	Description
0	$\hat{x}_{r,c,\lambda} \quad 0$	No prediction
1	$\hat{x}_{r,c,\lambda} \quad x_{r,c-1,\lambda}$	Cross track direction
2	$\hat{x}_{r,c,\lambda} \quad x_{r-1,c,\lambda}$	Along track direction
3	$\hat{x}_{r,c,\lambda} \quad x_{r,c,\lambda-1}$	Spectral direction

the input distribution is assumed *a priori*. Details of the CCSDS-recommended algorithm can be found in the green book of the standard.[44]

2.4.1 1D nearest-neighboring predictor

Assume that $x_{r,c,\lambda}$ is the value of a pixel of a hyperspectral datacube at row r, column c, and spectral band λ, and its predictive value is $\hat{x}_{r,c,\lambda}$. Three 1D NNPs are studied; they are listed in Table 2.1. This study attempts to examine the decorrelation capability offered by the built-in 1D NNP predictor of the USES chip.

2.4.2 2D/3D predictors

Four 2D/3D predictors are selected (see Table 2.2). This study attempts to use multidimensional predictors to examine the decorrelation capability.

2.4.3 Predictors within a focal plane image

Most hyperspectral sensors use area arrays on the focal plane of the instrument. Each time the array is exposed, a cross-track line (one spatial dimension) is acquired. Each ground sample within the cross-track line has an

Table 2.2 2D/3D predictors.

ID	Predictor
4	$\hat{x}_{r,c,\lambda} \quad \frac{1}{2}(x_{r-1,c,\lambda} + x_{r,c-1,\lambda})$
	2D NNP in two spatial directions
80	$\hat{x}_{r,c,\lambda} \quad \frac{1}{6}[(3x_{r-1,c,\lambda} \quad x_{r-2,c,\lambda}) + (3x_{r,c-1,\lambda} \quad x_{r,c-2,\lambda}) + (3x_{r,c,\lambda-1} \quad x_{r,c,\lambda-2})]$
	3D, 2 point NNP in three directions
84	$\hat{x}_{r,c,\lambda} \quad \frac{1}{2}(x_{r-1,c,\lambda} + x_{r,c-1,\lambda} \quad x_{r-1,c,\lambda-1} \quad x_{r,c-1,\lambda-1}) + x_{r,c,\lambda-1}$
	5 point 3D predictor
87	$\hat{x}_{r,c,\lambda} \quad \frac{1}{9}\left[\begin{matrix}4(x_{r-1,c,\lambda} \quad x_{r-1,c,\lambda-1}) + 4(x_{r,c-1,\lambda} \quad x_{r,c-1,\lambda-1}) \\ +(x_{r-1,c-1,\lambda} \quad x_{r-1,c-1,\lambda-1})\end{matrix}\right] + x_{r,c,\lambda-1}$
	7 point 3D predictor

Table 2.3 Predictors within the focal plane image.

ID		Predictor	
82	2 point	$\hat{x}_{r,c,\lambda}$	$2x_{r,c,\lambda-1} \quad x_{r,c,\lambda-2}$ if slope increases or decreases,
		$\hat{x}_{r,c,\lambda}$	$\frac{1}{2}(x_{r,c,\lambda-1} + x_{r,c,\lambda-2})$ otherwise
100		$\hat{x}_{r,c,\lambda}$	$Ax_{r,c-1,\lambda} + Bx_{r,c,\lambda-1}$
101		$\hat{x}_{r,c,\lambda}$	$Ax_{r,c-1,\lambda} + Bx_{r,c-2,\lambda}$
102		$\hat{x}_{r,c,\lambda}$	$Ax_{r,c,\lambda-1} + Bx_{r,c,\lambda-2}$
103		$\hat{x}_{r,c,\lambda}$	$Ax_{r,c-1,\lambda} + Bx_{r,c-1,\lambda-1}$
104		$\hat{x}_{r,c,\lambda}$	$Ax_{r,c,\lambda-1} + Bx_{r,c-1,\lambda-1}$
105	3 point	$\hat{x}_{r,c,\lambda}$	$Ax_{r,c,\lambda-1} + Bx_{r,c-1,\lambda} + Cx_{r,c-1,\lambda-1}$
106		$\hat{x}_{r,c,\lambda}$	$Ax_{r,c,\lambda-1} + Bx_{r,c,\lambda-2} + Cx_{r,c-1,\lambda}$
107		$\hat{x}_{r,c,\lambda}$	$Ax_{r,c,\lambda-1} + Bx_{r,c-1,\lambda} + Cx_{r,c-2,\lambda}$
108		$\hat{x}_{r,c,\lambda}$	$Ax_{r,c,\lambda-1} + Bx_{r,c,\lambda-2} + Cx_{r,c-1,\lambda-1}$
109		$\hat{x}_{r,c,\lambda}$	$Ax_{r,c-1,\lambda} + Bx_{r,c-2,\lambda} + Cx_{r,c-1,\lambda-1}$
		$\hat{x}_{r,c,\lambda}$	$\min(x_{r,c-1,\lambda},\ x_{r,c,\lambda-1})$ if $x_{r,c-1,\lambda-1} \geq \max(x_{r,c-1,\lambda},\ x_{r,c,\lambda-1})$
6		$\hat{x}_{r,c,\lambda}$	$\max(x_{r,c-1,\lambda},\ x_{r,c,\lambda-1})$ if $x_{r,c-1,\lambda-1} < \min(x_{r,c-1,\lambda},\ x_{r,c,\lambda-1})$
		$\hat{x}_{r,c,\lambda}$	$x_{r,c-1,\lambda} + x_{r,c,\lambda-1} \quad x_{r,c-1,\lambda-1}$ otherwise

associated spectral profile. A frame of the 2D focal plane image is formed, which corresponds to one row of spatial pixels in the cross-track direction where each pixel in the row has an associated spectrum. The input to the data compressor will typically be a serial readout of the focal plane elements. A predictor using previous data in the focal plane image requires the least capacity of a buffer. In this subsection, the predictors are constrained to use data only within a focal plane image. Twelve predictors in the focal plane image have been selected (see Table 2.3).

Predictor ID #82 is a 1D, spectral, 2-point NNP. Predictor IDs #100–104 are 2-point predictors; they use two values (points) in either one or two dimensions of the focal plane image. The sum of the two coefficients A and B must equal 1. To facilitate real-time operation the coefficients are limited to values of $\pm 2^{\pm n}$. Three combinations of the coefficients were selected for the study, listed in Table 2.4 as the sub-IDs (SIDs).

Predictor IDs #105–109 and ID #6 are 3-point predictors; predictor ID #6 is called LOCO-I,[45] and it is used in an international standard of compression for still images. Each predictor has three coefficients, except for #6, the sum of which must equal 1. The same limitation (values of $\pm 2^{\pm n}$) is

Table 2.4 Coefficients of 2-point focal plane predictors.

SID	A	B
0	1/2	1/2
1	2	1
2	1	2

Table 2.5 Coefficients of 3-point focal plane predictors.

SID	A	B	C	SID	A	B	C
0	1	1	1	**8**	1/2	1	1/2
1	1	1	1	**9**	1/2	1	1/2
2	1	1	1	**10**	1/2	1/2	1
3	1/4	1/4	1/2	**11**	1/2	1/2	1
4	1/4	1/2	1/4	**12**	1/2	1/2	2
5	1/2	1/4	1/4	**13**	1/2	2	1/2
6	1	1/2	1/2	**14**	2	1/2	1/2
7	1	1/2	1/2				

adopted for selecting the three coefficients. Fifteen combinations of coefficients were selected for study and are listed in Table 2.5.

2.4.4 Adaptive selection of predictor

This scheme is composed of a predictor bank with a set of N fixed predictors and a predictor selector. The predictors in the bank are chosen from the best ones studied in Sections 2.4.1–2.4.3. In the compression, an input hyperspectral datacube is divided into blocks and routed through each predictor in parallel block by block. After prediction, N parallel blocks of the residuals obtained from the difference between the originals and their predictions by each predictor are sent to the selector and compared based on a predefined criterion. The block of residuals that yields the best performance is selected as output to be fed to the USES chip. An overhead of $\log_2 N$ bits is required to identify the best predictor for that block. Figure 2.5 shows the block diagram of this scheme. Four fixed predictors ($N = 4$) are used in the predictor bank.

This scheme includes two parts: (1) choosing four fixed predictors to form the predictor bank, and (2) studying the criterion for selecting the predictor that yields the best output residuals.

Three predictor selection criteria have been proposed and studied:

1. **Minimum sum**
   For block data $\mathbf{X}(b)$, where b is the block number, and its residual produced by predictor p is $\mathbf{R}_p(b)$, this criterion consists of computing the

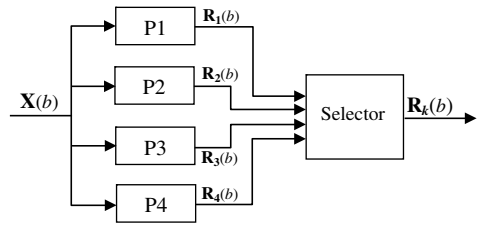


Figure 2.5 Block diagram of the adaptive selection of a predictor.

sum of absolute residuals of the entire block produced by each of N fixed predictors:

$$S_p = \sum_{i\ 1}^{L} \left| R_p^i(b) \right| \quad (p = 1, 2, \ldots, N), \tag{2.8}$$

where L is the total number of the samples in the block. The residual $\mathbf{R}_k(b)$ produced by predictor k is selected as output of the block if $S_k = \min\ (S_1, S_2, \ldots, S_N)$.

2. **Most common**
   For block data $\mathbf{X}(b)$, a signed residual $\Delta_p^i(b)$ in the block, produced by fixed predictor p ($p = 1,2,\ldots,N$), is first mapped into unsigned data $R_p^i(b)$. This unsigned data is then used as an address of a register to locate a corresponding accumulator in the population array whose size is $R_{\max}$, where $R_{\max}$ is the maximum mapped unsigned value. The array needs to be initialized at the beginning of compression. The population of all unsigned residuals $R_p^i(b)\ (i = 1, 2, \ldots L)$ by each fixed predictor is summed as

$$P_p = \sum_{i\ 1}^{L} C[R_p^i(b)] \quad (p = 1, 2, \ldots, N), \tag{2.9}$$

   where operator C[addr] is used to get the content (i.e., population) of the accumulator specified by addr. The residuals $R_k^i(b)\ (i = 1, 2, \ldots, L)$ produced by predictor k are selected as output of the block if $P_k = \max\ (P_1, P_2, \ldots, P_N)$. Before going to the next block, each register at address $R_k^i\ (b)\ (i = 1, 2, \ldots, L)$ is increased by one to accumulate its population.

3. **Smallest maximum value**
   For block data $\boldsymbol{X}(b)$, the maximum absolute value of residuals in the block produced by a fixed predictor p is sought:

$$G_p = \max(|R_p^1(b)|, |R_p^2(b)|, \ldots, |R_p^L(b)|) \quad (p = 1, 2, \ldots, N). \tag{2.10}$$

   The residuals $R_k^i(b)\ (i = 1, 2, \ldots, L)$ obtained by predictor k are selected as the output of the block if $G_k = \min\ (G_1, G_2, \ldots, G_N)$.

2.4.5 Experimental results of the predictors

A total of six datacubes from two imaging spectrometers, AVIRIS and CASI, were used as test datasets. Three of them are 1994 AVIRIS public cubes: Cuprite94, Jasperidge, and Moffet. They have the same size (614 pixels × 512 lines × 224 bands × 16-bit). Three CASI cubes are derived from Boreal Ecosystem–Atmosphere Study flight datasets,[46] called *cube*1 through *cube*3. Their sizes are 128 pixels × 2200 lines × 72 bands × 16-bit, 225 pixels × 2852 lines × 72 bands × 16-bit, and 250 pixels × 900 lines × 72 bands × 16-bit, respectively.

2.4.5.1 Compression results using fixed coefficient predictors

All of the predictors in Sections 2.4.1 and 2.4.2, and in predictor IDs #82 and #6 in Section 2.4.3, are fixed coefficient predictors. Figure 2.6 shows the average compression ratio over the six test datacubes using the fixed coefficient predictors followed by two entropy encoders (bars). The variation ranges of the compression ratios are also shown (lines). In the figure, the first-order entropy $H_0(x \quad \hat{x})$ of the residual cubes ($\Delta = x \quad \hat{x}$) was calculated and used to estimate the compressibility produced by the studied predictors:

$$C_{rHo} = \frac{len}{H_0(x \quad \hat{x})}, \tag{2.11}$$

where *len* is the number of bits at which the original data are packed ($len = 16$ in this study).

Two entropy encoders, GPO2 and base-bit plus overflow-bit coding (BPOC),[47] were used to encode the residual datacubes. BPOC is a simple and effective entropy encoder developed by the author in an earlier project; it is shown here for the purpose of comparison with the CCSDS algorithm. BPOC is also a variable-length encoder composed of two parts: the base bits and the overflow bits. The length of base bits L_b is the only parameter of this coding method—it is determined by the inverse mean of the data to be encoded. The length of overflow bits L_o is variable—it depends on the sample value to be encoded. The length of overflow bits is $L_o = 0$ if the value of a sample to be

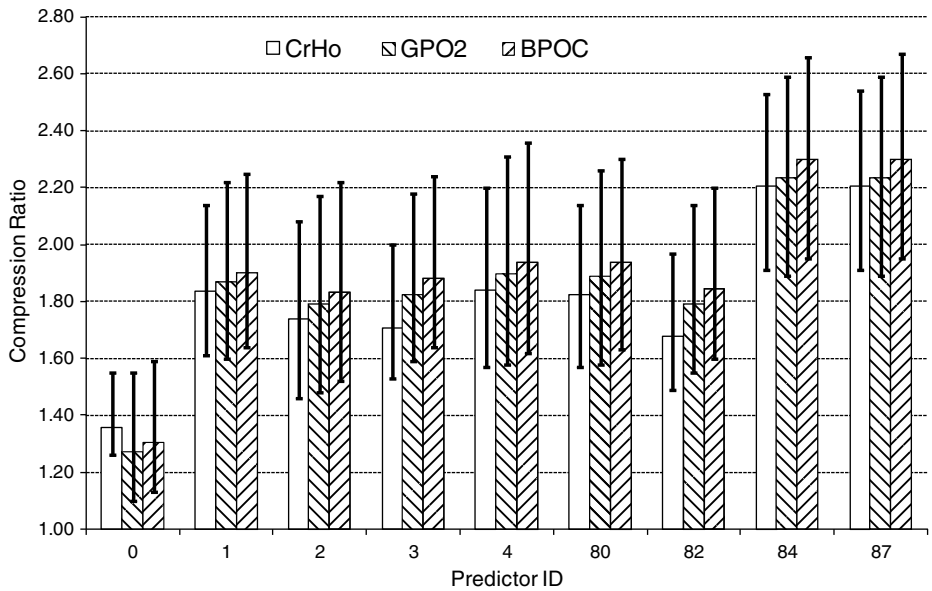


Figure 2.6 Average compression ratios of six test datacubes (bars) and their variation (lines) using the fixed coefficient predictors followed by GPO2 and BPOC entropy encoders.

encoded is smaller than 2^{Lb}. Otherwise, the length of overflow bits is $L_o = INT[\text{value}/2^{Lb}]$, where $INT[.]$ is the operator of integer. A comma (which takes 1 bit) is required in order to distinguish the variable code in this method. If the majority of data to be encoded can be expressed using only base bits as a result of selection of L_b, a high encoding efficiency can be achieved. In Fig. 2.6, the BPOC for the same residual datacube always provides a slightly better compression ratio than the GPO2 encoder.

Note that the compression ratios obtained using two entropy encoders (shaded bars) are slightly higher than Cr_{H0} (white bar). This probably results from dividing the datacubes into small blocks. Both of the entropy encoders separate the residual cubes into blocks and code them block by block. The Cr_{H0} is calculated based on $H_0(x \quad \hat{x})$, which is obtained for an entire residual cube.

The experimental results show that the 1D NNP in the cross-track direction (ID #1) provides the best decorrelation of the three 1D NNP predictors, whereas the 1D NNP in the spectral direction (ID #3) provides the worst decorrelation. This is probably because the equivalent instrument noise in the spectral domain is relatively strong in imaging spectrometers. The 2D NNP in two spatial directions (ID #4) performs better than all three 1D NNPs. The 3D, 2-point NNP in three directions (ID #80) does not perform better than predictor ID #4, and it is much more complex than ID #4. The predictor ID #82 is a 1D, 2-point spectral predictor that performs worse than all three 1D NNPs. The 5-point 3D (ID #84) and 7-point 3D (ID #87) predictors provide better decorrelation than predictor ID #4.

2.4.5.2 Compression results using variable coefficient predictors

Predictor IDs #100–109 are variable coefficient predictors. Figure 2.7 shows the average compression ratio (bar) over the six test datacubes and its variation (line) using predictor IDs #100–104, with three combinations of the two coefficients followed by the GPO2 entropy encoder. The compression results using the same predictors followed by the BPOC encoder are not shown in the figure. They are all similar to those produced by the GPO2 encoder except that the compression ratios are slightly higher. The experimental results show that predictor IDs #100–101 perform best when coefficients $A = B = 1/2$ (SID #0).

Figure 2.8 shows the average compression ratios over the six test datacubes using predictor IDs #105 through #109 followed by a GPO2 encoder. In the figure, the compression ratios of each predictor with 15 combinations (SIDs #0–14) are shown in sequence. The predictor ID #105 performs the best of the five predictors. The average compression ratio for the six test datacubes is 1.88 when coefficients $A = B = 1$ and $C = 1$ (SID #0), and is 1.82 when coefficients $A = 1/2$, $B = 1$, and $C = 1/2$ (SID #8).

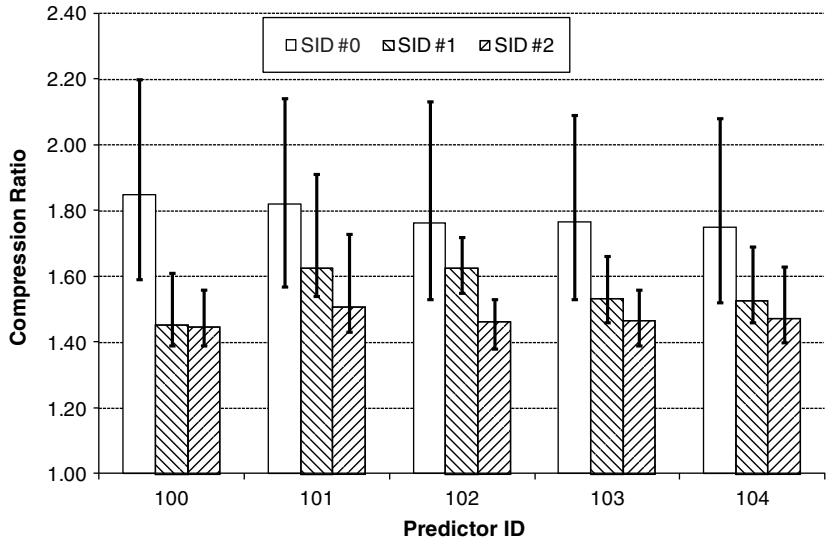


Figure 2.7 Average compression ratios of six test datacubes (bars) and their variation (lines) using predictor IDs #100–104, with three SIDs followed by a GPO2 encoder.

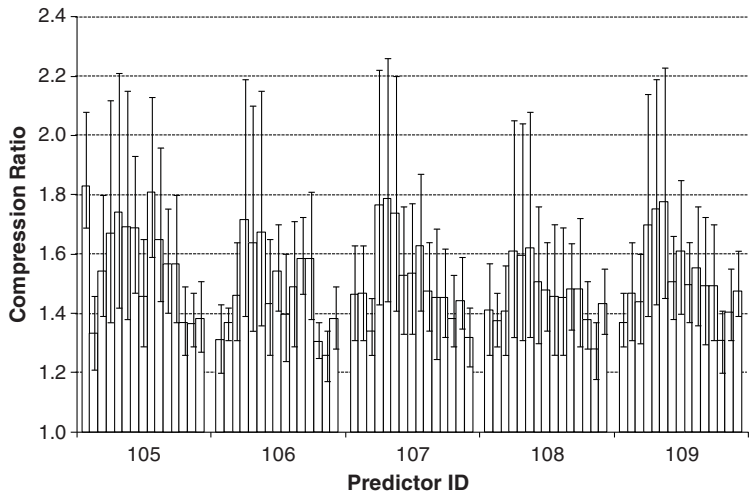


Figure 2.8 Average compression ratios of six test datacubes (bars) and their variation (lines) using predictor IDs #105–109, with fifteen SIDs followed by a GPO2 encoder.

2.4.5.3 Compression results using adaptive selection of predictor

Two predictor banks were constructed to examine the performance of the scheme by choosing the best predictors studied in Sections 2.4.1–2.4.3. They are listed in Tables 2.6 and 2.7.

Table 2.6 Predictor bank #1.

Predictor 1	$\hat{x}_{r,c,\lambda}$	$x_{r,c,\lambda-1} + x_{r,c-1,\lambda} \quad x_{r,c-1,\lambda-1}$ (ID #105, SID #0)
	$\hat{x}_{r,c,\lambda}$	$\min(x_{r,c-1,\lambda},\ x_{r,c,\lambda-1})$ if $x_{r,c-1,\lambda-1} \geq \max(x_{r,c-1,\lambda},\ x_{r,c,\lambda-1})$
Predictor 2	$\hat{x}_{r,c,\lambda}$	$\max(x_{r,c-1,\lambda},\ x_{r,c,\lambda-1})$ if $x_{r,c-1,\lambda-1} < \min(x_{r,c-1,\lambda},\ x_{r,c,\lambda-1})$
	$\hat{x}_{r,c,\lambda}$	$x_{r,c-1,\lambda} + x_{r,c,\lambda-1} \quad x_{r,c-1,\lambda-1}$ otherwise (ID #6)
Predictor 3	$\hat{x}_{r,c,\lambda}$	$\frac{1}{2}(x_{r,c-1,\lambda} + x_{r,c-2,\lambda})$ (ID #101, SID #0)
Predictor 4	$\hat{x}_{r,c,\lambda}$	$\frac{1}{2}(x_{r,c,\lambda-1} \quad x_{r,c-1,\lambda-1}) + x_{r,c-1,\lambda}$ (ID #105, SID #8)

Table 2.7 Predictor bank #2.

Predictor 1	$\hat{x}_{r,c,\lambda}$	$x_{r,c,\lambda-1} + x_{r,c-1,\lambda} \quad x_{r,c-1,\lambda-1}$ (ID 105, SID #0)
Predictor 2	$\hat{x}_{r,c,\lambda}$	$\frac{1}{2}(x_{r,c,\lambda-1} \quad x_{r,c-1,\lambda-1}) + x_{r,c-1,\lambda}$ (ID 105, SID #8)

There are four preselected predictors in predictor bank #1. Block sizes of 1, 4, 8, 12, 16, 24, 32, and 48 are tested. The residual data was encoded using the GPO2 encoder. Experimental results show that the compression ratios are very close when block sizes are between 8 and 24; the three predictor selection criteria discussed in Section 2.4.4 were used, and the criterion minimum sum yielded the best compression ratio. The average ratio reaches 2.23 when block size $L = 12$. The frequency of occurrence of each preselected predictor was also analyzed. The selection frequency of predictor 1 is over 36% when block sizes are between 8 and 24. With the same range of block sizes, the selection frequency of predictors 3 and 4 are around 24%, while the selection frequency of predictor 3 is 15%.

There are two preselected predictors in predictor bank #2. They are predictors 1 and 4 of the predictor bank #1, the two most-frequently selected predictors. Experimental results show that the predictor selection criterion—smallest maximum—provided the same compression ratios as those provided by the minimum sum criterion. The average compression ratio reached is 2.11 when block size $L = 12$. Though the overhead for recording the preset predictors in bank #2 is reduced by a factor of two, its decorrelation is slightly poorer than predictor bank #1.

In summary, a total of 99 predictors (including nine fixed coefficient predictors, five two-variable coefficient predictors with three coefficient combinations, and five three-variable coefficient predictors with 15 coefficient combinations) are examined and tested in Section 2.4. Three predictor selection criteria are proposed and tested. The residual data is encoded using the GPO2 and BPOC encoders.

2.5 Lookup-Table-Based Prediction Methods

2.5.1 Single-lookup-table prediction

Assuming a current pixel $I(x, y, \lambda)$ at the xth column, yth row, and λth band image of a datacube, make a prediction for the current pixel using all of the causal pixels in the current and previous band images. The procedure searches for the nearest-neighbor pixels in the previous band image ($\lambda - 1$) that have the same pixel value as the pixel $I(x, y, \lambda - 1)$, which is located in the same spatial position as the current pixel but in the previous band. The search is performed in reverse raster-scan order. This kind of search is referred to as a NN search. Each pixel in the previous band image ($\lambda - 1$) is searched to check if a pixel with value equal to $I(x, y, \lambda - 1)$ exists. If an equal-valued pixel $I(x', y', \lambda - 1)$ is found, then $I(x, y, \lambda)$ is predicted to have the same value as the pixel in the same position as the found pixel in the current band $I(x', y', \lambda)$. Otherwise, the predicted pixel value $I(x, y, \lambda)$ is equal to the value of the pixel in the previous band $I(x, y, \lambda - 1)$. The predictive value of the current pixel can be defined as

$$\hat{I}(x, y, \lambda) = \begin{cases} I(x', y', \lambda) & \text{if } I(x', y', \lambda - 1) = I(x, y, \lambda - 1) \\ I(x, y, \lambda - 1) & \text{otherwise.} \end{cases} \tag{2.12}$$

A lookup-table (LUT) data structure is adopted to accelerate the NN method by replacing time-consuming search procedures with a LUT operation, which uses the pixel co-located in the previous band as an index in the LUT. The nearest matching pixel is obtained by indexing the LUT. Its computational complexity is much less than the NN search. First, for each band image λ, a LUT L_λ is initialized so that the ith value of the LUT of band image λ is $L_{\lambda, i} = i, i \in [-2^{16}, 2^{16} - 1]$ for 16-bit data. Next, for all of the pixels in the scanning order, prediction, and update steps are performed. The pixel $I(x, y, \lambda - 1)$ co-located in the previous band to the current pixel $I(x, y, \lambda)$ is used as an index to the LUT. The prediction step predicts the current pixel's value to be the $I(x, y, \lambda - 1)$th value in the current band's LUT $L_{\lambda, I(x, y, \lambda - 1)}$. After prediction, the LUT is updated as follows: $L_{\lambda, I(x, y, \lambda - 1)} = I(x, y, \lambda)$. The use of the LUT corresponds to performing a search in the previous band in the reverse scanning order and making a prediction using causal pixels in the current band. Therefore, the NN method and the LUT method are identical in terms of predicted values. The residual values are obtained by subtracting the prediction values from the original values and encoded using an entropy encoder.[14]

An example illustrating the LUT search process is shown in Fig. 2.9, consisting of two consecutive band images $\lambda - 1$ and λ, each of which has 3×7 pixels and the lookup table L_λ of band image λ. In the example, pixel $I(7, 3, \lambda) = 225$ is the current pixel to be predicted. The causal pixels in the previous band are searched to find matches for the co-located pixel $I(7, 3, \lambda - 1) = 215$. Two matched pixels $I(3, 2, \lambda - 1)$ and $I(5, 2, \lambda - 1)$ are found and returned. The

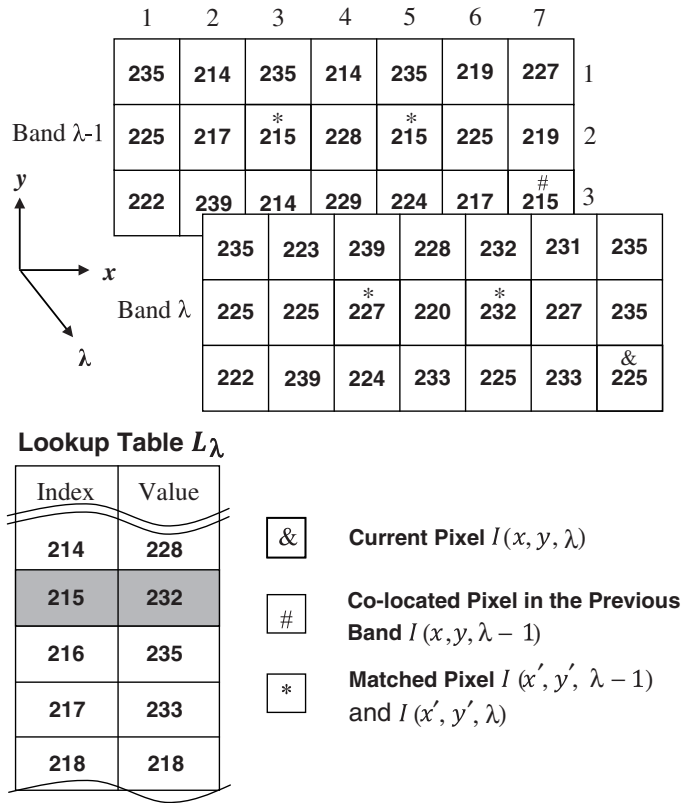


Figure 2.9 An example illustrating the lookup table search process for prediction of the current pixel $I(x, y, \lambda)$.

location $x' = 5,\ y' = 2$ of the matched pixel that is nearest to the co-located pixel is used to locate the predictive pixel in the current band. Based on Eq. (2.12), the predictive value for the current pixel is obtained: $\hat{I}(x, y, \lambda) = I(x', y', \lambda) = I(5, 2, \lambda) = 232$. A time-consuming search is avoided because the LUT directly returns the predictive value for the current pixel at index 215.

2.5.2 Locally averaged, interband-scaling LUT prediction

In the LUT method, the nearest matching pixel value $I(x', y', \lambda)$ might not be the closest to the current pixel. In the example shown in Fig. 2.9, the pixel in the current band corresponding to another matching pixel $I(3, 2, \lambda) = 227$ is closer to the current pixel value 225 than the nearest matching pixel value 232. This type of behavior of the LUT method motivated the development of the locally averaged, interband-scaling lookup table (LAIS-LUT) method,[15] which uses a predictor to guide the selection between two LUTs. The LAIS-LUT method works by first computing a LAIS estimate by scaling pixels

co-located in the previous band. The LAIS scaling factor is an average of ratios between three neighboring causal pixels in the current and previous bands:

$$\alpha = \frac{1}{3}\left[\frac{I(x \quad 1, y, \lambda)}{I(x \quad 1, y, \lambda \quad 1)} + \frac{I(x, y \quad 1, \lambda)}{I(x, y \quad 1, \lambda \quad 1)} + \frac{I(x \quad 1, y \quad 1, \lambda)}{I(x \quad 1, y \quad 1, \lambda \quad 1)}\right]. \quad (2.13)$$

The factor α is multiplied by the co-located pixel $I(x, y, \lambda \quad 1)$ to get an estimate for the current pixel:

$$\hat{I}(x, y, \lambda) = \alpha I(x, y, \lambda \quad 1). \quad (2.14)$$

LAIS-LUT prediction uses two LUTs that are similar to the one used in Fig. 2.9. The second LUT is updated with the previous entries of the first LUT. The predictive value returned by the LUT that is the closest one to the LAIS estimate is chosen as the predictor for the current pixel. If the LUTs return no match, then the LAIS estimate is used as the estimated pixel value. Figure 2.10 illustrates the LAIS-LUT search process. The LAIS estimates for the two matching pixels in the LUT example are shown in Fig. 2.9. Recall that the current pixel is $I(7, 3, \lambda) = 225$, and the causal pixels in the previous band are searched to find matches for the co-located pixel $I(7, 3, \lambda \quad 1) = 215$. The two matching pixels are $I(3, 2, \lambda) = 227$ and $I(5, 2, \lambda) = 232$ (highlighted with asterisks in Fig. 2.9). The LAIS estimate values for the two matching pixels are 222 and 216, respectively. The LAIS estimate 222 is closer to the matching pixel $I(3, 2, \lambda) = 227$, which is placed in the second LUT. Therefore, the pixel value from the second LUT is used as the predictor for the current pixel $\hat{I}(x, y, \lambda) = I(x'', y'', \lambda) = I(3, 2, \lambda) = 227$. LAIS-LUT prediction provides more-accurate predictive values than the LUT prediction method.

LAIS Estimates for LAIS-LUT

Pixel Position (x, y)	Pixel Value	LAIS Estimate
(3,2)	**227**	**222**
(5,2)	**232**	**216**

Lookup Tables L

Index	1st LUT	2nd LUT
214	**228**	**224**
215	**232***	**227***
216	**235**	**235**
217	**233**	**225**
218	**228**	**228**

Figure 2.10 LAIS estimation and the two lookup tables in the LAIS-LUT search process for predicting the current pixel $I(x, y, \lambda)$.

2.5.3 Quantized-index LUT prediction

The co-located pixel in the previous band $I(x, y, \lambda \quad 1)$ is quantized with a predefined factor (e.g., 10) before it is used as an index for the LUTs. The quantized indices reduce the size of the LUTs significantly.[48] Except for a changed LAIS approach and an additional quantization step, the locally averaged, interband-scaling quantized-index lookup table (LAIS-QLUT) is the same algorithm as the LAIS-LUT, as discussed in Section 2.5.2.

Instead of using Eq. (2.13), a slightly simpler LAIS method from the work[49] is used:

$$\alpha = \frac{I(x \quad 1, y, \lambda) + I(x, y \quad 1, \lambda) + I(x \quad 1, y \quad 1, \lambda)}{I(x \quad 1, y, \lambda \quad 1) + I(x, y \quad 1, \lambda \quad 1) + I(x \quad 1, y \quad 1, \lambda \quad 1)}. \quad (2.15)$$

There are two separate variants of the LAIS-QLUT method. The first variant is called LAIS-QLUT-OPT, which selects the optimal uniform quantization factor for each band image. An exhaustive search of all possible quantization values is performed to find the optimal quantization factor for each band image. The quantization factor selection is based on which quantization factor achieves the best compression efficiency for that specific band. The excessive time complexity of the LAIS-QLUT-OPT method can be decreased slightly by computing entropy of the residual image instead of actually encoding residuals for the determination of the optimal quantization factor.

The second variant is called LAIS-QLUT-HEU; it uses constant quantization factors that are selected using a heuristic. The heuristic selects the constant quantization factors to be the band-wise mean values of the optimal quantization factors of a datacube. A division operation required by the quantization represents the only increase in the time complexity of the LAIS-QLUT-HEU method compared to the LAIS-LUT version.

2.5.4 Multiband LUT prediction

The LUT and LAIS-LUT methods have been generalized to a multiband and multi-LUT method.[50] In the extended method, the prediction of the current band relies on N previous bands. LUTs are defined based on each of the previous bands, and each band contains M LUTs. There are thus $N \times M$ different predictors from which to choose. The decision among one of the possible prediction values is based on the closeness of the values contained in the LUTs to a reference prediction.

Two different types of purely spectral, multiband prediction estimates were proposed. These LUT-based compression methods based on spectral relaxation-labeled prediction (S-RLP) and spectral fuzzy-matching pursuits (S-FMP) are referred to as S-RLP-LUT and S-FMP-LUT, respectively. One of the reference predictors is crisp, and the other one is fuzzy. The first

method is spectral relaxation labeled prediction.[5] The method partitions image bands into blocks. A predictor out of a set of predictors is selected for prediction. In the S-FMP method, pixels are given degrees of membership to predictors. The membership function of a pixel to a predictor is inversely related to the average prediction error produced by that predictor in a causal neighborhood of that pixel. The causal neighborhoods of each pixel are clustered using fuzzy c-means clustering. Each predictor is then recalculated based on the membership of pixels to it. The procedure is analogous to relaxation labeling, in which the labeling is not crisp but fuzzy. The final sets of refined predictors, one per wavelength, are transmitted as side information.

2.6 Vector-Quantization-Based Prediction Methods

An ultraspectral sounder produces 3D datacubes, two of which are spatial dimensions corresponding to scan lines and cross-track footprints within each scan line. The third dimension corresponds to thousands of infrared components (often referred to as channels) of each footprint. The ultraspectral sounder data is generated using a Michelson interferometer or a grating spectrometer for retrieving atmospheric temperature, moisture and trace gases profiles, surface temperature and emissivity, as well as cloud and aerosol optical properties. Due to the high volume of the ultraspectral sounder data and the sensitivity of the compression error to downstream data processing, lossless compression is highly desirable. A fast precomputed vector quantization (FPVQ) scheme is proposed to compress ultraspectral sounder data.[51]

This method first converts the ultraspectral sounder data into Gaussian-distribution, predictive residual data using linear prediction and then groups the predictive residual error data based on their bit-lengths. Vector quantization with a set of precomputed, 2^k-dimensional, normalized Gaussian codebooks of size-2^m codewords is then performed, and bit allocation for all subgroups is performed via a new bit-allocation scheme that reaches an optimal solution under the constraint of a given total bitrate. The precomputed-codebook FPVQ method eliminates the time for online codebook generation, and the precomputed codebooks do not need to be sent to the decoder as side information.

The FPVQ method includes five steps.

2.6.1 Linear prediction

This step reduces the data variance and makes the data close to the Gaussian distribution. Linear prediction employs a set of neighboring pixels to predict the current pixel. For ultraspectral sounder data, the spectral correlation is generally much stronger than the spatial correlation.[52] It makes sense to

predict the value at a spectral channel as a linear combination of neighboring channels. The problem can be formulated as

$$\hat{\mathbf{X}}_i = \sum_{k\ 1}^{N_\lambda} c_k \hat{\mathbf{X}}_{i\ \ k} \quad \text{or} \quad \hat{\mathbf{X}}_i = \mathbf{X}_\lambda \mathbf{C}, \tag{2.16}$$

where $\hat{\mathbf{X}}_i$ is the vector with N_λ elements of the current channel representing a 2D spatial frame, $\mathbf{X}_\lambda$ the matrix consisting of N_λ neighboring channels, and $\mathbf{C}$ is the vector of the prediction coefficients. The prediction coefficients are obtained from

$$\hat{\mathbf{X}}_i = \mathbf{C} = (\boldsymbol{X}_\lambda^{\boldsymbol{T}} \boldsymbol{X}_\lambda)^{pi} (\boldsymbol{X}_\lambda^{\boldsymbol{T}} \hat{\boldsymbol{X}}_i), \tag{2.17}$$

where the superscript *pi* represents the pseudo-inverse that is robust against the case of the matrix being ill-conditioned. The prediction error vector (or residual) is the difference between the original channel vector and the predicted vector.

2.6.2 Grouping based on bit-length

The channels that have the same bit-length of prediction error are assigned to the same group. Given n_d bit-lengths, the channels are grouped such that

$$n_c = \sum_{i\ 1}^{n_d} n_i, \tag{2.18}$$

where n_i is the number of channels in the ith group. The precomputed VQ codebooks are applied to each group independently.

2.6.3 Vector quantization with precomputed codebooks

Precomputed codebooks are generated to avoid online codebook training. After linear prediction, the prediction error of each channel is close to a Gaussian distribution with a different standard deviation. The codebooks of size-2^m codewords for 2^k-dimensional, normalized Gaussian distributions are precomputed using the Linde–Buzo–Gray (LBG) algorithm. It is known that any number of channels n_i in the ith group can be represented as a linear combination of 2^k, as follows:

$$n_i = \sum_{k\ 0}^{\log_2 n_i} d_{i,k} 2^k, d_{i,k} = 0 \text{ or } 1. \tag{2.19}$$

All of the 2^k channels with $d_{i,k} = 1$ form a subgroup within the ith bit-length group. The total number of the subpartitions is

$$n_s = \sum_{i\ 1}^{n_d} n_{i,b,} \tag{2.20}$$

where $n_{i,b}$ is the number of subgroups within the ith bit-length group. The codebook is the precomputed, normalized Gaussian codebook that is scaled by the standard deviation spectrum of the datacube.

2.6.4 Optimal bit allocation

The number of bits for representing the quantization residuals within each subgroup depends on the dimension of the subgroup and the codebook size. An improved bit-allocation scheme is proposed that guarantees an optimal solution under the constraint. It formulates the problem to minimize the expected total number of bits for the quantization errors in the ith partition and jth subpartitions and for the quantization indices. The proposed bit-allocation scheme consists of nine steps. The first six steps are for marginal analysis. The idea of the marginal analysis of the bit-allocation scheme is similar to the algorithms developed by Riskin[53] (with convexity assumption) and those Cuperman[54] developed for lossy compression. Steps 7–9 compare neighboring bit allocations along the hyperplane of the constraint to reach a local minimum of the cost function. The proposed scheme is much faster in the sense that it only needs to update the margin for the subgroup that gives the minimum margin for both convex and nonconvex cases. The difference between the proposed scheme and those by Riskin and Cuperman is that the proposed scheme allows comparison of neighboring bit allocations along the hyperplane of the constraint to reach a local minimum of the cost function.

2.6.5 Entropy coding

Arithmetic coding[38] is selected as an entropy encoder. After the VQ stage, a context-based adaptive arithmetic coder is used to encode the data quantization indices and quantization errors.

The proposed FPVQ-based lossless compression method has been applied to 3D ultraspectral sounder data acquired by the Atmospheric Infrared Sounder (AIRS) instrument.[55] Ten granules were tested—each granule contains a 2D spatial image consisting of 135 scan lines by 90 cross-track footprints per scan line. Each footprint corresponds to a spectrum of 2,378 infrared channels within a wavelength range of 3.74–15.4 μm. Thirty-two predictors are used in linear prediction. The distributions of prediction residuals appear close to Gaussian and indicate that they are well decorrelated. The compression ratio produced by the FPVQ method for test AIRS ultraspectral granules is 3.2:1 ~ 3.4:1. The compression ratios produced by JPEG2000 and by linear prediction followed by entropy encoding of the residuals without the VQ method are 2.2:1 ~ 2.5:1 and 2.8:1 ~ 3.2:1, respectively. The compression ratio produced by the FPVQ method is approximately 1.4 times higher than that of JPEG2000 and 1.08 times higher than that of linear prediction without the VQ method.[51]

2.7 Band Reordering

In the prediction-based lossless compression of hyperspectral datacubes, band reordering has also been proposed to obtain improved performance. This process organizes the spectral bands of a datacube to be compressed in such a way as to maximize the correlation of adjacent bands to optimize the performance of the prediction step.

Band reordering is motivated by the fact that the previous band may not necessarily be the most-correlated (or similar) band to the current band for prediction, which is a necessary element. The most-correlated band can only be found among the bands that have already been encoded.

In order to describe how similar bands l and i are, and hence the quality of the prediction of band i using band l, a similarity metric $s_{l,i}$ needs to be defined. Tate[25] used this metric, and a number of bits are needed to encode band i from band l. In Toivanen et al.,[26] $s_{l,i}$ is used as the correlation coefficient between band l and band i; $s_{l,i}$ is also used as the correlation coefficient in Zhang and Liu,[27] but band grouping is introduced to limit the complexity.

In the prediction-based lossless compression method, band reordering is an option that a user may or may not use without affecting the algorithm design and implementation. It has been reported that not all datacubes to be compressed benefit from band reordering.[56] In some cases the gain is reasonable, while in other cases it is negligible. It is up to the user to decide whether the effort in calculating an optimal reordering is worth the performance gain.

For optimal band reordering, a specific ordering has to be computed and used for each datacube. This increases the computational complexity. To reduce this computational complexity, a different approach to band reordering is proposed.[56] In this approach, a reasonably "good" band reordering is computed on the ground based on selected typical sample data, and then the LUT of the "good" band ordering is uploaded to the satellite for use in the compression of all datacubes. The reason being that the optimal ordering arguably depends on both the sensor and the scene, with the former potentially dominating the ordering. If the contribution of the scene to the optimal ordering is small, then a per-sensor ordering would be almost as good as the optimal per-image ordering. The authors have shown that this is indeed the case. Thus, this approach achieves the benefit of improving the performance by using a good optimal reordering without adding complexity to the algorithm, as all computations needed to reorder bands have been performed on the ground using sample data to generate the LUT.

2.8 Transform-Based Lossless Compression Using the KLT and DCT

A combination of the KLT and DCT is adopted to implement a lossy-to-lossless hyperspectral image compression.[31] A reversible KLT is utilized for spectral decorrelation. For spatial decorrelation, instead of applying a DWT, a reversible DCT is used.

The KLT (sometimes referred to as principal components analysis) is an optimal orthogonal transform for compacting the energy of a vector or image into its first n components. The transform matrix is obtained by calculating the eigenvectors of the covariance of the input vector or image. It has been widely used in image processing. The studies on reversible KLTs for spectral decorrelation have been reported. Hao and Shi[57] proposed a reversible integer KLT, and the late Galli and Salzo[58] have improved it.

The DCT is widely used in image and video compression, such as JPEG, MPEG, and H.26X formats. It has its own unique advantages, such as low memory usage, flexibility at the block-by-block level, parallel processing, etc. However, a key problem of DCT-based coding is the so-called "block effect" at low bitrates due to the encoder processing each block independently. In order to reduce the block effect of the DCT compression, some de-blocking methods based on filtering have been proposed in which some low-pass filters are applied to the boundary pixels. However, filtering-based methods usually blur image content. Tran et al.[32] have designed pre- and postfilters to improve the performance of the DCT, calling the combination of the filters a time-domain lapped transform (TDLT), which performs better than the DWT in energy compatibility and lossy compression. However, it does not perform well in lossless compression, where the reversible transform is required. Wang et al.[31] introduced a reversible TDLT using an integer operation to replace the integer wavelet transform.

In order to reduce the high computational complexity, the low-complexity KLT proposed by Penna is used.[59] In the proposed low-complexity KLT, all of the processing is not carried out on the complete set of spectral vectors but rather on a randomly selected subset of vectors. The complexity of calculating the covariance matrix becomes $O(\rho B^2 MN)$ rather than $O(B^2 MN)$, i.e., it is reduced by a factor ρ ($\rho = 0.01$, for example). The performance of the low-complexity KLT is very similar to the conventional KLT, but the computational complexity is reduced significantly. To further reduce the computational complexity the evaluation of the covariance matrix is simplified by subsampling the input vectors at a rate of 100:1.

In terms of the matrix factorization theory,[60] a nonsingular matrix can be factorized into a product of at most three triangular elementary reversible matrices (TERMs). If the diagonal elements of the matrices are equal to 1, the reversible integer-to-integer transform can be achieved by multilifting. In this

way, these integer-to-integer DCT and KLT are implemented using multilifting based on matrix factorization.

The forward DCT is defined as follows:

$$C(k) = \varepsilon_k \sqrt{\frac{2}{N}} \sum_{n\ 0}^{N\ \ 1} x(n) \cos\left[(2n+1)\frac{\pi k}{2N}\right], \tag{2.21}$$

for $k = 0, 1, \ldots, N \quad 1$, where $\varepsilon_k = 1/\sqrt{2}$ if $k = 0$, and $\varepsilon_k = 1$ otherwise.

In the conventional DCT compression algorithms, an image is normally divided into blocks of 8×8 pixels. Blocking artifacts occur because each block is compressed independently. To overcome the blocking defects, eight-point pre- and postfilters are used for the eight-point DCT. Prefiltering acts as a flattening operation on two adjacent blocks. The pixels in a block are more homogeneous after prefiltering as a result of better energy concentration. Before transforming, the filtered DCT matrices are factorized into TERMs. An integer-to-integer DCT transform is achieved with measures for reducing blocking artifacts.

The reversible-integer KLT and DCT are implemented in the same way. The integer transform can be achieved by shifting and adding without a multiplier when the floating-point lifting coefficients are replaced by fractions whose dominators are powers of two. For example, calculating $75/256 = 1/4 + 1/32 + 1/64 + 1/256$ can be converted to the operation of 1/4, 1/32, 1/64, and 1/256, which can be implemented by bit-shifting. Experimental results show that the multilifting-based filter approximates the floating-point filter very well.

A lossless compression algorithm that combines the reversible time-domain lapped transform (RTDLT) and reversible KLT has been proposed.[31] This algorithm uses the RTDLT to decorrelate spatial redundancy and the KLT to decorrelate spectral redundancy. Three AVIRIS datacubes were tested using three algorithms—RTDLT/KLT, 3D SPECK,[61] and JPEG2000 multicomponent (JPEG2000-MC)[62]—in order to assess the decorrelation capability of the combined reversible KLT and RTDLT transform algorithm against the other transform-based methods. The two transforms compared are the asymmetric 3D-5/3DWT and 2D-5/3DWT used in an early work.[61] The RTDLT/KLT algorithm outperforms the asymmetric 3D-5/3DWT with a 7.4–8.6% better compression ratio,[31] comparable to the 2D-5/3DWT + reversible-KLT. The RTDLT/KLT algorithm outperforms the JPEG2000-MC with a 9.3–10.8% better compression ratio. However, the RTDLT/KLT algorithm underperforms the M-CALIC algorithm,[9] which is a prediction-based lossless compression, with a 4.2–12.2% worse compression ratio.

2.9 Wavelet-Transform-Based Methods

Wavelet transforms are used as a means to repackage the energy of an image or datacube for better encoding in that low-frequency components

normally contain the majority of signal energy and are thus more important than high-frequency components during reconstruction. Wavelet transforms not only provide excellent repackaging of information in terms of frequency but also retain much of the spatial and spectral structure of the original image.

2.9.1 Wavelet decomposition structure

For a 2D image, the wavelet transform decomposes the image into horizontal subband H, vertical subband V, diagonal subband D, and baseband B images using the low-pass filter (LPF) and high-pass filter (HPF) in the filter bank. The size of the subband image is one-fourth of the original image. Multiple stages of decomposition can be cascaded together by recursively decomposing the baseband image. The subband images are usually arranged in a pyramidal form, as illustrated in Fig. 2.11.

For hyperspectral datacubes, a 3D wavelet transform is implemented by applying the 1D transform separately, i.e., a 1D transform in the spatial-row, spatial-column, and spectral-band directions. The addition of a third dimension permits several options for the order of decomposition. For example, one level of decomposition of the datacube can be performed along each direction, followed by further decomposition of the low-pass subband, leading to a dyadic decomposition (illustrated in Fig. 2.12). This dyadic decomposition structure is the most-straightforward 3D generalization of the 2D dyadic decomposition of Fig. 2.11(a).

An alternative 3D wavelet transform referred to as a wavelet-packet transform is also used. In this transform, each spectral band image of the datacube is first decomposed using a 2D transform, followed by a 1D decomposition in the spectral direction. With this approach, an m-level spatial decomposition followed by an n-level spectral decomposition is achieved, where it is possible for $m \neq n$. For example, the wavelet-packet transform depicted in Fig. 2.12(b) uses a three-level decomposition ($m = n = 3$) in all directions. Comparing the two decomposition structures, the wavelet-packet transform is more flexible because the spectral decomposition can be better tailored to the data than in the dyadic transform. Thus, this wavelet-packet decomposition typically yields more-efficient coding for hyperspectral datacubes than the dyadic decomposition does.

2.9.2 Lossy-to-lossless compression: 3D set-partitioning embedded block

A 3D set-partitioning embedded block (3D-SPECK)[63] has been proposed by modifying and extending the 2D-SPECK. The 3D-SPECK algorithm uses either the integer wavelet transform or floating-point wavelet transform. The

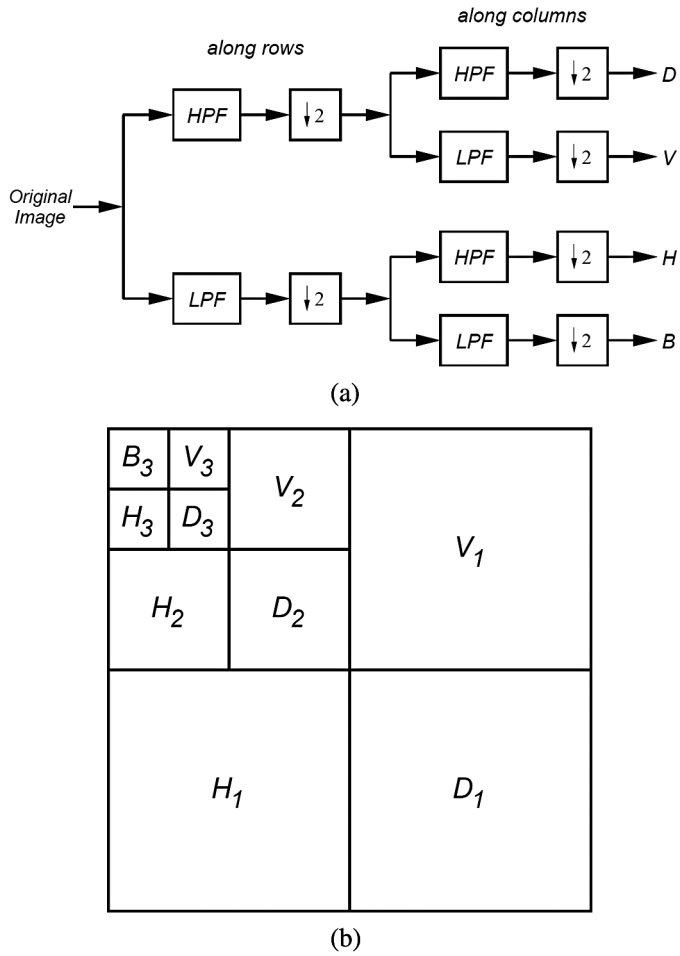


Figure 2.11 2D wavelet transform decomposition: (a) one-level decomposition with a low-pass filter and a high-pass filter applied to the columns and rows independently; (b) pyramid-form arrangement of subband images after three-level, 2D wavelet transform decomposition.

integer wavelet transform enables lossless decompression from the same bitstream. Wavelet-packet structure and coefficient scaling are used to make the integer filter transform approximately unitary.

For a hyperspectral datacube that has been transformed using either an integer or floating-point wavelet transform, the transformed subband images form a hierarchical pyramidal structure, as shown in Fig. 2.12(a), with the topmost level being the root. The finest pixels lie at the bottom level of the pyramid, whereas the coarsest pixels lie at the top (root) level. A wavelet subband image is represented by an indexed set of transformed

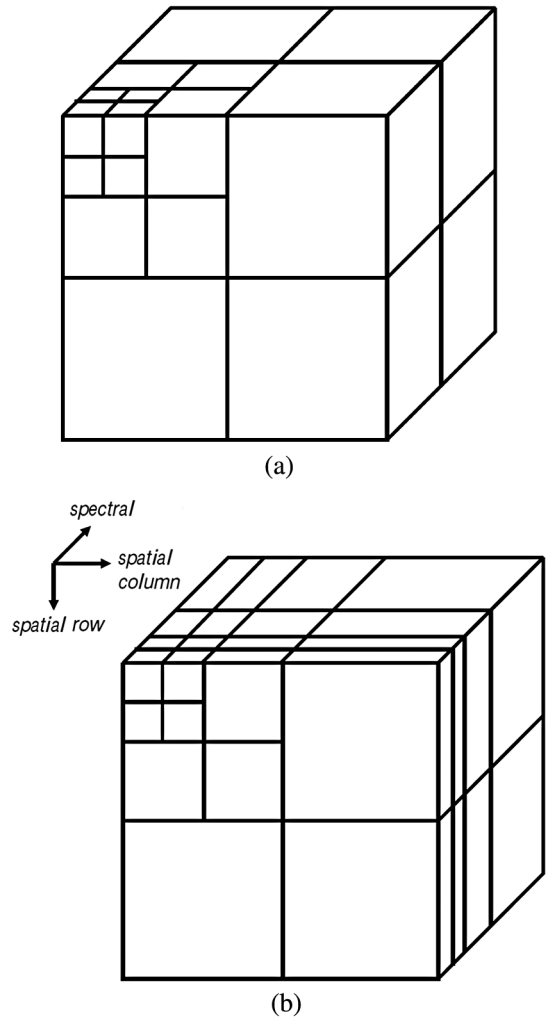


Figure 2.12 3D wavelet transform decomposition: (a) three-level dyadic wavelet transform decompositions; (b) 3D wavelet packet transform with $m = 3$ spatial decompositions and $n = 3$ spectral decompositions.

coefficients $\{c_{i,j,k}\}$, located at pixel position (i,j,k) in the transformed subband image.

The coefficients are grouped together in sets that comprise regions in the transformed subband images. The 3D-SPECK has only one type of set S that is significant with respect to n if

$$\max_{(i,j,k)\in S}|c_{i,j,k}| \geq 2^n, \tag{2.22}$$

where $c_{i,j,k}$ denotes the transformed coefficients at coordinate (i,j,k), otherwise it is insignificant. The significance function of a set S is defined as

$$\Gamma_n(S) = \begin{cases} 1 & \text{if } 2^{n+1} > \max_{(i,j,k)\in S}|c_{i,j,k}| \geq 2^n \\ 0 & \text{otherwise.} \end{cases} \tag{2.23}$$

In the 3D-SPECK, each subband image in the pyramidal structure is taken as a codeblock and referred to as set S, whose dimensions vary. The dimension of a set S depends on the dimension of the original images and the subband level of the pyramidal structure at which the set lies. The size of set S is the number of elements in the set. There are two lists of the sets in the 3D-SPECK:

1. A list of insignificant sets (LIS), which contains S sets of varying sizes; and
2. A list of significant pixels (LSP), which contains pixels that have been found significant with respect to the threshold n.

The 3D-SPECK consists of four steps: (1) initialization, (2) sorting pass, (3) refinement pass, and (4) quantization. The initialization step set the initial values for output $n = \log_2\max|c_{i,j,k}|$, LSP = 0, and LIS = {all subbands of transformed images of wavelet coefficients}. In the sorting pass step, the 3D-SPECK processes a set S to test its significance with respect to the threshold n. If not significant, it stays in the LIS. Otherwise, it is partitioned into eight approximately equal subsets $O(S)$. The 3D-SPECK then treats each of these subsets as a new S set and tests its significance. This process continues recursively until it reaches the pixel level where the significant pixel in the original set S is located. The algorithm then sends the significant pixel to the LSP, outputs a bit 1 to indicate the significance of the pixel, and outputs another bit to represent the sign of the pixel. In the refinement pass step, the algorithm outputs the nth-most-significant bit of the absolute value of each entry (i, j, k) in the LSP, except for those included in the just-completed sorting pass. The procedure refines significant pixels that were found during previous passes progressively. The last step is to decrease n by 1 and return to the sorting pass of the current LIS, making the whole process run recursively.

The adaptive arithmetic coding algorithm is used to encode the significance map. The significance test result of the first subset is encoded without any context, and the significance test result of the second subset is encoded by using the context of the first coded subset, and so on.

2.9.3 Lossy-to-lossless compression: 3D embedded zeroblock coding

A wavelet-transform-based, lossy-to-lossless compression algorithm for hyperspectral images has been proposed.[65] This algorithm modifies the motion-compensated 3D embedded zeroblock coding (MC 3D-EZBC)[66] that was developed for video coding by removing its motion-compensation part,

and it replaces the DWT with a reversible 3D integer wavelet-packet transform to achieve lossy-to-lossless compression.

Zeroblock coding is based on the set partitioning in hierarchal trees (SPIHT) technique.[67] The adaptive quadtree splitting method[68] is used to separate the significant and insignificant (i.e., zero) coefficients and then encode a block of zero pixels into one symbol. In this method, a quadtree representation of WT coefficients is first established for individual subband images. The bottom level of a quadtree consists of the wavelet coefficients. A single node at the top level of a quadtree (the root node) corresponds to the maximum amplitude of all of the WT coefficients. To start with, the root is the only insignificant node to process. Each quadtree node is split into four insignificant descendent nodes of the next-lowest level once it is confirmed as insignificant with respect to the threshold of the current bit-plane coding pass. The same splitting process is recursively applied to the individual descendent nodes until the bottom level of the quadtree is reached. In this way, high-energy areas and regions of all-zero coefficients are compactly represented.

Both the 3D-EZBC and 3D-SPECK methods use quadtree splitting; the main difference between the two algorithms is that the former utilizes a context-modeling scheme to efficiently encode the quadtree structure. With the context models built separately for the individual lists, statistical characteristics of the individual quadtree and subband images at different levels can be more-accurately modeled. Another difference between them is that the lists in the EZBC are all separately established for quadtree and subband images at different levels. Therefore, all nodes in a list come from the same quadtree level and subband image. Another advantage of the 3D EZBC is that it can be applied to resolution-scalable coding applications once a proper bitstream parsing method is set up.

The coding procedure of the 3D-EZBC includes three steps: (1) a 3D integer wavelet-packet transform; (2) quadtree-based, set-partitioning zero-block coding; and (3) context-based, adaptive arithmetic coding. The 3D-EZBC was implemented by decomposing a hyperspectral datacube using a wavelet-package transform (WPT) with a 5/3 integer wavelet filter. A four-level spatial WPT and a four-level spectral WPT, as shown in Fig. 2.12(b), were applied.

The implemented algorithm was tested using four AVIRIS hyperspectral datasets (publically available at NASA's website):[69] Lunar Lake, Jasper Ridge, Low-Altitude, and Cuprite. For each of the four datasets, only scene 1 was tested. The testing datacubes were a subset with a size of 256 pixels $\times$ 256 pixels $\times$ 224 bands extracted from the original datacubes. Figure 2.13 displays the lossless compression ratios produced by the 3D-EZBC and other WT-based algorithms, including the JPEG2000-MC,[62] 3D-SPECK,[63] 3D set partitioning in hierarchical trees (3D-SPIHT),[64] and asymmetric tree

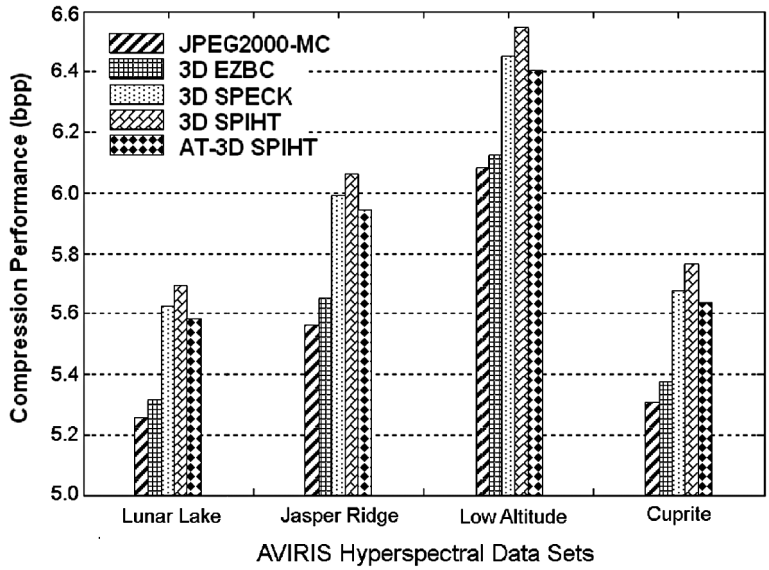


Figure 2.13 Bitrates of wavelet-transform-based, 3D lossless compression algorithms on four AVIRIS hyperspectral datasets.

3D-SPIHT (AT-3D SPIHT).[70] The JPEG2000-MC tested here also used a four-level spatial WPT and a four-level spectral WPT. It can be seen from the figure that the JPEG2000-MC produces the shortest bitrates compared to the four WT-based, 3D lossless compression algorithms for the testing AVIRIS datasets. The 3D-EZBC is the second best, and the 3D-SPIHT is the worst.

Just like the KLT+DCT-transform-based, lossless compression algorithms mentioned in Section 2.8, the WT-based, lossless compression algorithms underperform the prediction-based lossless compression algorithms.

References

1. Roger, R. E. and M. C. Cavenor, "Lossless compression of AVIRIS images," *IEEE Trans. Image Process.* **5**(5), 713–719 (1996).
2. Mielikainen, J., A. Kaarna, and P. Toivanen, "Lossless hyperspectral image compression via linear prediction," *Proc. SPIE* **4725**, 600 (2002) [doi: 10.1117/12.478794].
3. Aiazzi, B., P. Alba, L. Alparone, and S. Baronti, "Lossless compression of multi/hyperspectral imagery based on a 3-D fuzzy prediction," *IEEE Trans. Geosci. Remote Sens.* **37**(5), 2287–2294 (1999).
4. Aiazzi, B., L. Alparone, and S. Baronti, "Near-lossless compression of 3-D optical data," *IEEE Trans. Geosci. Remote Sens.* **39**(11), 2547–2557 (2001).

5. Aiazzi, B., L. Alparone, S. Baronti, and C. Lastri, "Crisp and fuzzy adaptive spectral predictions for lossless and near-lossless compression of hyperspectral imagery," *IEEE Geosci. Remote Sens. Lett.* **4**(4), 532–536 (2007).
6. Mielikainen, J. and P. Toivanen, "Clustered DPCM for the lossless compression of hyperspectral images," *IEEE Trans. Geosci. Remote Sens.* **41**(12), 2943–2946 (2003).
7. Mielikainen, J. and P. Toivanen, "Parallel implementation of linear prediction model for lossless compression of hyperspectral airborne visible infrared imaging spectrometer images," *J. Electron. Imaging* **14**(1), 013010 (2005).
8. Wu, X. and N. Memon, "Context-based lossless interband compression - Extending CALIC," *IEEE Trans. Image Process.* **9**(6), 994–1001 (2000).
9. Magli, E., G. Olmo, and E. Quacchio, "Optimized onboard lossless and near-lossless compression of hyperspectral data using CALIC," *IEEE Geosci. Remote Sens. Lett.* **1**(1), 21–25 (2004).
10. Rizzo, F., B. Carpentieri, G. Motta, and J. A. Storer, "Low-complexity lossless compression of 589 hyperspectral imagery via linear prediction," *IEEE Signal Process. Lett.* **12**(2), 138–141 (2005).
11. Slyz, M. and L. Zhang, "A block-based inter-band lossless hyperspectral image compressor," *Proc. Data Compression Conf. 2005*, 427–436 (2005).
12. Wang, H., S. Babacan, and K. Sayood, "Lossless hyperspectral image compression using context-based conditional averages," *Proc. Data Compression Conf. 2005*, 418–426 (2005).
13. Jain, S. and D. Adjeroh, "Edge-based prediction for lossless compression of hyperspectral images," *Proc. Data Compression Conf. 2007*, 153–162 (2007).
14. Mielikainen, J., "Lossless compression of hyperspectral images using lookup tables," *IEEE Signal Processing Letters* **13**(3), 157–160 (2006).
15. Huang, B. and Y. Sriraja, "Lossless compression of hyperspectral imagery via lookup tables with predictor selection," *Proc. SPIE* **6365**, 63650L (2006) [doi: 10.1117/12.690659].
16. Ryan, M. and J. Arnold, "The lossless compression of AVIRIS images by vector quantization," *IEEE Trans. Geosci. Remote Sens.* **35**(3), 546–550 (1997).
17. Mielikainen, J. and P. Toivanen, "Improved vector quantization for lossless compression of AVIRIS images," *Proc. XI Eur. Signal Process. Conf.*, 495–497 (2002).
18. Linde, Y., A. Buze, and R. M. Gray, "An algorithm for vector quantizer design," *IEEE Transactions on Communication* **28**, 84–95 (1980).

19. Motta, G., F. Rizzo, and J. Storer, "Partitioned vector quantization application to lossless compression of hyperspectral images," *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.* **1**, 553–556 (2003).
20. Canta, G. R. and G. Poggi, "Kronecker-product gain-shape vector quantization for multispectral and hyperspectral image coding," *IEEE Trans. Image Process.* **7**(5), 668–678 (1998).
21. Gelli, G. and G. Poggi, "Compression of multispectral images by spectral classification and transform coding," *IEEE Trans. Image Process.* **8**(4), 476–489 (1999).
22. Abousleman, G. P., M. W. Marcellin, and B. R. Hunt, "Hyperspectral image compression using entropy-constrained predictive trellis coded quantization," *IEEE Trans. Image Process.* **6**(4), 566–573 (1997).
23. Motta, G., F. Rizzo, and J. A. Storer, "Compression of hyperspectral imagery," *Proc. 2003 Data Comp. Conf.*, 333–342 (2003).
24. Rizzo, F., B. Carpentieri, G. Motta, and J. A. Storer, "High performance compression of hyperspectral imagery with reduced search complexity in the compressed domain," *Proc. 2004 Data Comp. Conf.*, 479–488 (2004).
25. Tate, S. R., "Band ordering in lossless compression of multispectral images," *IEEE Trans. Computers* **46**(4), 477–483 (1997).
26. Toivanen, P., O. Kubasova, and J. Mielikainen, "Correlation-based band-ordering heuristic for lossless compression of hyperspectral sounder data," *IEEE Geosci. Remote Sens. Lett.* **2**(1), 150–154 (2005).
27. Zhang, J. and G. Liu, "An efficient reordering prediction-based lossless compression algorithm for hyperspectral images," *IEEE Geosci. Remote Sens. Lett.* **4**(2), 283–287 (2007).
28. Kaarna, A., "Integer PCA and wavelet transforms for lossless compression of multispectral images," *Proc. Int. Geosc. & Rem. Sen. Symp. 2001*, 1853–1855 (2001).
29. Mielikainen, J. and A. Kaarna, "Improved back end for integer PCA and wavelet transforms for lossless compression of multispectral images," *Proc. 16th Int. Conf. Pattern Recog.*, 257–260 (2002).
30. Adams, M. D. and F. Kossentini, "Reversible integer-to-integer wavelet transforms for image compression: performance evaluation and analysis," *IEEE Trans. Image Proc.* **9**(6), 1010–1024 (2000).
31. Wang, L. et al., "Lossy-to-Lossless Hyperspectral Image Compression Based on Multiplierless Reversible Integer TDLT/KLT," *IEEE Geosci. Remote Sens. Lett.* **6**(3), 587–591 (2009).
32. Tran, T. D., J. Liang, and C. Tu, "Lapped transform via time-domain pre- and post-processing," *IEEE Trans. Signal Process.* **51**(6), 1557–1571 (2003).

33. Baizert, B., M. Pickering, and M. Ryan, "Compression of hyperspectral data by spatial/spectral discrete cosine transform," *Proc. Int. Geosci. Remote Sens. Symp.* **4**, 1859–1861 (2001).
34. Bilgin, A., G. Zweig, and M. Marcellin, "Three-dimensional image compression with integer wavelet transforms," *Appl. Opt.* **39**(11), 1799–1814 (2000).
35. Weinberger, M. J., G. Seroussi, and G. Sapiro, "LOCO-I: a low complexity, context-based lossless image compression algorithm," *Proceedings of the 1996 IEEE Data Compression Conference*, 140–149 (1996).
36. Wu, X. and N. Memon, "CALIC-A context based adaptive lossless image codec," *Proc. 1996 IEEE International Conference on Acoustics, Speech, and Signal Processing*, 1890–1893 (1996).
37. Grangetto, M. et al., "Optimization and Implementation of the Integer Wavelet Transform for Image Coding," *IEEE Trans. Image Process.* **11** (6), 596–604 (2002).
38. Witten, I. H., R. M. Neal, and J. G. Cleary, "Arithmetic coding for data compression," *Communications of the ACM* **30**(6), 520–540 (1987).
39. Golomb, S. W., "Run-length encodings," *IEEE Trans. Information Theory* **12**(3), 399–401 (1966).
40. Rice, R. F. and J. R. Plaunt, "Adaptive variable-length coding for efficient compression of spacecraft television data," *IEEE Trans. Comm.* **19**(6), 889–897 (1971).
41. Kiely, A. "Simpler Adaptive Selection of Golomb Power-of-Two Codes," *NASA Tech Briefs - November 2007*, 28–29 (2007).
42. Qian, S.-E., A. B. Hollinger, and Y. Hamiaux, "Study of real-time lossless data compression for hyperspectral imagery," *Proc. IGARSS'1999 IEEE Int. Geosc. & Remote Sen. Symp.* **4**, 2038–2042 (1999).
43. Lossless Data Compression, CCSDS Recommendation for Space Data System Standards, Blue Book 120.0-B-1, May 1997.
44. Lossless Data Compression, CCSDS Report Concerning Space Data System Standards, Green Book 120.0-G-1, May 1997.
45. Weinberger, M., G. Seroussi, and G. Sapiro, "The LOCO-I lossless image compression algorithm: Principles and standardization into JPEG-LS," *IEEE Trans. Image Process.* **9**(8), 1309–1324 (2000).
46. Sellers, P. J. et al., "The Boreal Ecosystem-Atmosphere Study (BOREAS): An Overview and Early Results," *22nd Conference on Agricultural and Forest Meteorology*, Atlanta, GA (Jan. 1996).
47. Qian, S.-E. et al., "Difference Base-bit Plus Overflow-bit Coding," *J. Infrared & Millimeter Waves* **11**(1), 59–64 (1992).

48. Mielikainen, J. and P. Toivanen, "Lossless compression of hyperspectral images using a quantized index to lookup tables," *IEEE Geosci. Remote Sens. Lett.* **5**(3), 474–477 (2008).
49. Mielikainen, J., P. Toivanen, and A. Kaarna, "Linear prediction in lossless compression of hyperspectral images," *Opt. Eng.* **42**(4), 1013–1017 (2003) [doi: 10.1117/1.1557174].
50. Aiazzi, B., S. Baronti, S. , and L. Alparone, "Lossless Compression of Hyperspectral Images Using Multiband Lookup Tables," *IEEE Signal Process. Lett.* **16**(6), 481–484 (2009).
51. Huang, B., "Fast Precomputed Vector Quantization with Optimal Bit Allocation for Lossless Compression of Ultraspectral Sounder Data," in *Satellite Data Compression,* B. Huang, Ed., 253–268, Springer, Berlin (2012).
52. Huang, B. et al., "Lossless compression of 3D hyperspectral sounding data using context-based adaptive lossless image codec with bias-adjusted reordering," *Opt. Eng.* **43**(9), 2071–2079 (2004) [doi: 10.1117/1.1778732].
53. Riskin, E. A., "Optimal bit allocation via the generalized BFOS algorithm," *IEEE Trans. Inform. Theory* **37**, 400–402 (1991).
54. Cuperman, V., "Joint bit allocation and dimensions optimization for vector transform quantization," *IEEE Trans. Inform. Theory* **39**, 302–305 (1993).
55. Aumann, H. H. and L. Strow, "AIRS, the first hyper-spectral infrared sounder for operational weather forecasting," *Proc. IEEE Aerospace Conf. 2001* **4**, 1683–1692 (2001).
56. Abrardo, A. et al., "Low-Complexity Approaches for Lossless and Near-Lossless Hyperspectral Image Compression," in *Satellite Data Compression* Huang, B., Ed., 47–65, Springer, Berlin (2012).
57. Hao, P. and Q. Shi, "Reversible integer KLT for progressive-to-lossless compression of multiple component images," *Proc. IEEE ICIP*, Barcelona, Spain, I-633–I-636 (2003).
58. Galli, L. and S. Salzo, "Lossless hyperspectral compression using KLT," *Proc. IEEE IGARSS 2004* **1**, 313–316 (2004).
59. Penna, B., T. Tillo, E. Magli, and G. Olmo, "Transform coding techniques for lossy hyperspectral data compression," *IEEE Trans. Geosci. Remote Sens.* **45**(5), 1408–1421 (2007).
60. Hao, P. and Q. Shi, "Matrix Factorizations for Reversible Integer Mapping," *IEEE Trans. Signal Process.* **42**(10), 2314–2324 (2001).
61. Wu, J., Z. Wu, and C. Wu, "Lossy to lossless compression of hyperspectral images using three-dimensional set partitioning algorithm," *Opt. Eng.* **45**(2), 027005 (2006) [doi: 10.1117/1.2173996].

62. International Standard Organization, *Information Technology - JPEG 2000 Image Coding System: Core Coding System*, 2nd Ed., Geneva, Switzerland (2004).
63. Tang, X. and W. A. Pearlman, "Three-dimensional wavelet-based compression of hyperspectral images," in *Hyperspectral Data Compression* Mottaet al.et al., Eds., 273–308, Kluwer Academic Publishers, Norwell, MA (2006).
64. Dragotti, P. L., G. Poggi, and A. R. P. Ragozini, "Compression of multispectral images by three-dimensional SPIHT algorithm," *IEEE Trans. Geosci. Remote Sens.* **38**(1), 416–428 (2000).
65. Hou, Y. and G. Liu, "Hyperspectral Image Lossy-to-Lossless Compression Using the 3D Embedded ZeroBlock Coding Algorithm," *Proc. IEEE Int. Workshop on Earth Observation & Remote Sens. Appl. 2008,* **2394** (2008).
66. Hsiang, S. T., "Highly scalable subband/wavelet image and video coding," Ph.D Dissertation, Rensselaer Polytechnic Institute, Troy, NY (2002).
67. Said, A. and W. A. Pearlman, "A new, fast, and efficient image codec based on set partitioning in hierarchal trees," *IEEE Trans. Circuits and Systems for Video Technology* **6**(3), 243–250 (1996).
68. Andrew, J., "A simple and efficient hierarchical image coder," *Proc. Int. Conf. Image Process. 1997* **3**, 658–661 (1997).
69. http://aviris.jpl.nasa.gov/html/data.html
70. Tang, X., S. Cho, and W. A. Pearlman. "3D set partitioning coding methods in hyperspectral image compression," *Proc. IEEE Int. Conf. Image Processing 2003* **2**, 239–242 (2003).

Chapter 3
International Standards for Spacecraft Data Compression

3.1 CCSDS and Three Data Compression Standards

Three international data-compression standards designed for data acquired by satellites or spacecraft have been developed by the Data Compression Working Group (DCWG) within the Consultative Committee for Space Data Systems (CCSDS).[1] (The author of this book is a member of the working group and participates in the development of the standards.) CCSDS is an international organization formed in 1982 by the major space agencies of the world for the development of communications and data systems standards for spaceflight. It is a subcommittee of the International Organization of Standards (ISO) and currently consists of 11 member agencies, 28 observer agencies, and over 100 industrial associates from 26 nations. The committee meets periodically to address spacecraft-data-system problems that are common to all participants and to formulate sound technical solutions to these problems. Since its establishment, it has developed recommended standards for space data and information systems to

1. Reduce the cost to the various agencies of performing common data functions by eliminating unjustified project-unique design and development, and
2. Promote interoperability and cross-support among cooperating space agencies to reduce operational costs by sharing facilities.

CCSDS has developed and published 53 recommended standards for space data and information systems (Blue Books), which have been issued as international standards by the ISO. The CCSDS-developed standards define specific interfaces, technical capabilities, or protocols, or provide prescriptive and/or normative definitions of interfaces, protocols, or other controlling standards such as encoding approaches. More than 600 space missions have used these standards.

Although JPEG and JPEG2000 exist as the international standards for image compression, they are not designed for compressing data and images acquired by satellites or spacecraft. The compression algorithms recommended by CCSDS are intended to be suitable for use aboard spacecraft. In particular, the complexity of the algorithms is designed to be sufficiently low to make high-speed hardware implementation feasible. The algorithms permit memory-efficient implementation that does not require large intermediate frames for buffering. In addition, the CCSDS standards are intended to limit the effects of data loss that can occur on the communications downlinking channel. Consequently, the compression algorithms are appropriate for frame-based image formats (two dimensions acquired simultaneously) produced, for example, by CCD arrays (called image frames) as well as strip-based input formats (i.e., images acquired one line at a time).

The three CCSDS data-compression standards include the following:

1. Lossless data compression: CCSDS 121.0-B[2] (ISO-15887:2000[3,4]) for lossless compression of 1D science data,
2. Image data compression: CCSDS 122.0-B[5] (ISO-26868:2009[6]) for lossy-to-lossless image compression,[7] and
3. Lossless multispectral and hyperspectral image compression: CCSDS 123.0-B[8] (ISO-18381:2013[9]) for lossless multispectral/hyperspectral image compression.

3.2 Lossless Data Compression Standard

The CCSDS Lossless Data Compression (LDC) standard (CCSDS-121) has lower computational complexity. It consists of two functional parts: the preprocessor and the adaptive entropy coder, as shown in Fig. 3.1.

3.2.1 Preprocessor

The preprocessor transforms the data into samples that can be more efficiently compressed by the adaptive entropy encoder. It transforms the original data in such a way that shorter codewords occur with higher probability than longer

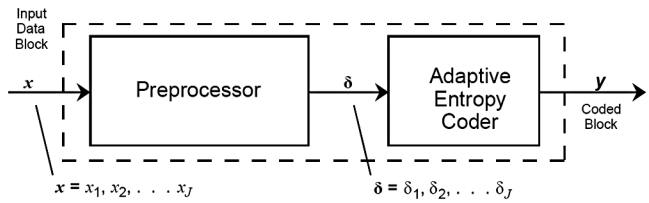


Figure 3.1 Flowchart of the CCSDS LDC coder (source: CCSDS).

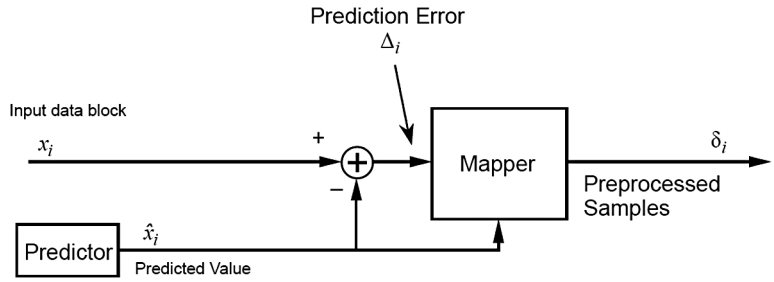


Figure 3.2 Preprocessor of the CCSDS-121 LDC coder (source: CCSDS).

codewords. To ensure that compression is lossless, the preprocessor must be reversible. A preprocessor that removes correlation between samples in the input data block will generally improve the performance of the entropy coder. In CCSDS-121, the preprocessing is done by a predictor followed by a prediction error mapper. For some types of data, more-sophisticated transform-based techniques can offer improved compression efficiency at the expense of higher complexity.

A preprocessor contains two functions, prediction and mapping, as shown in Fig. 3.2. The inputs to the compressor are $\mathbf{x} = x_1, x_2, \ldots, x_J$, which is a block of J-size samples with n-bit word-length.

In CCSDS-121, users are allowed to select any prediction scheme that best decorrelates the input datastream. Assuming an image with intensity values $x(i, j)$, where i represents the scan line, and j the pixel number within the scan, a possible predictor can be one of the following:

- **1D first-order predictor:**
  The predicted intensity value $\hat{x}_i$ might be equal to the previous samples on the same scan line $x(i, j \quad 1)$ or the neighboring sample value from a previous scan line $x(i \quad 1, j)$ The unit-delay predictor is an example of a 1D first-order predictor. Figure 3.3 shows the preprocessor with a unit-delay predictor.
- **2D second-order predictor:**
  The predicted intensity value $\hat{x}_i$ might be the average of the adjacent samples $x(i, j \quad 1)$ and $x(i \quad 1, j)$.

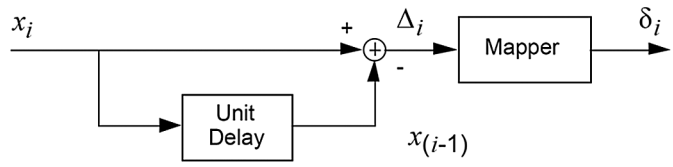


Figure 3.3 Preprocessor of the CCSDS-121 with a unit delay predictor (source: CCSDS).

- **2D third-order predictor:**
  The predicted intensity value $\hat{x}_i$ might be equal to a weighted combination of three neighboring values of $x(i, j \quad 1)$, $x(i \quad 1, j)$, and $x(i \quad 1, j \quad 1)$.

A reference sample is an unaltered data sample upon which successive predictions are based. Reference samples are required in the decoder side in order to recover the original values from difference values. The user must determine how often to insert references. When required, the reference must be the first sample of a block of J input data samples.

The preprocessor subtracts the predicted value $\hat{x}_i$ from the current data value x_i. The resultant $(n + 1)$-bit prediction error Δ_i is then mapped to an n-bit integer value δ_i based on the predicted value $\hat{x}_i$:

$$\delta_i = \begin{cases} 2\Delta_i & 0 \le \Delta_i \le \theta_i \\ 2|\Delta_i| \quad 1 & \theta_i \le \Delta_i < 0 \\ \theta_i + |\Delta_i| & \text{otherwise,} \end{cases} \tag{3.1}$$

where

$$\theta_i = \min(\hat{x}_i \quad x_{\min},\ x_{\max} \quad \hat{x}_i); \tag{3.2}$$

for a signed n-bit signal value,

$$x_{\min} = \quad 2^{n \ 1},\ x_{\max} = 2^{n \ 1} \quad 1; \tag{3.3}$$

and for a non-negative n-bit signal value,

$$x_{\min} = 0,\ x_{\max} = 2^n \quad 1. \tag{3.4}$$

When a predictor is properly chosen, the prediction error tends to be small. The unit-delay prediction technique is adopted in this standard.

3.2.2 Adaptive entropy encoder

The function of the adaptive entropy encoder (shown in Fig. 3.4) is to calculate variable-length codewords corresponding to each block of samples input from the preprocessor. It converts preprocessed samples δ into an encoded bit sequence y. The code selected is a variable-length code that utilizes Rice's algorithm.[10,11]

3.2.2.1 Variable-length coding

The Rice algorithm uses a set of variable-length codes to achieve compression. Each code is nearly optimal for a particular geometrically distributed source. Variable-length codes, such as Huffman codes and the codes used by the Rice algorithm, compress data by assigning shorter codewords to symbols that are expected to occur with higher frequency. By using several different codes and transmitting the code identifier, the Rice algorithm can adapt to many source datasets with an entropy value ranging

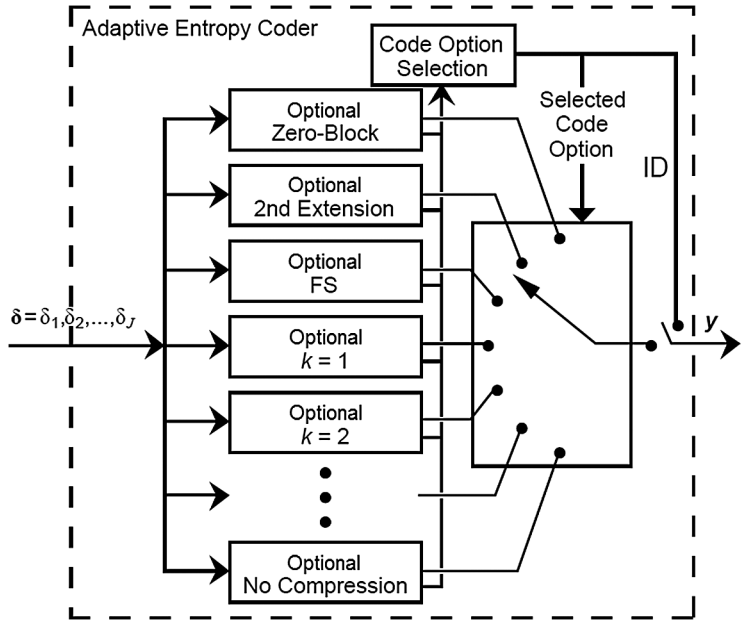


Figure 3.4 Block diagram of the adaptive entropy coder (source: CCSDS).

from low (more compressible) to high (less compressible). Because blocks of source samples are encoded independently, side information does not need to be carried across data packet boundaries, and the performance of the algorithm is independent of packet size.

The adaptive entropy coder consists of a collection of variable-length codes that can be applied to each block of J-size preprocessed samples. For each block, the coding option that achieves the best compression is selected to encode the block. The encoded block is preceded by an identifier (ID) bit pattern that identifies the coding option to the decoder. Because a new code option can be selected for each block, the Rice algorithm can adapt to changing source statistics. Thus, shorter values of the block-length parameter J allow faster adaptation to changing source statistics. However, the fraction of encoded bits used to indicate the coding option (the "overhead" associated with code option identification) decreases for larger values of J.

The first issue of the CCSDS-121 standard allowed J to take values $\{8, 16\}$. In practice, larger block sizes often offer improved overall compression effectiveness because of reduced overhead. Motivated by this fact—especially when the standard is used as the entropy coding option for multispectral or hyperspectral image compression as specified in Ref. [8]—issue 2 of the CCSDS-121 standard expands the allowed values of J to $\{8, 16, 32, 64\}$.

3.2.2.2 Coding options

In an adaptive entropy coder, several coding options are concurrently applied to a block of J-size samples. The option that yields the shortest encoded length for the current block of data is selected for transmission. An ID bit sequence is attached to the code block to indicate to the decoder which decoding option to use.

k split-sample options

As shown in Fig. 3.4, there are $k + 4$ code options in the adaptive entropy coder, where the k code options called "split-sample options" are the majority. The kth split-sample option splits off the k least-significant bits of each sample in a block of J-size samples and encodes the remaining higher-order bits with a simple fundamental sequence (FS) codeword before appending the split bits to the encoded FS datastream. Each split-sample option is designed to produce compressed data with a range of about 1 bit/sample (approximately $k+1.5$ to $k+2.5$ bits/sample); the code option yielding the fewest encoded bits is chosen for the block by the option-select logic. This option-selection process ensures that the block will be coded with the best available code option on the same block of data, but this does not necessarily imply that the source entropy lies in that range. The actual source entropy value could be lower; the source statistics and the effectiveness of the preprocessing stage determine how closely entropy can be approached.

Zeroblock option

The zeroblock option and second extension option are the two low-entropy code options. They are particularly efficient when the preprocessed samples are of very small values. The zeroblock option is a special case. This option is selected when one or more consecutive blocks of size-J samples are all zeros. In this case, the number of zeroblocks $n_{zero\ block}$ is encoded by an FS codeword whose bit length is equal to $n_{zero\ block}$ or $n_{zero\ block} + 1$ if $n_{zero\ block} > 4$.

Second-extension option

The second-extension option is designed to produce compressed datarates in a range of 0.5 bits/sample to 1.5 bits/sample. When this option is selected, the encoder first pairs consecutive blocks of J-size samples and then transforms the paired samples into a new value that is coded with an FS codeword. The FS codeword for γ is transmitted, where:

$$\gamma = \frac{(\delta_i + \delta_{i+1})(\delta_i + \delta_{i+1} + 1)}{2} + \delta_{i+1}. \tag{3.5}$$

Fundamental sequence code option

The fundamental sequence code is a variable-length code. In the adaptive entropy coder, a FS codeword is defined so that each '1' digit signals the end of a codeword, and the number of preceding zeros identifies which symbol was transmitted. This simple decoding procedure allows a FS codeword to be decoded without the use of lookup tables. For example, FS codewords '1,' '01,' '001,' and '0001' are used to code symbol s_1, s_2, s_3, and s_4.

The reason that the FS codes can achieve compression is that when symbol s_1 occurs very frequently, and when symbols s_3 and s_4 are very rare, on average fewer than two encoded bits per symbol will be transmitted, whereas an uncoded symbol always requires two bits per symbol. Longer FS codes achieve compression in a similar manner.

No-compression option

When none of the previous options provides any data compression on a block, the no-compression option is selected. Under this option, the preprocessed block of data is transmitted without alteration except for a prefixed identifier.

3.2.2.3 Coded dataset format

The coded dataset (CDS) format has the following structure:

- When a split-sample option is selected, the ID bit sequence is optionally followed by an n-bit reference sample, compressed data, and concatenated k least-significant bits from each sample.
- When the zeroblock option is selected, the CDS contains the option ID field, optionally followed by an n-bit reference sample and a required FS codeword specifying the number of concatenated zero-valued blocks or the remainder of the segment condition.
- When the second-extension option is selected, the CDS contains the option ID field, optionally followed by an n-bit reference sample and the required FS codewords for $J/2$ transformed samples.
- When the FS option is selected, the CDS contains the option ID field, optionally followed by an n-bit reference sample and required FS codewords.
- When the no-compression option is selected, the CDS is fixed-length and contains the option ID field, optionally followed by an n-bit reference sample and J preprocessed samples δ_i.

3.2.3 Performance evaluation

The CCSDS-121 LDC algorithm is designed to compress 1D data in lossless mode. It can also be used to compress any dimensional data. However, it does not benefit from the correlation contained in 2D or 3D data when it is applied to compress high-dimensional data.

Table 3.1 Summary of lossless compression results of the CCSDS-121 algorithm on imaging and nonimaging data.

Data Dimension	Platform	Instrument that Acquired the Dataset	Compression Ratio
1D	Hubble Space Telescope	Goddard High Resolution Spectrometer	1.72
1D	Submillimeter Wave Astronomy Satellite	Acousto Optical Spectrometer	2.3
1D	Mars Observer	Gamma Ray Spectrometer	$5 \sim 26$
2D	Landsat 4	Thematic Mapper	1.83
2D	Heat Capacity Mapping	Heat Capacity Mapping Radiometer	2.19
2D	Hubble Space Telescope	Wide Field Planetary Camera	2.97
2D	Solar A	Soft X Ray Solar Telescope	4.69
3D	Small Satellite Technology Infusion	Hyperspectral Imager	2.6

A compression study has been carried out to evaluate the performance of the standard.[12] Datasets from eight different scientific imaging and nonimaging instruments were tested. The following are the dimensions of the datasets and the instruments that acquire them:

- 1D: Goddard High-Resolution Spectrometer
- 1D: Acousto-Optical Spectrometer
- 1D: Gamma-Ray Spectrometer
- 2D: Landsat Thematic Mapper
- 2D: Heat-Capacity-Mapping Radiometer
- 2D: Wide-Field Planetary Camera
- 2D: Soft X-Ray Solar Telescope
- 3D: Hyperspectral Imager

The compression ratio (CR) is used to evaluate the performance; it is the ratio of the number of bits per sample before compression to the encoded datarate:

$$CR = \frac{nJ}{\text{average CDS length in bits}}, \tag{3.6}$$

where n is the number of bits per sample, and J is the block size. Table 3.1 summarizes the test datasets and the compression ratios.

3.2.3.1 1D data: Goddard High-Resolution Spectrometer

The Goddard High-Resolution Spectrometer (GHRS) is on the Hubble Space Telescope (HST). Its primary scientific objective is to investigate the interstellar medium, stellar winds, evolution, and extragalactic sources. It generates a 1D spectrum in the UV range with 512 data values and 16-bit dynamic range.

Two different prediction schemes are applied: the first uses the previous sample within the same spectrum, and the second uses the previous spectrum

in the same category. For two tested spectra, the achieved CR is 1.64 and 1.65 when the previous sample within the same spectrum is used as the prediction value. When spectrum 1 is used as prediction, the CR of spectrum 2 is 1.72.

3.2.3.2 1D data: Acousto-Optical Spectrometer

The Acousto-Optical Spectrometer (AOS) is on the Submillimeter-Wave Astronomy Satellite (SWAS), the objective of which is to study the energy balance and physical conditions of the molecular clouds in the galaxy by observing the radio-wave spectrum specific to certain molecules. The AOS utilizes a Bragg cell to convert the RF energy from the SWAS submillimeter-wave receiver into an acoustic wave, which then diffracts a laser beam onto a 1D CCD array. The sensor has 1,450 elements with a 16-bit readout. Because of limited available onboard memory, a CR greater than 2 is required for this mission.

Three prediction schemes are studied: the previous sample, the previous spectrum, and the higher-order multispectral. The results show that even with the similarity in trace spectra, a predictor using an adjacent trace spectrum in a direct manner does not improve the compression performance (CR = 1.61) because of the uneven background offset. The multispectral predictor mode is effective in dealing with spatially registered, multiple data sources with background offsets; it achieved a CR of 2.32.

3.2.3.3 1D data: Gamma-Ray Spectrometer

The purpose of the Gamma-Ray Spectrometer (GRS) on the Mars Observer was to collect data through several instruments to help scientists understand the Martian surface, atmospheric properties, and interactions between the various elements involved. The GRS uses a high-purity germanium detector for gamma rays. The flight spectrum is collected over 16,000 channels, each corresponding to a gamma-ray energy range. The total energy range of a spectrum extends from 0.2–10 MeV. These spectra show the random nature of the counts; some are actually zero over several bits. The spectral-count dynamic range is 8 bits.

Two different coding schemes are used: one-layer and two-layer. The one-layer scheme applies the LDC algorithm directly to the spectrum. The block size J is set to 16, and the entropy-coding mode is selected to bypass the predictor and the mapping function. With this scheme, a CR greater than 20 is achieved for the 5-second spectrum. As gamma-ray collection time increases, the achievable compression decreases.

In the two-layer scheme, two passes are used to compress the data. The first pass finds the channel numbers that have valid counts and generates the run length of channels between them. Meanwhile, a count file is created that holds only the valid counts as data in the file. In the second pass, both the channel run-length file and the count file are compressed using the LDC

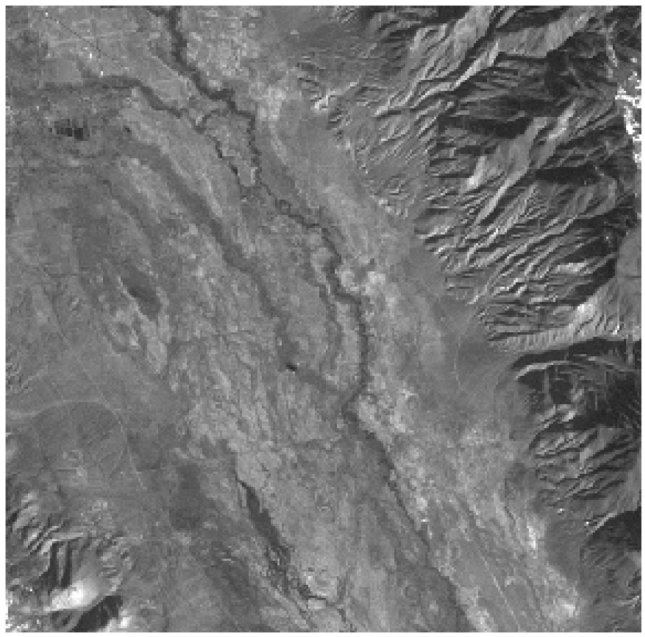


Figure 3.5 Landsat-4 TM image (band 1) over Sierra Nevada, California (source: NASA).

algorithm at a block size of 16 using the unit-delay prediction mode. The CRs of both schemes vary from 5 to 26.

3.2.3.4 2D image: Landsat Thematic Mapper

An image acquired by the Thematic Mapper (TM) aboard Landsat-4 at band 1 (0.45–0.52 μm) with a 30-m ground resolution has been tested. This 8-bit 512 × 512 image was taken over Sierra Nevada in California and has relatively high information over the mountainous area (shown in Fig. 3.5). In the test, a 1D unit-delay predictor in the horizontal direction was used. Block size was set to 16 samples, and one reference was inserted per image line. A CR of 1.83 is achieved for this 8-bit image.

3.2.3.5 2D image: Heat-Capacity-Mapping Radiometer

The Heat-Capacity-Mapping Radiometer (HCMR) is a solid-state photodetector sensitive in either the visible or infrared region. A typical image of 8 bits taken in the visible region with a ground resolution of 500 m over the Chesapeake Bay area was tested. With a 1D unit-delay predictor in the horizontal direction and a block size of 16 samples, the CR achieved is 2.19. When a 2D predictor takes the average of the previous sample and the sample on the previous scan line, and keeps other parameters the same, the CR is 2.28, or about a 5% increase over a 1D predictor.

3.2.3.6 2D image: Wide-Field Planetary Camera

The Wide-Field Planetary Camera (WFPC) is another payload aboard the HST. The camera uses four 2D CCD arrays. The test star-field image has the maximum value of a 12-bit resolution, which includes background noise of 9 bits. The test image has a minimum quantized value of 423. The size of the image is 800×800. The compression was performed using a block size of 16 and a unit-delay predictor in the horizontal line direction. The CR is 2.97.

3.2.3.7 2D image: Soft X-Ray Solar Telescope

The Soft X-Ray Telescope (SXT) is dedicated to the study of solar flares, especially of high-energy phenomena observed in the x-ray and gamma-ray ranges. It is a grazing-incidence reflecting telescope for the detection of x-rays in the wavelength range of 3–60 Å. It uses a 1024×1024 2D CCD detector array to cover the whole solar disk. A unit-delay predictor is applied to the high-contrast image. A CR of 4.69 is achieved, which means that only 3.2 bits are needed per pixel to provide the full precision of the 15-bit image.

3.2.3.8 3D image: hyperspectral imagery

The hyperspectral datacube tested was simulated from Airborne Visible Infrared Imaging Spectrometer (AVIRIS) data for the Small Satellite Technology Infusion (SSTI) mission. The objective of SSTI was to transition to new technologies for future missions to gain significant, measurable performance benefits.

The simulated datacube is used by two hyperspectral instruments to collect spectral–spatial information in the wavelength range from 0.4–2.5 μm: the visible–near-infrared (VNIR) imager and the short-wave-infrared (SWIR) imager. These imagers provide a 256-pixel spatial line with 128 spectral bands from the VNIR region and 256 spectral bands from the SWIR region. Both the VNIR and SWIR outputs are quantized to 12-bit data, and the GSD is 30 m. The SSTI also has a panchromatic imager that provides a 5-m ground resolution at 8-bit output. This CCD is a linear array of 2,592 elements. The study applied the LDC algorithm only to the hyperspectral VNIR and SWIR datacubes, which are sets of 256×256 spatial pixels, each with 384 spectral bands (see Fig. 3.6).

Compression was applied to either the VNIR or SWIR datacubes separately but in the same manner, that is, along the spatial direction in a spatial–spectral plane (shown in Fig. 3.6). An average compression ratio of 2.6 was obtained over both VNIR and SWIR data. If multispectral prediction is applied between the spectral lines, the CR is increased to 3.44.

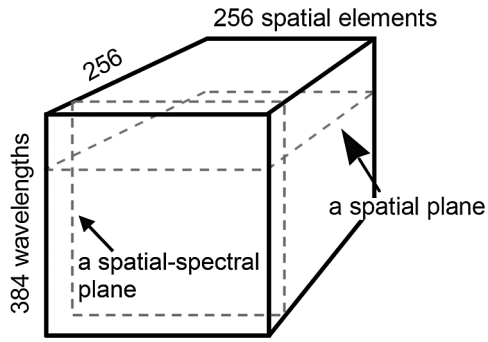


Figure 3.6 A hyperspectral datacube of size 256 lines × 256 pixels × 384 bands.

3.3 Image Data Compression Standard

3.3.1 Features of the standard

The CCSDS-122 Image Data Compression (IDC) standard defines a particular algorithm for compression of 2D images with 16-bit integers generated by many types of imaging instruments. The algorithm is intended to be suitable for use aboard spacecraft; in particular, the algorithm complexity is designed to be sufficiently low to make high-speed hardware implementation feasible. In addition, the algorithm permits a memory-efficient implementation that does not require large intermediate frames for buffering. Consequently, the ICD is appropriate for frame-based image formats (two dimensions acquired simultaneously) produced, for example, by CCD arrays as well as strip-based input formats (i.e., images acquired one line at a time) produced by push-broom-type sensors.

The IDC algorithm can provide both lossy and lossless compression. Under lossless compression, the original image data can be reproduced exactly, while under lossy compression, quantization and/or other approximations used in the compression process result in the inability to reproduce the original dataset. The lossless compression normally achieves a lower compression ratio compared to the lossy version for a given source image.

A discrete wavelet transform (DWT) is adopted in the standard to transform the original data to the wavelet domain to facilitate compression. The IDC algorithm supports two choices of DWT: an integer and a floating-point DWT. The integer DWT requires only integer arithmetic and thus is capable of providing lossless compression; it has lower implementation complexity. The floating-point DWT provides improved lossy compression effectiveness at low bitrates, but it requires floating-point calculations and cannot provide lossless compression.

In order to enhance resilience to bit errors of the compressed data and thus prevent data loss that may occur in the transmission downlink channel,

the wavelet-transformed subband data is partitioned into segments, each loosely corresponding to a different region of the image. Each segment is compressed independently so that the effects of data loss or corruption are limited to the affected segment. This wavelet-subband data partitioning also has the benefit of reducing the memory required for some implementations. The size of a segment can be adjusted to trade the degree of data protection versus compression effectiveness. Smaller segments provide increased protection against data loss, but they tend to reduce the overall compression ratio.

Each segment begins with a segment header that provides information about compression options and compressed data within the segment. The encoded segments do not include synchronization markers or other schemes intended to facilitate the automatic identification of segment boundaries. Users of the IDC algorithm must employ a suitable CCSDS packetization scheme[13] or other means of identifying segment boundaries.

Within each compressed segment, data is arranged so that earlier portions of the compressed data in the segment tend to make a larger contribution in the overall reconstructed fidelity than later portions. This embedded data structure allows a user to meet a constraint on the compressed-segment data volume by truncating the compressed bitstream for a segment at the appropriate point.

The trade-off between the reconstructed image quality and the compression ratio for each segment is controlled by specifying the maximum number of bytes in each segment along with a quality limit. The quality limit constrains the number of wavelet-transformed coefficients to be encoded in each segment. For each segment, compressed data is produced until the byte limit or quality limit is reached, whichever comes first. Lossless compression is achieved when the integer DWT is used and when the number of bytes required for losslessly encoding each segment does not exceed the segment byte limit.

The CCSDS IDC standard differs from JPEG2000[14] in several respects:

1. It specifically targets use aboard spacecraft;
2. A careful trade-off has been performed between compression performance and complexity;
3. Being less complex, it can be more-easily implemented in either hardware or software; and
4. It has a limited set of options, supporting its successful application without in-depth algorithm knowledge.

3.3.2 IDC compressor

The CCSDS IDC algorithm consists of two functional parts, depicted in Fig. 3.7: a DWT module that performs the DWT decomposition of image data and a bit-plane encoder (BPE) that encodes the transformed data. The standard

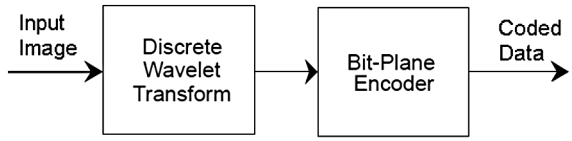


Figure 3.7 Block diagram of the CCSDS IDC standard.

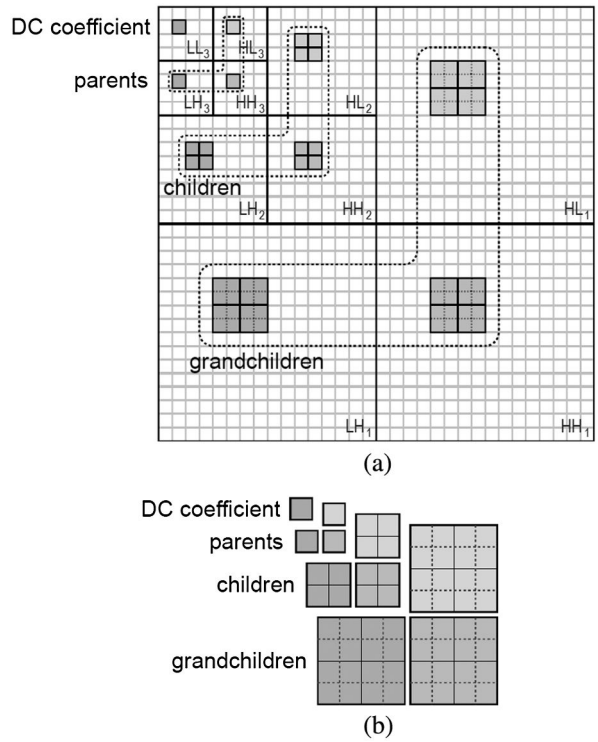


Figure 3.8 Three-level, 2D WT decomposition of an image and schematic of a transformed image with the 64 shaded pixels that consist of a single block: (a) the WT subband images and (b) a single block with 64 WT coefficients (source: CCSDS). For a color version of this figure, see Plate 1 in the color plate section of this book.

supports two choices of DWT, an integer and a floating-point DWT. The integer DWT requires only integer arithmetic and is capable of providing lossless compression.

The DWT performs three levels of 2D wavelet decomposition that produce 10 coefficient subband images, as illustrated in Fig. 3.8:

- LH_1, HH_1, and HL_1 after first-level WT decomposition on the entire input image;
- LH_2, HH_2, and HL_2 after second-level WT decomposition on the LL_1 subband image;

- LH_3, HH_3, and HL_3 after third-level WT decomposition on the LL_2 subband image; and
- LL_3, the third-level low-low-pass subband image (detailed subband image).

After the WT coefficients have been computed and placed in the buffer, the BPE stage begins encoding the WT coefficients. The BPE processes wavelet coefficients in groups of 64 coefficients, referred to as blocks. A block consists of a single coefficient from the lowest spatial frequency subband image, referred to as the DC coefficient (i.e., coefficients in subband image LL_3, as shown in Fig. 3.8), and 63 AC coefficients located in three-level subband images, as illustrated in Fig. 3.8. A block loosely corresponds to a localized region in the original image. Blocks are processed by the BPE in raster-scan order, i.e., rows of blocks are processed from top to bottom, proceeding from left to right horizontally within a row.

A segment is defined as a group of S consecutive blocks. Each segment loosely corresponds to a different region of the original image. Coding of DWT coefficients proceeds segment-by-segment, and each segment is coded independently of the others. The size of each segment (value of S) can change.

Partitioning DWT coefficients into segments has the advantage of limiting the effects of data loss that can occur on the communications channel. This is because each segment is compressed independently so that the effects of data loss or corruption, if they occur, are limited to the affected segment. Partitioning the DWT data into segments also has the benefit of limiting the memory required for some implementations. The segment size can be adjusted to trade the degree of data protection for compression effectiveness; smaller segments provide increased protection against data loss but tend to reduce the overall compression ratio.

A program and data flow diagram of the BPE stage of the compressor is shown in Fig. 3.9. The BPE takes DWT coefficient data from the DWT coefficient buffer, encodes the coefficient data, and places the encoded output in the compressed datastream. Coding of a segment happens in four steps:

1. Encode the segment header;
2. Encode quantized DC coefficients;
3. Encode the bit lengths of AC coefficient blocks; and
4. Encode bit planes of AC coefficients, including encoding parents, children, and grandchildren coefficients for all blocks within the segment.

Figure 3.9 shows the encoding flow of DWT coefficient data of a single segment, including complete encoding of all bit planes. In fact, encoding of a segment can terminate earlier: coding of a segment stops when the prescribed compressed-segment data-volume limit has been reached or when the prescribed segment-quality level has been reached, whichever comes first.

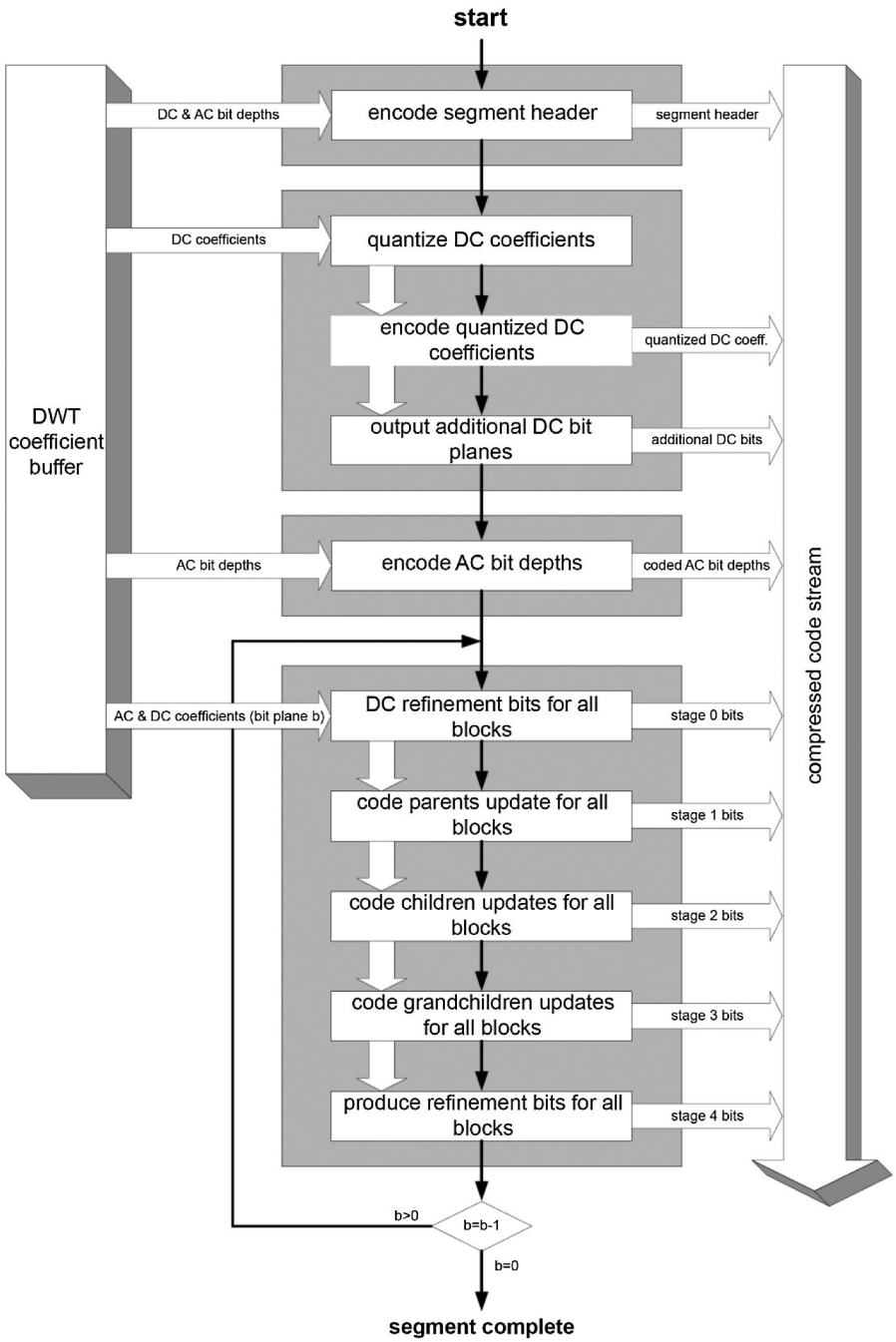


Figure 3.9 BPE encoding-block diagram and data flow (source: CCSDS).

The maximum number of bytes in each compressed segment is controlled via the SegByteLimit parameter, and image "quality" (more specifically, the number of DWT coefficients to be encoded) is constrained via the DCStop, BitPlaneStop, and StageStop parameters.[7] These parameters control the

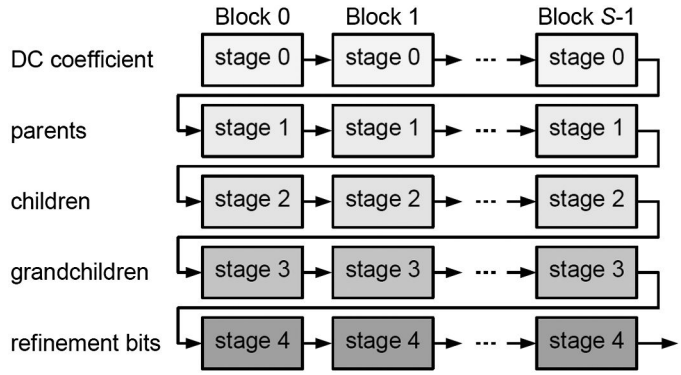


Figure 3.10 Data structure of an encoded bit plane of a segment (source: CCSDS).

trade-off between reconstructed segment quality and compressed data volume for each segment.

The encoded bitstream for a segment can be further truncated (or, equivalently, coding can be terminated early) at any point to further reduce the datarate in exchange for reduced image quality for the corresponding segment.

Figure 3.10 shows the data structure of the encoded bit plane of a segment. The entropy-coded data of a segment is arranged so that all parent coefficients in the segment are placed first, followed by children and then grandchildren coefficients, thereby supporting the desired embedded-data format. Finally, the segment includes (uncompressed) refinement bits for the AC coefficients in the segment for which more significant magnitude bits are not all zero.

There is no separate "lossless compression mode" in the CCSDS image-data compression standard. The lossless compression is achieved by using the integer DWT and setting some parameters to special values (i.e., set DCStop to 0, BitPlaneStop to 0, and StageStop to 3; and set the maximum number of bytes in each compressed-segment SegByteLimit sufficiently large to accommodate the compressed data volume needed to losslessly encode that segment).

3.3.3 Selection of compression options and parameters

While using the CCSDS-122 IDC, it is critical to select appropriate compression options and parameters. This has an effect on compression effectiveness and implementation complexity. For example, Fig. 3.11 illustrates that two different sets of compression parameters can yield remarkably different rate-distortion performance of the lossy compression on the same test image. In the figure, it can be seen that this method can achieve an MSE distortion value of approximately 10, and the bitrate

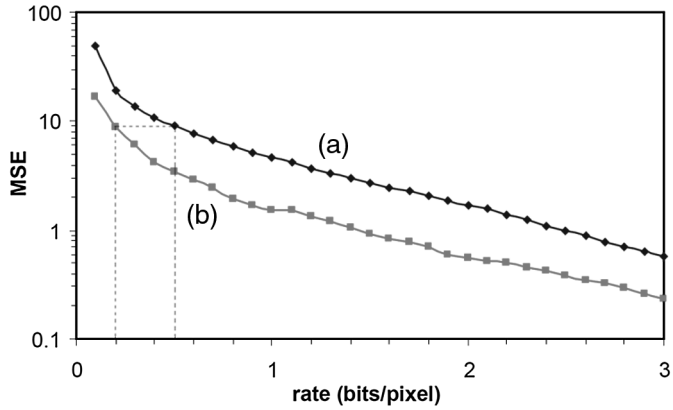


Figure 3.11 Rate-distortion performance of lossy compression on transposed version of a Coastal (band 7) test image using different compression parameters: (a) integer DWT, UseFill = 1 (fixed-rate compression), $S = 16$; and (b) float DWT, $S = 16386$ (full-frame compression) (source: CCSDS).

produced by the IDC with the parameter set (a) is 0.5 bits/pixel, whereas the bitrate produced by the IDC with the parameter set (b) is less than 0.2 bits/pixel. The former is more than a factor of two higher than the latter.

For a particular application, users are recommended to select appropriate compression options and parameters, including

- Optional segment headers,
- An integer or float DWT,
- A parameter that limits the compressed data volume,
- A parameter that controls reconstructed image fidelity,
- The number of blocks per segment,
- A Golomb code parameter, or
- A custom subband weight.

3.3.3.1 Segment headers

Each segment header may have up to four parts. The first part is mandatory, and the other parts are optional. Users need to select which of the optional parts should be included in the header. Bits used in parts 2 and 3 of the segment header can change from segment to segment, but in many applications one can expect them to be fixed for an image. Bits used in part 4 cannot change within an image. Thus, in a typical application, one might include all four header parts for the first segment of an image and only part 1 for subsequent segments. In some applications (e.g., if compression parameters are fixed for an entire mission), the header needs only part 1. Including optional header parts results in an increased bitrate, which may become particularly significant when segments are small and low bitrate compression is desired. For example, when $S = 16$,

including all optional header parts with each segment would contribute an additional 0.125 bits/pixel to the compressed bitrate.

3.3.3.2 Integer or float DWT

The use of an integer or floating DWT is indicated in the DWTtype field in the optional part 4 of the segment header. For lossless compression, the integer DWT must be used because the float DWT cannot provide lossless compression. The integer DWT requires only integer arithmetic and thus is fast, which may be preferable in some applications for complexity reasons. The float DWT may be preferable in some lossy image compression applications because it often provides better compression ratios than the integer DWT at low bitrates.

3.3.3.3 Parameters for controlling compression ratio and quality

To facilitate user control between bitrate or image distortion, the IDC algorithm provides means for specifying limits on compressed data volume and reconstructed image quality. For each segment, compressed data is produced until the segment's data volume limit or quality limit is reached, whichever comes first. These limits are specified using the four parameters SegByteLimit, DCStop, BitPlaneStop, and StageStop. The values of these parameters are encoded in the optional part 2 of the segment header.

The SegByteLimit parameter controls the compressed volume by directly limiting the number of compressed bytes (including headers) in a segment. The parameters DCStop, BitPlaneStop, and StageStop limit the number of DWT coefficients encoded in a compressed segment and thus indirectly control the reconstructed image quality.

A user can effectively eliminate the quality limit (by setting DCStop = 0, BitPlaneStop = 0, and StageStop = 3) so that the amount of compressed data produced for a segment is simply limited by the value of SegByteLimit. This scenario is referred to as rate-limited compression. The rate limit is assumed to be applied uniformly to every segment of the image. If, in addition, the UseFill parameter is set to 1 so that fill bits are added as needed to ensure that each compressed segment has exactly SegByteLimit bytes, then that compression is said to be fixed-rate.

At the other extreme, if a user specifies a sufficiently large value for SegByteLimit, then compression is limited by the quality limit determined by the values of the DCStop, BitPlaneStop, and StageStop parameters. The compression in this case is referred to as quality-limited.

3.3.3.4 Parameters for lossless compression

Lossless compression is achieved by (1) using the integer DWT; (2) setting DCStop to 0, BitPlaneStop to 0, and StageStop to 3; and (3) for each segment, setting the value of SegByteLimit sufficiently large to accommodate the

compressed data volume needed to losslessly encode that segment. There is no separate "lossless compression mode" in the IDC.

3.3.3.5 Segment size *S*

A user needs to select the number S of blocks per segment. The choice of S affects memory requirements, robustness to data errors or losses, and compression effectiveness because each segment is compressed independently.

An image with width w and height h generates $\lceil w/8 \rceil \times \lceil h/8 \rceil$ DWT blocks of DWT coefficients. These blocks can be viewed as an array with width $\lceil w/8 \rceil$ and height $\lceil h/8 \rceil$. When $S = \lceil w/8 \rceil \times \lceil h/8 \rceil$, the entire image is compressed as a single segment. This compression is referred to as *full-frame* compression. When $S = \lceil w/8 \rceil$, each image segment loosely corresponds to a thin horizontal strip of the image, and it is said that *strip* compression is being performed. Strip compression can lead to a relatively memory-efficient implementation and may be convenient, e.g., for images produced by push-broom-type sensors. In some circumstances, the compression ratio may be significantly improved when a larger value of S is used.

The minimum value of S is 16, except for the last segment of an image, which may consist of as little as a single block ($S = 1$). This limitation prevents users from using very small segments, which would tend to degrade compression effectiveness. The maximum value of S is constrained by the number of blocks produced for an image and the 20-bit field used to encode the value of S in the optional part 3 of the segment header. Thus, the maximum value of S is

$$S = \min\left\{ \left\lceil \frac{w}{8} \right\rceil \times \left\lceil \frac{h}{8} \right\rceil, 2^{20} \right\}. \tag{3.7}$$

A larger value of S generally leads to higher memory requirements because the BPE coding process requires the availability of a complete segment.

3.3.3.6 Golomb code parameter

As part of the segment encoding process, a simple differential coding procedure is used to losslessly encode the sequence of differences between quantized DC coefficients and the sequence of differences between quantized AC coefficients. The sequence of differences is partitioned into smaller sequences referred to as "gaggles." For each gaggle, one of the Golomb codes is used to encode the difference values. The selection of the Golomb code (i.e., the value of the parameter k) to encode each gaggle is normally done in the Rice coding by exhaustively trying each code option.

Alternatively, the CCSDS-122 standard allows the Golomb code for each gaggle to be selected by applying a computationally simpler but sometimes suboptimal heuristic-parameter-selection procedure.[15] In either

case, the k value selected is encoded in the compressed datastream, so the decompressor can decode the data without knowing which procedure was used. As reported in the reference, the difference in compression ratios between the optimal and heuristic parameter selection is negligible. Selecting whether to use the optimal or heuristic parameter selection is likely to be a matter of implementation complexity; the latter is computationally simple.

3.3.3.7 Custom subband weight

When the integer DWT is used, each DWT coefficient is multiplied by the same constant weight factor. The allowed weight factors are powers of two (e.g., 1, 2, 4, or 8) so that the multiplications used in the weighting process can be performed by bit-shift operations.

Subband weight factors may be assigned by using the default weight factors or custom weight factors. Subband weighting is not used in combination with the float DWT. The subband weight factors determine the relative order in which bitplanes from different subbands are encoded, which in turn affects compression effectiveness. The default set of weights were chosen to minimize the MSE image distortion obtained at a given rate. A custom set of weights might be appropriate, for example, to optimize a different image quality metric, and may be based on experiments with images from a particular instrument of interest.

The use of custom weights and the values of the weights are recorded in part 4 of the segment header. Note, however, that this part is optional even when custom weights are used. For example, if the same set of custom weights were used for an entire mission, one might elect not to encode the weight values in compressed images.

3.3.4 Performance evaluation

3.3.4.1 Lossless compression results

Table 3.2 lists the lossless compression results produced by the CCSDS IDC (CCSDS-122), CCSDS lossless data-compression (CCSDS-121, Rice algorithm), JPEG2000, JPEG-lossless (JPEG-LS), SPIHT (set partitioning in hierarchical trees), and ICER when applied to 30 images within the CCSDS test dataset. JPEG2000 results were produced using the verification model (VM) 9.0,[16] with either "frame-based" or "scan-based" compression options, a 5/3 integer filter DWT, and three-level wavelet decomposition. The SPIHT[17] algorithm is a low-complexity, progressive, zerotree wavelet image-compression algorithm. ICER[18] is also a progressive wavelet-based lossless-to-lossy image-compression algorithm. It offers wavelet-domain image segmentation for error-containment purposes. ICER produces slightly improved lossless and lossy CRs compared to the CCSDS-122 standard,

Table 3.2 Comparison of lossless compression results of CCSDS standards, JPEG2000, JPGE-LS, SPIHT, and ICER algorithms (source: CCSDS).

		Compressed bitrate (bits/pixel)							
		Strip-based compression				Frame-based compression			
Bit depth	Image	CCDS	CCSDS/Rice	JPEG-LS	JPEG 2000	CCSDS	JPEG 2000	SPIHT	ICER
8	coastal_b1	3 36	3 56	3 09	3 18	3 36	3 13	3 09	3 07
	coastal_b2	3 22	3 32	2 90	3 03	3 22	2 97	2 94	2 92
	coastal_b3	3 48	3 68	3 22	3 30	3 48	3 23	3 21	3 20
	coastal_b4	2 81	2 91	2 41	2 59	2 81	2 53	2 57	2 55
	coastal_b5	3 16	3 30	2 81	3 01	3 17	2 94	2 91	2 89
	coastal_b6h	3 02	2 75	2 50	2 68	3 02	2 60	2 71	2 54
	coastal_b6l	2 35	2 03	1 76	2 03	2 35	1 96	2 02	1 87
	coastal_b7	3 45	3 66	3 17	3 28	3 45	3 22	3 17	3 15
	coastal_b8	3 66	3 93	3 42	3 42	3 67	3 40	3 35	3 31
	europa3	6 61	7 48	6 64	6 56	6 60	6 52	6 46	6 30
	marstest	4 78	5 39	4 69	4 79	4 77	4 74	4 64	4 63
	lunar	4 58	5 23	4 35	4 56	4 58	4 49	4 43	4 40
	spot-la b3	4 80	5 20	4 53	4 74	4 79	4 69	4 70	4 56
	spot-la panchr	4 27	4 87	4 00	4 16	4 26	4 13	4 11	4 03
	8-bit-image averages	**3.82**	**4.09**	**3.54**	**3.67**	**3.82**	**3.61**	**3.59**	**3.53**
10	ice_2kb1	4 78	5 44	4 74	4 77	4 78	4 73	4 61	4 64
	ice_2kb4	3 37	3 86	3 23	3 28	3 37	3 25	3 17	3 18
	india_2kb1	4 77	5 25	4 63	4 76	4 77	4 72	4 63	4 63
	india_2kb4	4 06	4 70	3 97	4 05	4 07	4 01	3 93	3 94
	landesV_G7_10b	5 04	6 30	4 42	4 64	5 04	4 56	4 88	4 42
	marseille_G6_10b	6 74	7 56	6 57	6 77	6 72	6 72	6 60	6 49
	ocean_2kb1	4 94	5 32	4 61	4 91	4 94	4 88	4 79	4 75
	ocean_2kb4	3 81	4 41	3 60	3 76	3 81	3 73	3 67	3 64
	10-bit-image averages	**4.69**	**5.36**	**4.47**	**4.62**	**4.69**	**4.57**	**4.54**	**4.46**
12	foc	3 43	3 35	3 28	3 21	3 45	3 20	3 12	3 07
	pleiades_portdebouc_b3	7 87	8 63	7 79	8 01	7 87	7 97	7 71	7 73
	pleiades_portdebouc_pan	7 18	7 92	7 10	7 31	7 18	7 28	–	7 01
	solar	6 21	7 12	6 05	6 06	6 21	6 01	5 96	5 88
	sun_spot	5 79	6 63	5 67	5 71	5 78	5 66	5 66	5 49
	wfpc	3 82	4 04	3 61	3 49	3 84	3 47	3 43	3 32
	12-bit-image averages	**5.72**	**6.28**	**5.58**	**5.63**	**5.72**	**5.60**	**–**	**5.42**
16	P160_B_F	12 23	12 62	12 22	12 50	12 22	12 47	–	12 03
	sar16bit	9 92	10 31	9 90	10 07	9 92	10 02	–	9 68
	16-bit-image averages	**11.07**	**11.47**	**11.06**	**11.29**	**11.07**	**11.25**	**–**	**10.86**

but it has slightly higher complexity and has only been implemented as a frame-based compressor. It uses four levels of wavelet decomposition and a 2/6 integer filter DWT.

It can be seen from Table 3.2 that ICER produces the lowest average bitrate on the test images. However, JPEG-LS is equally effective on the 8-bit and 10-bit images and has significantly lower complexity than the wavelet-based compressors. Comparing the average compressed bitrate over all images at a given bit depth, the CCSDS-122 standard produces a higher bitrate than frame-based SPIHT, ICER, and JPEG2000 compressors on 8-bit, 10-bit, and 12-bit test images. For 16-bit test data, the CCSDS-122 performs better than JPEG2000. In both strip-based and frame-based options, the performances of the JPEG2000 and the CCSDS-122 standard are very close. The CCSDS-122 standard provides similar performances in both strip-based and frame-based compressions. The CCSDS/Rice (CCSDS-121) compressor exhibits the lowest performance, which is expected because only 1D correlation was explored in the compression.

3.3.4.2 Lossy compression results

To evaluate the lossy compression performance, the PSNR [Eq. (1.9)] and MAD [Eq. (1.10)] metrics are calculated at bitrates of 0.25, 0.5, 1.0, and 2.0 bits/pixel for each of the test images. Three compression algorithms are compared in the evaluation: CCSDS-122 (using a software implementation developed at NASA GSFC), the JPEG2000 (VM v9.0)[16] in frame-based and scan-based modes, and the SPIHT algorithm.[17]

For the CCSDS-122, the parameters used for lossy compression are as follows:

- Float DWT;
- SegByteLimit is adjusted to achieve rate-controlled compression at the desired bitrate;
- S is adjusted to achieve strip compression and also full-frame compression;
- All optional header parts are included only in the first coded segment, but no optional headers are included for subsequent coded segments; and
- CodeWordLength is set to 0, corresponding to single-byte output words.

Figures 3.12 and 3.13 show the average compression ratios versus PSNR of the test images with 12-bit and 16-bit dynamic range using CCSDS-122, JPEG, and SPIHT. For CCSDS-122 and JPEG2000, both strip and frame compression modes are performed.

For lossy compression, the compression results produced by the CCSDS-122 frame-based algorithm are better than those by its strip-based algorithm by roughly 1 dB. Overall, the trade-off between implementation complexity and performance is observed for either strip-based or frame-based results: the

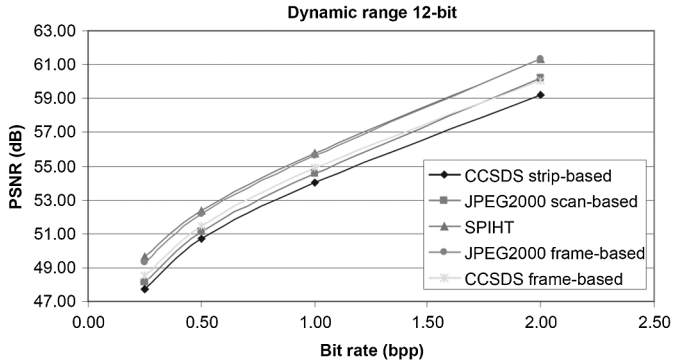


Figure 3.12 Mean PSNR of CCSDS-122, JPEG2000, and SPIHT on 12-bit test images (source: CCSDS).

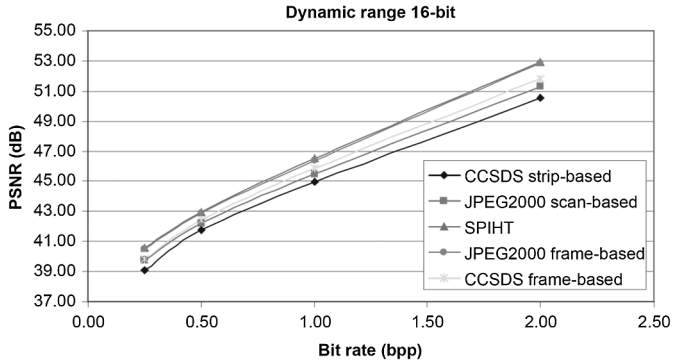


Figure 3.13 Mean PSNR of CCSDS-122, JPEG2000, and SPIHT on 16-bit test images (source: CCSDS).

CCSDS-122 algorithm is slightly lower than JPEG2000. However, this trade-off in performance is within 1 dB, and this difference may be smaller than the performance penalty obtained when similar complexity constraints are imposed on JPEG2000 in a practical hardware implementation.

3.4 Lossless Multispectral/Hyperspectral Compression Standard

3.4.1 Compressor composition

This CCSDS-recommended standard (CCSDS-123)[8] defines a lossless data compressor for 3D datacubes that are produced, e.g., by multispectral imagers, hyperspectral imagers, and sounders. The compressed datacube output from the compressor is an encoded bitstream from which the input

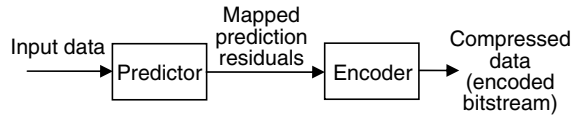


Figure 3.14 Two functional parts of the CCSDS-123 compression algorithm.

datacube can be recovered exactly. Because of variations in image content, the volume of compressed datacubes will vary from datacube to datacube. That is, the compression ratio is variable.

The compressor consists of two functional parts: a predictor and an encoder, as shown in Fig. 3.14. The predictor uses an adaptive linear prediction method to predict the value of each sample in a datacube based on the values of nearby samples in a small 3D neighborhood. Prediction is performed sequentially in a single pass. The prediction residual, i.e., the difference between the predicted value and the sample, is then mapped to an unsigned integer that can be represented using the same number of bits as the input data sample. These mapped prediction residuals are then encoded by an entropy encoder. Entropy-encoder parameters are adaptively adjusted during the encoding process to adapt to changes in the statistics of the mapped prediction residuals.

3.4.2 Adaptive linear predictor

The adaptive linear predictor is a key of the compressor. Assuming that an input 3D datacube with signed or unsigned integer sample values is denoted as $\{s_{z,y,x}\}$, where x and y are indices in the spatial dimensions, and the index z indicates the spectral band. Indices x, y, and z take on integer values in the ranges $0 \leq x \leq N_x \quad 1$, $0 \leq y \leq N_y \quad 1$, $0 \leq z \leq N_z \quad 1$, where each image dimension N_x, N_y, and N_z shall have a value of at least 1 and at most 2^{16}.

Prediction is performed causally in a single pass through the datacube. Prediction of the current sample $s_{z,y,x}$, that is, the calculation of $\hat{s}_{z,y,x}$ and mapped prediction residual $\delta_{z,y,x}$, generally depends on the values of nearby samples in the current spectral band and P preceding spectral bands, where P is a user-specified parameter. Figure 3.15 illustrates the typical neighborhood of samples used for prediction.

Within each spectral band, the predictor computes a local sum $\sigma_{z,y,x}$ of neighboring sample values. Each such local sum is used to compute a local difference. Predicted sample values are calculated using the local sum in the current spectral band and a weighted sum of local difference values from the current and previous spectral bands. The weights used in this calculation are dynamically updated following the calculation of each predicted sample value. Each prediction residual, that is, the difference between the current sample

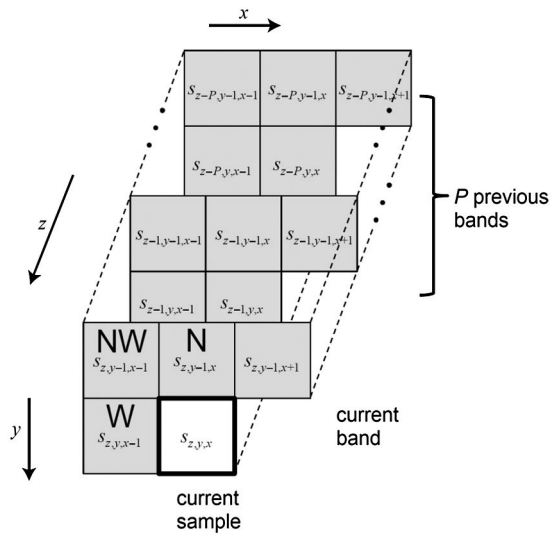


Figure 3.15 Typical prediction neighborhood (source: CCSDS).

value $s_{z,y,x}$ and the corresponding predicted sample value $\hat{s}_{z,y,x}$, is mapped to an unsigned integer $\delta_{z,y,x}$, the mapped prediction residual.

The local sum $\sigma_{z,y,x}$ is a weighted sum of samples in spectral band z that are adjacent to the current sample $s_{z,y,x}$. Figure 3.16 illustrates the samples used to calculate the local sum. This standard provides two options for a user to perform prediction using either neighbor-oriented or column-oriented local sums. When neighbor-oriented local sums are used, the local sum is equal to the sum of four neighboring sample values in the spectral band z. Equation (3.8) defines the neighbor-oriented local sum $\sigma_{z,y,x}$ and the boundary conditions when $y = 0$, $x = 0$, or $x = N_x - 1$:

$$\sigma_{z,y,x} = \begin{cases} s_{z,y,x\ \ 1} + s_{z,y\ \ 1,x\ \ 1} + s_{z,y\ \ 1,x} + s_{z,y\ \ 1,x+1} & y > 0, \quad 0 < x < N_x \quad 1 \\ 4s_{z,y,x\ \ 1} & y = 0, \quad x > 0 \\ 2(s_{z,y\ \ 1,x} + s_{z,y\ \ 1,x+1}) & y > 0, \quad x = 0 \\ s_{z,y,x\ \ 1} + s_{z,y\ \ 1,x\ \ 1} + 2s_{z,y\ \ 1,x} & y > 0, \quad x = N_x \quad 1. \end{cases} \tag{3.8}$$

When column-oriented local sums are used, the local sum is equal to four times the neighboring sample value in the previous row. Equation (3.9) defines the column-oriented local sum $\sigma_{z,y,x}$ and the boundary condition when $y = 0$:

$$\sigma_{z,y,x} = \begin{cases} 4s_{z,y\ \ 1,x} & y > 0 \\ 4s_{z,y,x\ \ 1} & y = 0, \ x > 0. \end{cases} \tag{3.9}$$

The local sums are used to calculate local difference values. In each spectral band there are four local difference values: $d_{z,y,x}$, $d^N_{z,y,x}$, $d^W_{z,y,x}$, and

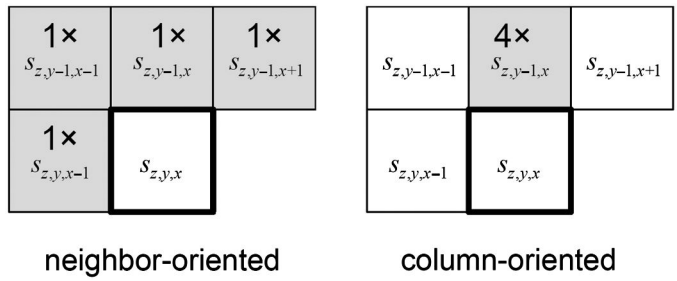


Figure 3.16 Two options of calculating local sum $\sigma_{z,y,x}$: neighbor-oriented and column-oriented. (source: CCSDS)

$d^{NW}_{z,y,x}$. The central local difference $d_{z,y,x}$ is equal to the difference between the local sum $\sigma_{z,y,x}$ and four times the sample value $s_{z,y,x}$:

$$d_{z,y,x} = 4\ s_{z,y,x} \qquad \sigma_{z,y,x}. \tag{3.10}$$

The three directional local differences, $d^{N}_{z,y,x}$, $d^{W}_{z,y,x}$, and $d^{NW}_{z,y,x}$, are each equal to the difference between $d_{z,y,x}$ and four times a sample value labeled as 'N,' 'W,' or 'NW,' as shown in Fig. 3.15:

$$d^{N}_{z,y,x} = \begin{cases} 4s_{z,y\ \ 1,x} & \sigma_{z,y,x} \quad y > 0 \\ 0 & \qquad\quad y = 0; \end{cases} \tag{3.11}$$

$$d^{W}_{z,y,x} = \begin{cases} 4s_{z,y,x\ \ 1} & \sigma_{z,y,x} \quad y > 0,\ x > 0 \\ 4s_{z,y\ \ 1,x} & \sigma_{z,y,x} \quad y > 0,\ x = 0 \\ 0 & \qquad\quad y = 0; \end{cases} \tag{3.12}$$

$$d^{NW}_{z,y,x} = \begin{cases} 4s_{z,y\ \ 1.x\ \ 1} & \sigma_{z,y,x} \quad y > 0,\ x > 0 \\ 4s_{z,y\ \ 1,x} & \sigma_{z,y,x} \quad y > 0,\ x = 0 \\ 0 & \qquad\quad y = 0. \end{cases} \tag{3.13}$$

During compression of a datacube, a user may choose to perform prediction in either full or reduced mode. Under the full prediction mode, prediction in spectral band z uses both a weighted sum of the central local differences $d_{z,y,x}$ computed in preceding bands and the three directional local differences $d^{N}_{z,y,x}$, $d^{W}_{z,y,x}$, and $d^{NW}_{z,y,x}$ calculated in the current band. Under reduced mode, prediction uses a weighted sum of the central local differences $d_{z,y,x}$ computed in preceding bands; the directional local differences are not used.

As described in the CCSDS 120.2-G-0 green book,[19] the use of reduced prediction mode in combination with column-oriented local sums tends to yield higher compression ratios for uncalibrated, raw datacubes acquired by

push-broom imagers that exhibit significant along-track streaking artifacts. The use of full prediction mode in combination with neighbor-oriented local sums tends to yield higher compression ratios for whisk-broom imagers, frame imagers, and calibrated imagery.

After the prediction residual, i.e., the difference between the predicted value and the sample, is obtained, it is then mapped to an unsigned integer that can be represented using the same number of bits as the input data sample. These mapped prediction residuals are then encoded by an entropy encoder.

The mapped prediction residual $\delta_{z,y,x}$ is an integer defined as

$$\delta_{z,y,x} = \begin{cases} |\Delta_{z,y,x}| + \theta_{z,y,x}, & |\Delta_{z,y,x}| > \theta_{z,y,x} \\ 2|\Delta_{z,y,x}|, & 0 \leq (\quad 1)\hat{s}_{z,y,x}\ \Delta_{z,y,x} \leq \theta_{z,y,x} \\ 2|\Delta_{z,y,x}| \quad 1, & \text{otherwise,} \end{cases} \tag{3.14}$$

where the prediction residual $\Delta_{z,y,x}$ is the difference between the predicted value and actual sample value:

$$\Delta_{z,y,x} = s_{z,y,x} \quad \hat{s}_{z,y,x}; \tag{3.15}$$

and $\theta_{z,y,x}$ is defined as

$$\theta_{z,y,x} = \min\{\hat{s}_{z,y,x} \quad s_{\min}, s_{\max} \quad \hat{s}_{z,y,x}\}. \tag{3.16}$$

3.4.3 Encoder

The mapped prediction residuals are sequentially encoded in an order selected by the user. This encoding order need not correspond to the order in which samples are output from the imaging instrument or processed by the predictor. To encode the mapped prediction residuals for a datacube, a user may choose to use the sample-adaptive entropy coding approach or the block-adaptive approach. The latter approach relies on the lossless data compressor defined in the CCSDS-121 standard.[2] The sample-adaptive entropy coder typically yields smaller compressed images than the block-adaptive entropy coder. Further examples and comparisons can be found elsewhere.[19]

Under the sample-adaptive entropy coding approach, each mapped prediction residual is encoded using a variable-length binary codeword.[20] The variable-length codes used are adaptively selected based on statistics that are updated after each sample is encoded. Separate statistics are maintained for each spectral band, and the compressed data size does not depend on the order in which mapped prediction residuals are encoded.

Under the block-adaptive entropy-coding approach, the sequence of mapped prediction residuals is partitioned into short blocks, and the encoding method used is independently and adaptively selected for each block. Depending on the encoding order, the mapped prediction residuals in a block may be from the same

or different spectral bands, and thus the compressed image size depends on the encoding order when this method is used.

The compressed bitstream of a datacube consists of a header followed by a body. The variable-length header records compression parameters. The body consists of losslessly encoded mapped prediction residuals $\delta_{z,y,x}$ from the predictor.

3.4.4 Performance evaluation

Experimental results of lossless compression of hyperspectral images using the CCSDS-123 standard have been reported[21,22] and compared with the LUT method,[23] JPEG-LS,[24] and differential JPEG-LS, where JPEG-LS is applied to the differences between two adjacent spectral-band images. Table 3.3 compares the compression results of raw hyperspectral images (i.e., uncalibrated). Twenty-one hyperspectral images were tested. The AIRS images were hyperspectral sounder images with 1,501 bands. Yellowstone images were acquired using the AVIRIS sensor. The Hyperion images were spaceborne hyperspectral images.

Table 3.3 Comparison of lossless compression results (bits per pixel per band) of hyperspectral images using the CCSDS-123 standard, JPGE-LS, JPGE-LS-Diff, and LUT algorithms.

Image	CCSDS-123	LUT	JPEG-LS	JPEG-LS-Diff
AIRS Granule 9	4.24	5.47	6.87	5.19
AIRS Granule 16	4.22	5.40	6.71	5.06
AIRS Granule 60	4.37	5.84	7.33	5.39
AIRS Granule 82	4.17	5.16	6.39	4.94
AIRS Granule 120	4.30	5.60	6.79	5.20
AIRS Granule 126	4.40	5.81	7.19	5.41
AIRS Granule 129	4.17	5.32	6.08	4.90
AIRS Granule 151	4.42	5.94	6.95	5.37
AIRS Granule 182	4.42	6.15	7.02	5.40
AIRS Granule 193	4.41	5.84	7.11	5.39
CASI t0180f07 raw	4.78	5.51	5.23	4.93
CASI t0477f06 raw	4.97	5.81	5.44	5.20
SFSI Mantar Raw	4.76	5.23	4.89	5.12
Yellowstone Sc00	6.41	7.16	9.18	6.98
Yellowstone Sc03	6.27	6.93	8.87	6.86
Yellowstone Sc10	5.67	6.28	7.32	6.19
Yellowstone Sc11	5.97	6.72	8.50	6.51
Yellowstone Sc18	6.52	7.24	9.30	6.96
Maine	2.78	3.45	4.53	3.39
Hawaii 614	2.71	3.26	4.61	3.30
Hyperion GeoSample	4.64	5.82	5.03	4.57
Hyperion GeoSample (Flat Fielded)	4.09	4.46	4.83	4.36
Average	**4.67**	**5.65**	**6.64**	**5.30**

It can be seen from the table that the CCSDS-123 standard produces the shortest bitrates. The average bitrate is 4.67 bits per pixel per band (bpppb), and it yields the best coding performance compared to the LUT method, JPEG-LS, and JPEG-LS Diff. There is a better-than 1.0-bpppb gain of the JPEG-LS Diff over the JPEG-LS. The performance of the LUT algorithm is closer to the JPEG-LS Diff, although it is still 1.0 bpppb worse than the CCSDS-123 standard.

References

1. http://public.ccsds.org/default.aspx
2. "Recommendation for Space Data System Standards," *Lossless Data Compression*, CCSDS 121.0-B-2, Blue Book, Issue 2, available at http://public.ccsds.org/publications/archive/121x0b2.pdf, CCSDS, Washington, D.C. (May 2012).
3. "Space data and information transfer systems - Data systems - Lossless data compression," ISO 15887, available at http://www.iso.org/iso/iso catalogue/catalogue tc/catalogue detail.htm?csnumber=29440 (2000).
4. "Space data and information transfer systems - Data systems - Lossless data compression," ISO 15887 (2000) / Cor 1 (2009), available at http://www.iso.org/iso/iso catalogue/catalogue tc/catalogue detail.htm?csnumber=52985.
5. "Recommendation for Space Data System Standards," *Image Data Compression*, CCSDS 122.0-B-1, Blue Book, Issue 1, available at http://public.ccsds.org/publications/archive/122x0b1c3.pdf, CCSDS, Washington, D.C. (Nov. 2005).
6. "Space data and information transfer systems - Data systems - Lossless Image Compression," ISO 26868, available at http://www.iso.org/iso/iso catalogue/catalogue tc/catalogue detail.htm?csnumber=43849 (2009).
7. "Report Concerning Space Data System Standards," *Image Data Compression*, CCSDS 120.1-G-1, Green Book, Issue 2, available at http://public.ccsds.org/publications/GreenBooks.aspx, CCSDS, Washington, D.C. (June 2007).
8. "Recommendation for Space Data System Standards," *Lossless Multispectral & Hyperspectral Image Compression*, CCSDS 123.0-B-1, Blue Book, Issue 1, available at http://public.ccsds.org/publications/archive/123x0b1ec1.pdf, CCSDS, Washington, D.C. (May 2012).
9. Space data and information transfer systems - Lossless multispectral and hyperspectral image compression, ISO 18381:2013 (available at http://www.iso.org/iso/home/store/catalogue tc/catalogue detail.htm?csnumber=62319).
10. Rice, R. F. and R. Plaunt, "Adaptive Variable-Length Coding for Efficient Compression of Spacecraft Television Data," *IEEE Trans. on Comm.*, **16**(9), 889–897 (1971).

11. Rice, R. F. (1979), "Some Practical Universal Noiseless Coding Techniques," JPL Publication 79–22, Mar. 1979.
12. "Report Concerning Space Data System Standards," *Lossless Data Compression*, CCSDS 120.0-G-3, Green Book, Issue 3, available at http://public.ccsds.org/publications/GreenBooks.aspx, CCSDS, Washington, D.C. (April 2013).
13. "Recommendation for Space Data System Standards," *Space Packet Protocol*, CCSDS 133.0-B-1, Blue Book, Issue 1, available at http://public.ccsds.org/publications/archive/133x0b1.pdf, CCSDS, Washington, D.C. (September 2003).
14. International Standard Organization, *Information Technology - JPEG 2000 Image Coding System: Core Coding System*, 2nd Ed., Geneva, Switzerland (2004).
15. Kiely, A., "Selecting the Golomb Parameter in Rice Coding." The Interplanetary Network Progress Report 42, no. 159 (2004).
16. ISO/IEC JTC1/SC29/WG1 N2021, Information JPEG2000 Verification Model 9.0
17. Said, A. and W. A. Pearlman, "A new, fast, and efficient image codec based on set partitioning in hierarchal trees," *IEEE Trans. Circuits and Systems for Video Technology* **6**(3), 243–250 (1996).
18. Kiely, A. and M. Klimesh, "The ICER Progressive Wavelet Image Compressor," *The Interplanetary Network Progress Report* **42**(155), 1–46 (2003).
19. "Report Concerning Space Data System Standards," *Lossless Multispectral & Hyperspectral Image Compression*, CCSDS 120.2-G-0, Green Book, Issue 1, CCSDS, Washington, D.C. (2013).
20. Kiely, A., "Simpler Adaptive Selection of Golomb Power-of-Two Codes," *NASA Tech Briefs* - November 2007, 28–29 (2007).
21. Sánchez, J. E., E. Auge, J. Santaló, I. Blanes, J. Serra-Sagristà, and A. Kiely, "Review and implementation of the emerging CCSDS Recommended Standard for multispectral and hyperspectral lossless image coding," IEEE Int. Conf. on Data Compression, Comm., and Process. (2011).
22. Sanchez, J. E., E. Auge, A. Kiely, I. Blanes, and J. Serra-Sagristà, "Performance impact of parameter tuning on the CCSDS-123 lossless multi- and hyperspectral image compression standard," Proc. Onboard Payload Data Compression (2012).
23. Mielikainen, J., "Lossless compression of hyperspectral images using lookup tables," *IEEE Signal Processing Letters* **13**(3), 157–160 (2006).
24. "JPEG-LS, Information Technology," 14495-1, ISO/IEC; ITU-T, Recommandation T.87 (1999).

Chapter 4
Vector Quantization Data Compression

4.1 Concept of Vector Quantization Compression

Vector quantization (VQ) is an efficient coding technique to quantize signal vectors. It has been widely used in signal and image processing, such as pattern recognition and speech and image coding. A VQ compression procedure has two main steps: codebook training (sometimes also referred to as codebook generation) and coding (i.e., codevector matching). In the training step, similar vectors in a training sequence are grouped into clusters, and each cluster is assigned to a single representative vector called a codevector. In the coding step, each input vector is then compressed by replacing it with the nearest codevector referenced by a simple cluster index. The index (or address) of the matched codevector in the codebook is then transmitted to the decoder over a channel and is used by the decoder to retrieve the same codevector from an identical codebook. This is the reconstructed reproduction of the corresponding input vector. Compression is thus obtained by transmitting the index of the codevector rather than the entire codevector itself.[1]

Unlike scalar quantization, vector quantization requires segmentation of the source data into vectors. In 2D image data compression with VQ, a block with $n \times m$ (n may be equal to m) pixels is usually taken as a vector whose dimension is equal to $n \times m$. Vectors constituted in this way have no physical meaning. Because the blocks are segmented according to the row and column indices of an image, the vectors obtained in this manner change at random as the pixel patterns change from block to block. The reconstructed image shows an explicit blocking effect for large compression ratios.

There are several approaches to constituting vectors for a 3D datacube of hyperspectral imagery. The simplest approach is to treat the 3D datacube as a set of monochromatic images and then segment each monochromatic image into vectors independently, as in the 2D image

case. This approach, however, does not make use of the high correlation of data in the spectral domain. This book's approach to constituting a vector for VQ compression is to define one spectral profile corresponding to a footprint on the ground as a vector. There are a total of $N_r \times N_c$ vectors for a 3D datacube with spatial dimensions N_r rows by N_c columns, and the dimension of each vector is equal to the number of spectral bands N_b. The vectors constituted in this way have a physical meaning: each vector is a spectrum, and each spectrum is an indicator of the material on the Earth's surface that lies within the field of view of the hyperspectral sensor. Because the number of materials in the scene is usually limited, the number of different encoded spectra can be much smaller than the total number of vectors $N_r \times N_c$. Thus, all of the spectral vectors can be expressed using a codebook with comparatively few codevectors and achieve good reconstruction fidelity. This constitution of vectors makes good use of the high correlation often found between bands in the spectral domain and achieves a high compression ratio. The following sections show that it also leads to fast codebook generation and fast codevector matching.

Figure 4.1 illustrates the concept of the VQ compression of a hyperspectral datacube using the vectors formed as defined earlier. In the figure, a hyperspectral datacube with N_b spectral band images, whose spatial size is N_r rows by N_c column pixels, is used as an input datacube. A spectral profile (i.e., N_b elements in the spectral direction) corresponding to a footprint on the ground is taken as a vector. A codebook containing N codevectors has been generated and stored in the training step. In the coding step, in order to compress the datacube, each of the vectors of the datacube is compared to the N codevectors in the codebook to find the best matching one. The index of the best matched codevector is the output of the coding result of the input vector; it is assigned to the element of the index map at the same spatial location as the ground sample cell at the datacube corresponding to the vector. After all of the vectors of the datacube are coded, an index map with all the indices filled is formed. The index map is the compression result of the datacube. The number of bits needed to express an index is $\log_2(N)$ for a codebook with N codevectors.

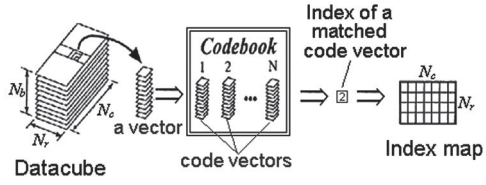


Figure 4.1 Illustration of the concept of a VQ compression algorithm.

Thus the total number of bits needed to encode the datacube is $N_r N_c \log_2 N$. The compression ratio can be computed as

$$C_r = \frac{N_r N_c N_b L}{N_r N_c \log_2 N} = \frac{N_b L}{\log_2 N}, \tag{4.1}$$

where L is the word-length of the original datacube, i.e., the number of bits used to express the value of the original datacube. Equation (4.1) indicates that the compression ratio depends on the spectral band number N_b, the word-length of the original datacube L, and the codebook size N. Codebook size is the only parameter that controls the compression ratio, as N_b and L are fixed for a datacube to be compressed. The compression ratio is independent from the size of the datacube. If the size of a hyperspectral datacube increases while the size of the codebook N remains the same, then the compression ratio will remain the same but the compression fidelity will decrease. Assuming a hyperspectral 3D datacube acquired using AVIRIS, $L = 16$ bits, $N_b = 224$, a codebook with $N = 4096$ codevectors is used, and the compression ratio is $C_r = 298.7$.

Many existing VQ algorithms for codebook designs are available, such as the LBG algorithm,[2] the tree-structure codebook algorithm,[1] and the self-organizing feature map (SOFM).[3] Among these, the LBG algorithm is the most widely used because of its fidelity. [This algorithm assumes a training sequence $\mathbf{X}_j$ $(j = 1, 2, \ldots, n)$, with n vectors whose dimension is k, is used to generate a codebook with N codevectors, giving an initial codebook with N codevectors $\hat{\mathbf{A}}_m = \{\mathbf{Y}_i; i = 1, 2, \ldots, N\}$.] The algorithm then finds the minimum distance partition (MDP) $P(\hat{\mathbf{A}}_m) = \{\mathbf{S}_i; i = 1, 2, \ldots, N\}$ for each of vectors in the training sequence: $\mathbf{X}_j \in \mathbf{S}_i$ if $d(\mathbf{X}_j, \mathbf{Y}_i) \leq d(\mathbf{X}_j, \mathbf{Y}_l)$ for all $l = 1, 2, \ldots, N$. Codebook training is an iterative process: The algorithm updates the codebook $\hat{\mathbf{A}}_m$ at the end of an iteration loop, where m defines the number of iterations to adjust the codebook. The most-common distortion measure is the Euclidean distance, which is a squared-error distortion measure:

$$d(\mathbf{X}, \mathbf{Y}) = \mathbf{X} \quad \mathbf{Y}^2 = \sum_{i=1}^{N_b} (x_i \quad y_i)^2. \tag{4.2}$$

If this measure is used, the algorithm needs to calculate $d(\mathbf{X}, \mathbf{Y})$ N times for each training vector $\mathbf{X}_j$ $(j = 1, 2, \ldots, n)$ to find the MDP (or nearest partition) S_i. To compute one $d(\mathbf{X}, \mathbf{Y})$ requires k product and $k - 1$ addition operations, where k is the dimension of the vectors. Thus, it takes $N \times k$ products and $N \times (k - 1)$ additions to find the MDP S_i for one vector. For a training sequence with a total of n vectors, it takes $n \times N \times k$ products and $n \times N \times (k - 1)$ additions to find the MDP $P(\hat{\mathbf{A}}_m) = \{\mathbf{S}_i; i = 1, 2, \ldots, N\}$ to go through more than one iteration of codebook training. The number of iteration loops is controlled by a distortion threshold ε

(≥ 0). It is common for over 10 iterations of training to be required to generate a codebook.

The following is an example of training a codebook for AVIRIS datacubes. The size of an AVIRIS datacube is 224 spectral band images, each of which has 512 lines, and each line has 614 pixels. If the vectors are constituted by defining an entire spectrum of each ground sample of the datacube as a vector, an AVIRIS datacube contains $512 \times 614 = 314{,}368$ vectors of dimension $k = 224$. Supposing that two AVIRIS datacubes of this size are used as a training set, the total number of training vectors is $n = 628{,}736$. Generating a codebook with $N = 4096$ codevectors requires $628{,}736 \times 4096 \times 224 = 57.7 \times 10^{10}$ products per iteration. Supposing that one product operation takes roughly 10 ns, one iteration of training can take 1.6 h. The disadvantages of this algorithm are its computational complexity and the time needed to form the codebook. Many constrained VQ techniques[4] have been developed to reduce the encoding complexity at the cost of a slight performance penalty. As well, many fast search methods have been developed to reduce the search complexity of the unconstrained VQ techniques.

The main focus of this chapter is (1) the fast VQ algorithms that overcome the computational complexity of the conventional VQ algorithms, and (2) how the VQ technique can be applied to compress 3D datacubes of hyperspectral imagery in the near-lossless sense. Section 4.2 briefly reviews fast VQ algorithms. Sections 4.3 and 4.4 describe two fast VQ search algorithms that reduce the full-search of the classical VQ algorithm. Section 4.5 discusses a VQ compression algorithm for compressing hyperspectral data using the spectral-feature-based binary code (SFBBC), which permits the use of the Hamming distance rather than the Euclidean distance in codebook training and codevector matching (and significantly quickens processing). Section 4.6 describes an algorithm called correlation vector quantization (CVQ), which exploits both spectral and spatial correlation simultaneously of hyperspectral datacubes. Section 4.7 presents an effective VQ algorithm with multiple subcodebooks that uses remote sensing information of the datacube to improve the VQ compression performance and speed up the processing. The final two sections discuss two near-lossless VQ compression techniques for hyperspectral datacubes.

4.2 Review of Conventional Fast Vector Quantization Algorithms

The classical generalized Lloyd algorithm (GLA) is the most cited and widely used full-search VQ method due to its simplicity and relatively good fidelity. However, it suffers from a serious complexity barrier that greatly limits its practical use. The research in developing fast search for VQ compression algorithms has been very active. A general approach to reducing the computational complexity of a full search in the GLA is based on identifying the geometric boundaries of the Voronoi cells and storing a suitable data

structure to ease the search process. In this way much of the search complexity is transferred to the offline design of the data structure.[5]

One approach is based on the use of *k*-dimensional (k-d) trees.[6 8] The search process is simplified because each node of the trees requires the examination of only one component of the training vector to see if it lies above or below a threshold. The k-d tree can be very efficient for codevector search, but it requires careful design based on the training data.

Another approach to reducing the complexity of a full search is based on triangle inequality elimination (TIE).[9 16] Reference points called anchor points are used in this approach. Their distances to each of the codevectors are precomputed and stored. The encoder then computes the distance between a training vector and each anchor point. After that, some simple comparisons using the precomputed data eliminate a large number of codevectors as candidates for the best-matching codevector. There is a trade-off between the search speed and the size of the precomputed data table.[10 12] Because the TIE-based methods require considerable memory to store the precomputed data table, a mean-ordered algorithm was developed to reduce the memory space.[13] Multiple triangle inequalities were then developed to further reduce the computational complexity.[14 16]

Other inequality elimination methods were also developed. A norm-ordered fast search algorithm was proposed by Wu and Lin[17] that calculates the norms of codevectors and sorts the codevectors based on the norms before searching. In the search process, codevectors are rejected from being searched through based on an inequality defined by the stored norms and the norm of the training vector.

In another work,[18] the distances from training vectors to the origin, their squares, sines, and cosines are calculated and stored in a predefined data structure. An inequality-of-cosines law was used to reduce the codevector searching area of the full-search process.

A more-sophisticated codevector search algorithm was proposed.[19] It introduced two extra codevector elimination criteria based on the mean and the variance of codevectors in addition to the criterion used in the work by Wu and Lin[17] to further reduce the search space. Fast search methods using the topological structure of the codebook were also introduced to eliminate unnecessary codevector matching.[20 22]

A method for speeding up the GLA, which uses neither the k-d trees approach nor the inequality elimination approach, was proposed by Kaukoranta et al.[23] It detects the activity of codevectors and uses this information to classify training vectors. For training vectors whose current codevector has not been modified, only the distances between them and the active codevectors are calculated. A large portion of the distance calculations can be omitted without sacrificing the optimality of the GLA. It further sped up the processing by a factor of over 2 when it was applied to several fast

GLA variants, such as partial distance search,[24] TIE,[9] and mean-distance-order partial search.[13]

A simple and effective fast codevector search method of the same category was proposed by Qian.[25] It modifies the GLA such that a training vector does not require a search to find the best matching codevector if its distance to the MDP is improved in the current iteration. Similar improvement of the processing time as in the work by Kaukoranta et al.[23] was achieved, but it is much simpler and requires only a minor modification of the GLA. The codebook generated by Qian[25] might not be identical to the GLA codebook. This is because the algorithm classifies a training vector to the same best matching codevector of the previous iteration if the vector moves closer to the MDP in the current iteration, whereas in the GLA, a training vector might not be classified to the same best MDP of the previous iteration, even though it moves closer to the MDP in the current iteration. This fast codevector search method is described in Section 4.3 in detail.

Following this work, Qian further improved the search method for VQ compression techniques.[26] The improved search method makes use of the fact that in the GLA, a vector in a training sequence is either placed in the same MDP as in the previous iteration or in a partition within a very small subset of partitions. The proposed method searches for the MDP for a training vector only in this subset of partitions plus the single partition that was the MDP in the previous iteration. Because the size of this subset is much smaller than the total number of codevectors, the search process is significantly faster. This method generates a codebook identical to that generated using the GLA, but it is much faster. This fast codevector search method is described in Section 4.4 in detail.

4.3 Fast Vector-Quantization Algorithm Based on Improved Distance to MDP

This section describes a fast VQ compression algorithm, which speeds up the conventional GLA. It has the advantages of being simple, requiring only a minor modification of the GLA and a very small amount of additional memory, and producing results as good as the GLA. It can be easily applied to existing fast methods to further reduce computation time.

Assume a training sequence $\{\mathbf{X}_j\}\, j = 1, 2, \ldots, n$ with n training vectors. In training a codebook with N codevectors, each of which is the centroid corresponding to a partition, the GLA makes a full search of all N partitions $\{\mathbf{Y}_i\}\ i = 1, 2, \ldots, N$ for finding the MDP:

$$\mathbf{Y}_m = \min\left[d(\mathbf{X}_j, \mathbf{Y}_1), d(\mathbf{X}_j, \mathbf{Y}_2), \ldots, d(\mathbf{X}_j, \mathbf{Y}_N)\right] \tag{4.3}$$

for a vector $\mathbf{X}_j$ in the training sequence. This proposed fast VQ algorithm attempts to reduce the full-search to a much-smaller number than N as well as to reduce the accumulation operation in calculation of the vector distance

$d(\mathbf{X}_j, \mathbf{Y}_i)$ between a training vector $\mathbf{X}_j$ and a partition $\mathbf{Y}_i$ while maintaining the same quality of the codebook as the GLA algorithm.[25]

4.3.1 Analysis of the generalized Lloyd algorithm for fast training

The codebook training in the GLA is an iterative process. It is observed from analysis of the GLA full-search process that most of the vectors in a training sequence are placed in the same MDP as in the previous iteration when the training moves to the current iteration, although the codevectors have been updated at the end of the previous iteration. Figure 4.2 shows examples of the percentage of training vectors that remain in the same MDP in the next training iteration as a function of iteration number for codebook sizes $N = 16$ to 2048. Over 80% of the vectors in the training sequence remain in the same MDP when the training process goes to the next iteration. The percentages increase to 98% when the iteration number goes up to 20.

This observation indicates that only a very small portion of vectors in the training sequence change their MDP from one iteration to the next. If the changing vectors could be known beforehand, the MDP could be searched for them, which would speed up the full-search process dramatically. Unfortunately, this is not possible before conducting the full search.

In order to obtain information that can be used to determine which vectors in the training sequence do not need a full search, the training vectors that remain in the same MDP in the next iteration were further analyzed. Two hyperspectral datacubes, Jasper Ridge and Cuprite, acquired by AVIRIS were used. Analysis showed that a majority of the vectors had their distance to the MDP improved when they remained in the same MDP, which suggests the possibility that a training vector will most likely be placed in the same MDP

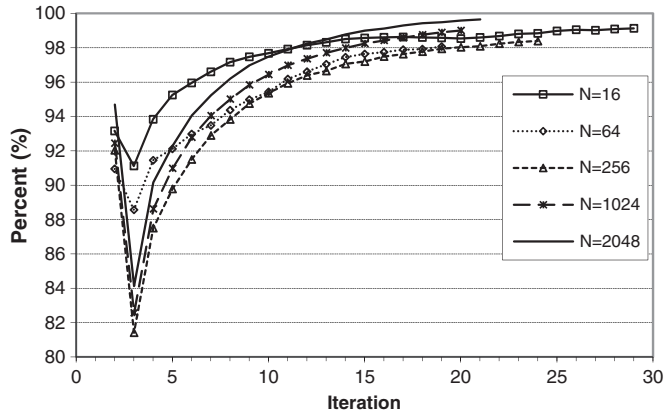


Figure 4.2 Percentage of training vectors that remain in the same MDP in the next iteration as a function of training iteration number for codebook sizes $N = 16$ to 2048 (modified and reprinted from Ref. 25).

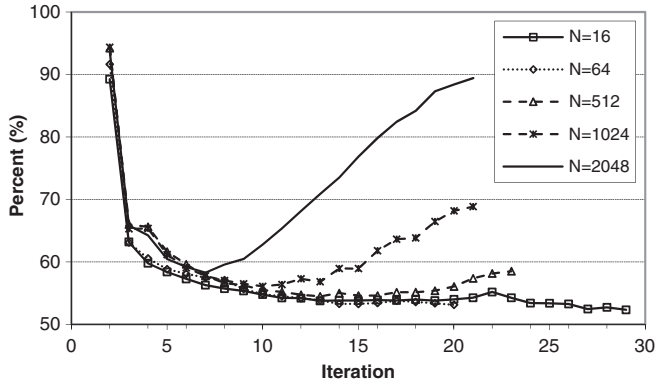


Figure 4.3 Percentage of training vectors that do not need a search when they remain in the same MDP as a function of iteration number for codebook sizes $N = 16$ to 2048 (modified and reprinted from Ref. 25).

as in the previous iteration if the distance between the training vector and the partition has improved compared to the previous iteration. This possibility was tested to determine whether or not a vector in the training sequence needs a full search to find its MDP. A training vector is placed in the same MDP as in the previous iteration if the distance between the training vector and the partition is improved in the current iteration. Because no search is conducted for this kind of training vector, computation time is saved during each iteration.

Figure 4.3 shows the results of the analysis for the percentage of training vectors that do not need a search because they remained in the same MDP and their distance to the MDP was improved. It can be seen that in the earlier iterations the percentage is very large (up to 95%). With more iteration loops the percentage tends to be in a constant range (50–55%). The percentage climbs in late iterations for the cases of large codebook sizes ($N = 512$, 1024, and 2048). This is encouraging because the computation time savings are proportional to the codebook size N in the proposed method. The high percentage of training vectors that do not require a search for a large-size codebook will produce even more time savings. The experimental results for these cases will be shown later in the chapter.

The MDPs obtained by the proposed method were compared with those obtained by a full search to examine the effectiveness of the proposed method. The experimental results show that 96% of the vectors in the training sequence that the MDPs found by the proposed method are identical to those found by a full search in early iterations. This is because the partition adjustment is unstable in early iterations. The identification rate goes up to 99.99% in the later iterations. Figure 4.4 shows the frequency of disagreement of MDP between the two methods as a function of the iteration number. It can be seen

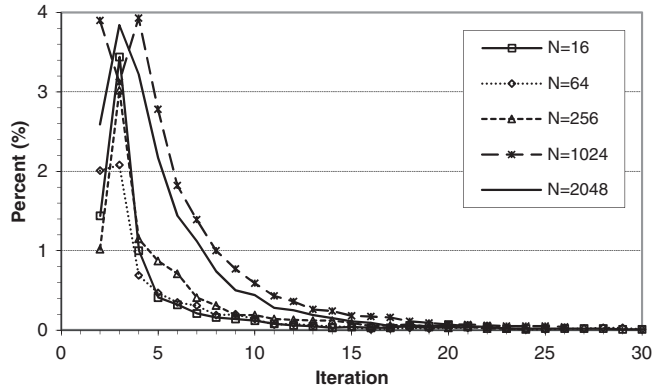


Figure 4.4 Frequency of disagreement of MDP found by the GLA and by the proposed method as a function of training the iteration number for codebook sizes $N = 16$ to 2048 (modified and reprinted from Ref. 25).

that the frequency reduces to 0.01% in later iterations, which is too small to have an impact on the codebook quality.

4.3.2 Fast training based on improved distance to MDP

The flowchart of the proposed fast-training algorithm is shown in Fig. 4.5. The codebook training iteration begins after initialization of N codevectors, which are the centroids of the N partitions. The proposed algorithm operates the same as the GLA in the first iteration. As shown in the right column of the flowchart, for each vector $\mathbf{X}_j$ in the training sequence, a full search is applied to find its MDP. After the minimum distance $d_{\min}$ of vector $\mathbf{X}_j$ is found, it is stored in $d^p_{\min}(j)$. The index I of the MDP is also stored in $IX(j)$. They are the two parameters required in the next iteration for the fast MDP search.

After the first iteration, for each vector $\mathbf{X}_j$ in the training sequence, the distance $d(\mathbf{X}_j, \mathbf{Y}_I)$ between the training vector and the partition that was the MDP in the previous iteration identified by $I = IX(j)$ is calculated. The same partition as in the previous iteration is assigned to the training vector as the MDP if $d(\mathbf{X}_j, \mathbf{Y}_I) < d^p_{\min}(j)$, where $d^p_{\min}(j)$ is the minimum distance between the training vector and the partition in the previous iteration. No search is applied to the vector, as shown in the left column of Fig. 4.5. Otherwise, a full-search process is applied to find the MDP for the training vector, as shown in the right column.

The proposed algorithm also attempts to reduce the cost of computation of $d(\mathbf{X}_j, \mathbf{Y}_i)$ using partial distance.[24] In a full-search process, a distance $d(\mathbf{X}_j, \mathbf{Y}_i)$ between training vector $\mathbf{X}_j$ and each of the codevectors $\mathbf{Y}_i$ ($i = 1, 2, \ldots, N$) is calculated and then compared with the current minimum distance $d_{\min}$. Given that the distance $d(\mathbf{X}_j, \mathbf{Y}_i)$ is calculated on an

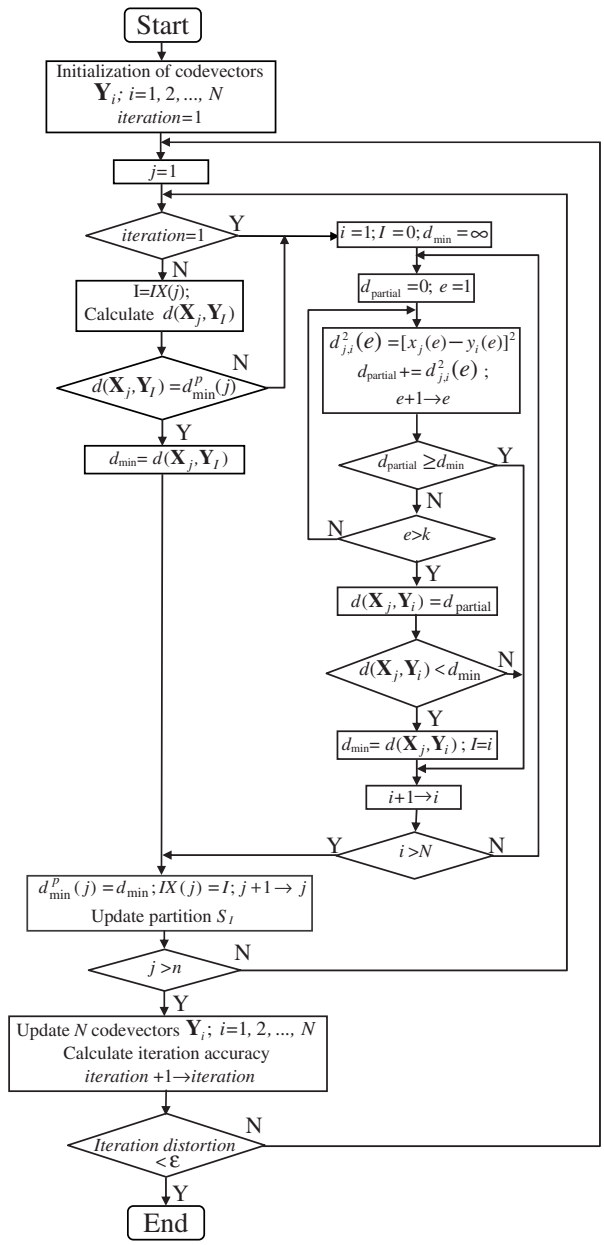


Figure 4.5 Flowchart of the fast VQ algorithm based on improved distance to the MDP (reprinted from Ref. 25).

element-by-element basis of the vectors, the accumulator $d_{partial}$ carries the partial distance between the two vectors so far, up to the element e. The distance $d(\mathbf{X}_j, \mathbf{Y}_i)$ will be greater than the current minimum distance $d_{\min}$ if the partial distance $d_{partial}$ at element e is equal to or greater than the current

minimum distance $d_{\min}$. Hence, it is not necessary to complete the rest of the calculation of the distance if $d_{partial} \geq d_{\min}$. The calculation is aborted once $d_{partial} \geq d_{\min}$, as shown in the flowchart. This speeds up the calculation of vector distance and ultimately the full-search process.

The additional memory requirements of the proposed algorithm are n long integers for storing n minimum distance $d^{p}_{\min}(j)$ of the previous iteration, where n is the total number of training vectors.

4.3.3 Experimental results

Two hyperspectral datacubes, Jasper Ridge and Cuprite, acquired by AVIRIS (http://popo.jpl.nasa.gov/html/aviris.freedata.html/aviris.free-data.html) in 1992 and 1996, respectively, were used. They have been converted to at-sensor radiance and are stored as 16-bit digital numbers (DNs). They contain typical scenes such as mineral, urban, cloud, etc., and are widely used in the hyperspectral community. The size of the datacubes was originally 512 lines by 614 pixels per line with 224 spectral bands. In this work, a subset of them with 256 lines by 256 pixels with all 224 bands was tested. A spectrum in the datacube with 224 spectral values was taken as a vector so that it has 224 elements.

In order to test the usefulness of the proposed method for a much-wider variety of hyperspectral scene types, two other hyperspectral datacubes were also tested. One datacube contains artificial targets and forest and cloud scene types for target detection; it was acquired using Short-Wave Infrared Full-Spectrum Imager II (SFSI-II)[27] with a ground sample size of 3.5 m $\times$ 3.5 m and 240 spectral bands between 1200 nm and 2450 nm, with a band interval of 5 nm. The size of the datacube is 140 lines by 446 pixels per line, with 240 spectral bands. Another datacube, called Acadie, contains urban and agriculture scenes, as shown in Fig. 4.6.

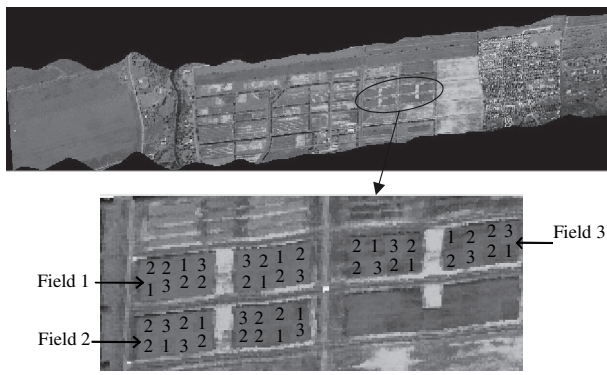


Figure 4.6 The scene of the CASI Acadie datacube (reprinted from Ref. 25). For a color version of this figure, see Plate 2 in the color plate section of this book.

It was acquired over a cornfield near the small town L'Acadie, Quebec, Canada using CASI with a ground sample size of 2 m × 2 m and 72 spectral bands in the wavelength range of 408–947 nm, with a band interval of 7.6 nm. The size of the datacube is 1225 lines by 406 pixels per line, with 72 spectral bands.[28]

Peak signal-to-noise ratio (PSNR) and signal-to-noise ratio (SNR) were used to measure the quality of the codebooks and the statistical performance of the compression data. For convenience, their definitions are rewritten as follows:

$$PSNR = 10\log_{10}\frac{Peak\ signal^2}{MSE}, \tag{4.4}$$

$$SNR = 10\log_{10}\frac{Signal\ power}{MSE}, \tag{4.5}$$

where

$$Signal\ power = \frac{1}{n_x n_y n_b}\sum_{x\ 1}^{n_x}\sum_{y\ 1}^{n_y}\sum_{b\ 1}^{n_b}\left[DN_O(x,y,b)\right]^2, \tag{4.6}$$

$$MSE = \frac{1}{n_x n_y n_b}\sum_{x\ 1}^{n_x}\sum_{y\ 1}^{n_y}\sum_{b\ 1}^{n_b}\left[DN_O(x,y,b) \quad DN_D(x,y,b)\right]^2. \tag{4.7}$$

$DN_O(x,y,b)$ and $DN_D(x,y,b)$ are the DNs of the original and the reconstructed data of band b at location (x,y), n_y is the total number of lines in the datacube, n_x is the total number of pixels per line, and n_b is the total number of bands in the datacube. The *Peak signal* used in Eq. (4.4) is the maximum value in the datacube.

Tables 4.1–4.4 show the training and compression results for the four hyperspectral datacubes with codebook sizes $N = 16-2048$. The iteration distortion threshold ε of the codebook training process was set to 0.001 in the experiments in order to attain better codebook fidelity. The number of iterations and computation times (CTs) for both the GLA and the proposed method are listed. The ratio of computation time improvement (CTg/CT) and the difference between PSNRs and SNRs yielded by the GLA and by the proposed method are also shown. The proposed method improves the computation time by a factor of 3.08–27.35. The larger the codebook size is, the more time savings can be achieved—this is because there are more training vectors that do not need a search, as shown in Fig. 4.3, when the codebook size is larger. The CT and CTg were the times used to compress the datacubes by a Sun Fire 280R workstation with a 900-MHz UrltraSPARC-III+ CPU.

Table 4.1 Codebook training and compression results for the AVIRIS Jasper Ridge datacube.

		GLA (Full-search)			Proposed Method				
Codebook Size (*N*)	**Comp. Ratio**	**Itera-tions**	**CTg (sec)**	**PSNRg | SNRg**	**Itera-tions**	**CT (sec)**	**PSNR | SNR**	**CTg/ CT**	**PSNRg - PSNR (dB)**
16	735	29	273.3	40.60 | 25.03	33	72.4	40.57 | 25.00	3.77	0.03
32	531	24	443.8	42.26 | 26.69	26	80.6	42.20 | 26.63	5.51	0.06
64	377	20	731.1	43.60 | 28.03	22	94.2	43.53 | 27.96	7.76	0.07
128	256	18	1318.9	44.71 | 29.14	22	139.6	44.69 | 29.12	9.45	0.02
256	163	25	3716.2	45.86 | 30.29	24	238.2	45.87 | 30.30	15.60	0.01
512	97	21	6224.4	46.81 | 31.24	23	422.2	46.88 | 31.31	14.74	0.07
1024	54	21	12393.2	47.51 | 31.94	21	747.1	47.53 | 31.96	16.59	0.02
2048	29	21	24768.5	48.28 | 32.71	21	1584.7	48.30 | 32.73	15.63	0.02

Table 4.2 Codebook training and compression results for the AVIRIS Cuprite datacube.

		GLA (Full-search)			Proposed Method				
Codebook Size (*N*)	**Comp. Ratio**	**Itera-tions**	**CTg (sec)**	**PSNRg | SNRg**	**Itera-tions**	**CT (sec)**	**PSNR | SNR**	**CTg/ CT**	**PSNRg - PSNR (dB)**
16	735	26	327.9	40.03 | 27.60	26	106.5	40.03 | 27.60	3.08	0
32	531	28	502.6	42.26 | 29.83	30	122.0	42.26 | 29.83	4.12	0
64	377	32	1138.8	43.53 | 31.10	33	217.7	43.55 | 31.13	5.23	0.02
128	256	41	3259.7	49.72 | 35.10	41	499.5	49.59 | 34.98	6.53	0.13
256	163	57	9202.6	51.55 | 36.94	55	1547.3	51.46 | 36.85	5.95	0.09
512	97	52	17004.4	54.43 | 39.82	54	1470.7	54.48 | 39.88	11.56	0.05
1024	54	24	15730.1	55.78 | 41.16	22	682.4	55.80 | 41.18	23.05	0.02
2048	29	18	23611.3	56.83 | 42.21	22	1035.2	56.89 | 42.27	22.81	0.06

Table 4.3 Codebook training and compression results for the SFSI datacube.

		GLA (Full-search)			Proposed Method				
Codebook Size (*N*)	**Comp. Ratio**	**Itera-tions**	**CTg (sec)**	**PSNRg | SNRg**	**Itera-tions**	**CT (sec)**	**PSNR | SNR**	**CTg/ CT**	**PSNRg - PSNR (dB)**
16	714	22	151.9	40.38 | 28.19	20	38	40.35 | 28.16	4.00	0.03
32	495	28	381.2	41.81 | 29.62	24	57.2	41.77 | 29.58	6.66	0.04
64	334	21	567.8	42.95 | 30.76	18	54.1	42.92 | 30.73	10.50	0.03
128	213	19	1027.4	44.11 | 31.92	23	95.2	44.11 | 31.92	10.79	0
256	128	15	1647.4	45.02 | 32.83	20	115.2	45.03 | 32.84	14.30	0.01
512	72	19	4189.8	45.95 | 33.78	21	188.5	45.97 | 33.79	22.23	0.02
1024	39	16	7071.8	46.87 | 34.67	19	284.6	46.88 | 34.69	24.85	0.01
2048	20	14	12372.1	47.86 | 35.67	16	452.3	47.87 | 35.68	27.35	0.01

It can be seen that for the four test datacubes, the proposed method produced codebooks as good as the GLA when used to compress the data. In 17 compression cases out of 32, the PSNRs (SNRs) are slightly improved (up to 0.07 dB). In five cases the PSNRs remain the same. In 10 cases the

Table 4.4 Codebook training and compression results for the CASI Acadie datacube.

		GLA (Full-search)			Proposed Method				
Codebook Size (N)	Comp. Ratio	Itera-tions	CTg (sec)	PSNRg | SNRg	Itera-tions	CT (sec)	PSNR | SNR	CTg/ CT	PSNRg - PSNR (dB)
16	285	21	495.9	32.39 | 19.47	24	111.3	32.44 | 19.51	4.46	0.05
32	227	67	3108.7	35.54 | 22.62	68	693.7	35.58 | 22.62	4.48	0
64	187	54	4961.9	37.74 | 24.81	55	902.4	37.74 | 24.81	5.50	0
128	158	48	8773.6	39.69 | 26.76	46	883.6	39.66 | 26.73	9.93	0.03
256	134	48	17506.6	41.32 | 28.40	49	1195.4	41.34 | 28.42	14.64	0.02
512	113	45	30376.3	42.73 | 29.81	46	1761.3	42.75 | 29.83	17.25	0.02
1024	93	38	56709.3	44.05 | 31.12	42	3096.6	44.06 | 31.13	18.31	0.01
2048	73	41	122672	45.27 | 32.35	42	5539.9	45.30 | 32.37	22.14	0.03

PSNRs are slightly worse. The maximum loss of PSNR is 0.13 dB for AVIRIS Cuprite data with $N = 128$. The PSNR loss was caused by the disagreeing MDP found by the GLA and the proposed method. It is possible that the partition that was the MDP in the previous iteration may not be the closest partition for a training vector in the current iteration, even though the distance between the pixel and the partition is improved. In the GLA, a training vector might not be placed in the same MDP even though it moves closer to the MDP; whereas in the proposed method, a training vector is kept in the same MDP if it moves closer to the MDP. This difference may cause some training vectors to be classified suboptimally in an iteration process, introduce certain iteration error compared to the full-search process, and ultimately require more iteration loops to reach the predefined threshold. The experimental results shown in Tables 4.1–4.4 indicate that the proposed method took 1–2 more iteration loops on average than the full-search process in training a codebook.

The compression ratios are listed in the second column of the tables. They were calculated by dividing the number of bits in the original datacubes by the number of bits in the compressed data (including both the index map and the codebook; the inclusion of the codebook as the compressed data is discussed in Section 4.7).

4.3.4 Assessment of preservation of spectral information

The preservation of spectral information is critical in the data compression of hyperspectral imagery, as spectral information is the fingerprint of the target objects. Extra care for the spectral information should be taken, as its loss decreases the value of hyperspectral data for remote sensing applications. Extensive studies of the impact of VQ-based data compression on hyperspectral imagery for remote sensing applications have been reported.[29 35] Chapter 8 assesses the impact data compression has on the derivation of final remote sensing products.

The preservation of spectral information for the CASI and SFSI datacubes, when VQ-based data compression is used, has been reported by Hu et al.[33] and Qian et al.[35] This chapter uses the Cuprite datacube for evaluation as an example of how compression can preserve the original information. The spectral angle mapper (SAM) is used to evaluate the preservation of spectral features of the data after compression. A typical spectrum profile corresponding to a spatial sample in the scene of the datacube before and after compression will be shown. Abundance images (derived using spectral unmixing) from the datacube before and after compression are produced to assess the impact of data compression on remote sensing applications.

A compressed datacube with a compression ratio of 29:1 ($N = 2048$) was selected for evaluation because a ratio between 10:1 and 30:1 meets most of the requirements for onboard hyperspectral data compression. Compression ratios above this range are usually suitable for a quick look or browsing of hyperspectral data, as noise introduced with higher compression ratios is unlikely to be acceptable for remote sensing applications.

The SAM is defined as follows:

$$SAM = \cos^{-1}\left[\frac{\sum_{b=1}^{n_b} DN_O(b) \cdot DN_D(b)}{\sqrt{\sum_{b=1}^{n_b} [DN_O(b)]^2 \sum_{b=1}^{n_b} [DN_D(b)]^2}}\right], \tag{4.8}$$

where $DN_O(b)$ and $DN_D(b)$ are digital numbers at spectral band b of an original spectrum and the reconstructed spectrum after compression at the same location of the scene, and n_b is the total number of spectral bands. SAM varies between 0 and π (radians), where 0 indicates a perfect match between the original and the reconstructed spectra. The SAM between each spectrum of the original Cuprite datacube and that of the reconstructed datacube was calculated; the SAM image is shown in Fig. 4.7. The displayed scale is stretched to between 0 and 0.03 rad (1.71 deg) in order to highlight the difference. The statistical values of the SAM image are mean = 0.003 rad (0.17 degrees), standard deviation = 0.001 rad (0.06 deg), minimum = 0 rad, and maximum = 0.041 rad (2.39 deg). The spectral angles between spectra of the original data and those of the reconstructed data are generally smaller than 0.003 rad. The maximum spectral angle between the two datacubes is located at spatial sample (432, 85). This maximum value is difficult to identify by eyesight alone, as it is a single point in the scene, whereas a bright spot (bottom left), which is composed of a cluster of about 20 "high" valued samples, is more visible. The maximum value of these samples is 0.037 rad (2.12 deg). These highest spectral angle values likely correspond to the presence of spectral anomalies, and all of the highway sample values in the scene are smaller than 0.010 rad (0.57 deg).

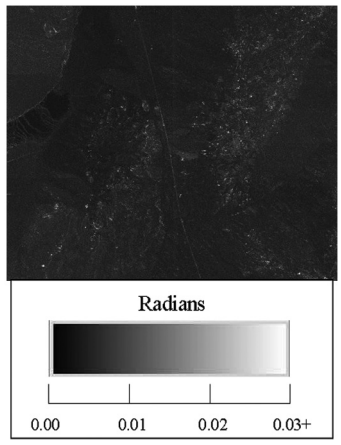


Figure 4.7 A SAM image between spectra of original Cuprite and those of compressed data at a compression ratio of 29:1 (the darkest denotes 0 of the spectral angle value). Reprinted from Ref. 25.

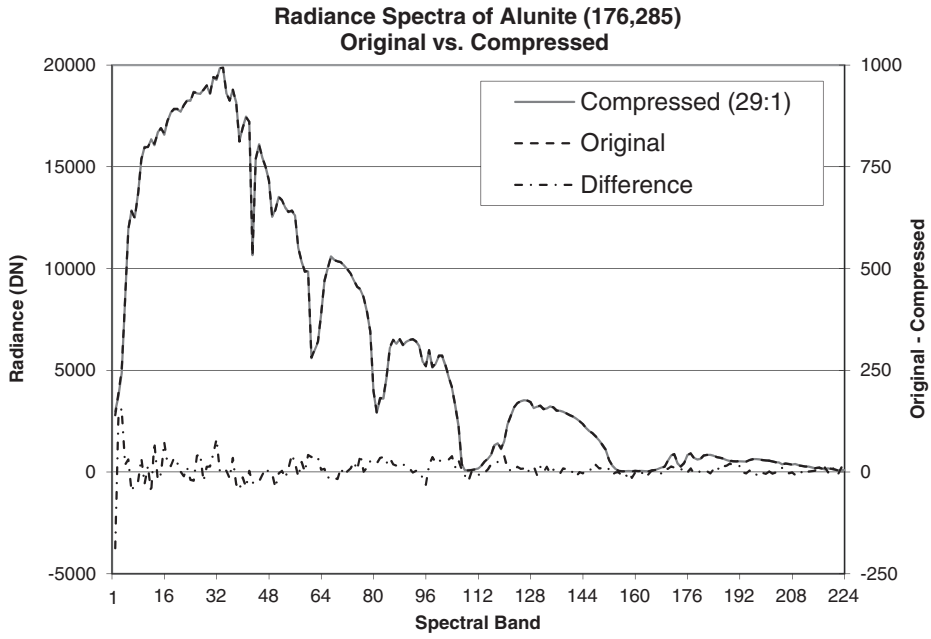


Figure 4.8 Spectral profile of a spatial sample of the Cuprite datacube before and after compression (29:1), and the difference. The compressed spectrum is overlapping well with the original (reprinted from Ref. 25). For a color version of this figure, see Plate 3 in the color plate section of this book.

Figure 4.8 shows the spectral profile of a spatial sample at location (176, 285) of the original Cuprite data, the reconstructed spectrum, and their difference. It can be seen that the two spectra are very similar. They overlap each other very well even in the absorption band regions. Their differences appear to be random,

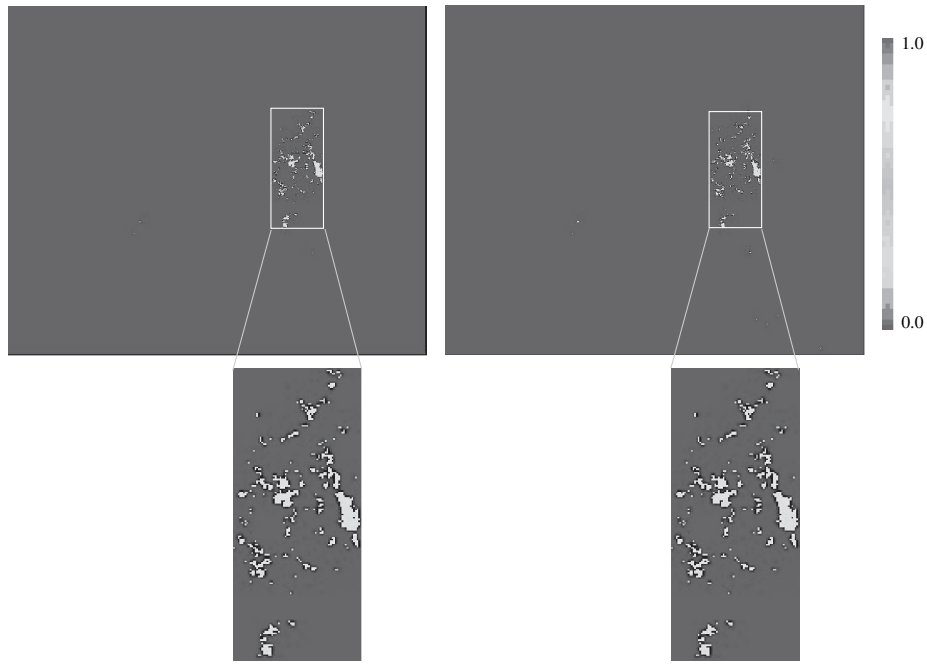


Figure 4.9 Abundance images (kaolinite) derived from (a) the original and (b) the reconstructed Cuprite datacube at a compression ratio of 29:1 (reprinted from Ref. 25).

and no large discontinuities in the absorption band regions can be seen. The relative rms error (R-RMSE) equals 0.85%. This spectrum is selected as an example because it is the purest pixel of the mineral alunite, which is one of the endmembers in the following spectral unmixing example.

Endmembers required for the spectral unmixing were extracted from the original Cuprite datacube and the reconstructed datacube using an automated method called iterative error analysis. Both the original and the reconstructed datacubes were unmixed with their own endmembers using a constrained linear technique to form the abundance images corresponding to the set of endmembers. Figure 4.9 shows the abundance images for the endmember kaolinite derived from the original Cuprite datacube and from the reconstructed datacube. It can be seen that the compressed datacube produces an abundance image similar to the original. A detailed evaluation and discussion can be found in the work by Staenz et al.[34]

4.4 Fast Vector Quantization Based on Searching Nearest Partition Sets

Following the discussion in the previous section, this section describes the further development of fast VQ searching.[26] This fast search algorithm makes use of the fact that, in GLA, a vector in a training sequence is either placed in

the same MDP as in the previous iteration or in a partition within a very small subset of partitions referred to as nearest partition set.

As described in Section 4.3, it is observed from the analysis of the GLA that at any given iteration loop, most of the vectors in a training sequence are placed in the same MDPs as in the previous iteration, although the centroids (i.e., codevectors) of the MDPs are updated at the end of the previous iteration. Over 80% of the vectors in the training sequence remain in the same MDPs when the training process goes to the next iteration; the percentage increases to 98% when the iteration number goes up to 20. This fact indicates that only very small portions of vectors in the training sequence change their MDPs from one iteration loop to the next. Unfortunately, it is unknown which ones they are before going through the full search.

4.4.1 Nearest partition sets

In order to study the training vectors that change their MDPs in the next iteration, an investigation of these vectors was performed to see how they are trained in the full-search process. It is observed from the investigation that a training vector that changes its MDP in the next iteration is placed in one of p nearest-neighbor partitions of the vector's MDP in the previous iteration. This set of p partitions is referred to as the nearest partition set (NPS). Figure 4.10 shows the p nearest-neighbor partitions of the vector's MDP in the previous iteration and illustrates how the training vector is classified to a new MDP within the NPS. In the figure, a black point stands for a 2D vector in an xy coordinate system. A training vector v (marked as a small white circle) has been assigned to partition A as its MDP in the previous iteration. The nearest neighbor partitions

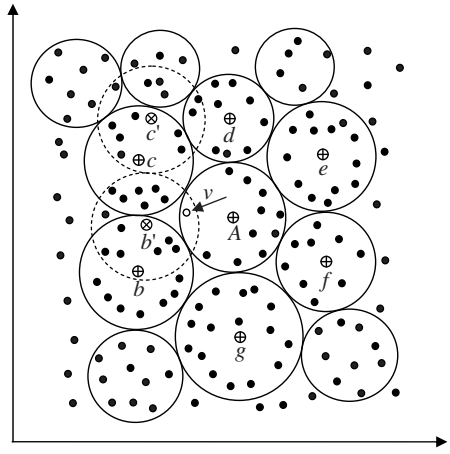


Figure 4.10 Six ($p = 6$) nearest-neighbor partitions (b, c, d, e, f, and g, marked with a solid circle) of MDP A of a training vector v in the previous iteration; and the two updated partitions (b' and c', marked with broken circle) at the end of the previous iteration.

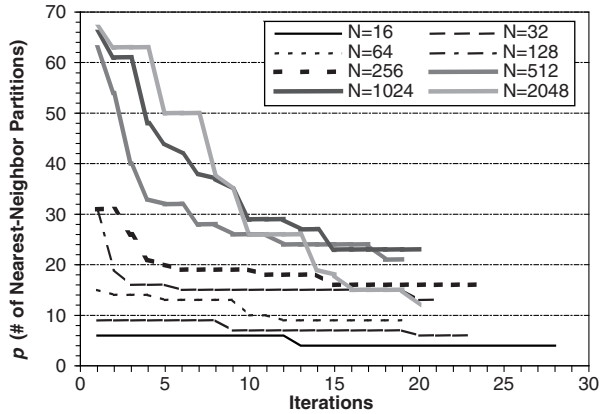


Figure 4.11 Sizes of nearest partition sets for training vectors that change their MDP in the next iteration as a function of iteration number, when codebooks of size $N = 16$ to 2048 are trained (reprinted from Ref. 26).

of partition A are partitions b, c, d, e, f, and g, marked with the solid circles. The centroids of the nearest-neighbor partitions are marked with $\oplus$. At the end of the previous iteration, all of the partitions are updated. Figure 4.10 shows the partitions b' and c' (marked with broken line) updated from partitions b and c. Their centroids are marked with $\otimes$. In the current iteration, the training vector v falls within the circle of partition b' and is assigned to partition b' as its MDP, which is one of the nearest-neighbor partitions.

Figure 4.11 shows examples of the investigation results—p values (i.e., sizes of NPSs)—for training vectors that change their MDP in the next iteration as a function of iteration number for codebook sizes N from 16 to 2048. It can be seen that p at each iteration is very small compared with the total number of partitions N. The p values decrease with the increase of the iteration number. For the case of codebook size $N = 2048$, p decreases from 67 to 12 from iteration loops 1 to 20; this represents 3.3% to 0.6% of the total number of partitions, respectively. For the case of codebook size $N = 16$, p is between 6 and 4 from iteration loops 1 to 28; this represents 37% to 25% of the total number of partitions, respectively. The larger the number of partitions N (i.e., codebook size) is, the smaller the percentage of the NPS size out of the total number of partitions. This important observation of the characteristic of the GLA is used to speed the codebook training process, especially for a codebook with a large size.

It can be concluded that a training vector is either placed in the same partition as in the previous iteration as the MDP or placed in a partition within the NPS of its MDP in the previous iteration. The full-search can be significantly accelerated by searching only in the NPS plus the single previous MDP, as the size of the NPS is much smaller than the total number of partitions. The only overhead is to set up the NPS of the previous MDP.

A description of the process for generating NPSs follows. There are N partitions if the codebook size is N. Each partition has a unique NPS, and thus there are a total of N NPSs. The NPS of a partition does not change at an iteration process and will be used many times for the partition that was a MDP in the previous iteration. It should be ready for use before commencing the next iteration. The logical time to set up the NPSs is right after the codevector update of the previous iteration.

The scheme used to create the NPSs directly affects the speed improvement of the proposed search method, as the NPSs need to be created at each iteration process. The proposed search method would not improve the overall processing time if the processing time spent on creating the NPSs is of the same order as the time saved by the proposed method. The schemes that efficiently generate the NPSs are described as follows: (1) the upper-triangle matrix of distances to reduce the number of calculations of the distances between a partition and all of its neighbors; (2) a fast sorting method for sorting partitions; and (3) the approach to determining the size of the NPSs.

4.4.2 Upper-triangle matrix of distances

The first step to generate the NPSs is to calculate the distances between a partition i and all of its neighbors $d(i,j)(i = 1, 2,\ldots, N; j = 1, 2,\ldots, N; i \neq j)$, so that its neighbor partitions can then be sorted according to the distances.

A partition has N 1 neighbor partitions, assuming that a codebook of size N is being trained, and thus N 1 distances need to be calculated before sorting. There are a total of $N \times (N$ 1) distances for all N partitions. In fact, only $N \times (N$ 1) / 2 distances are unique and need to be calculated, as the distances between partitions A and B and between partitions B and A are the same.

Figure 4.12 shows an upper-triangle matrix of distances that illustrates how the distances between a partition and its N 1 neighbors are calculated

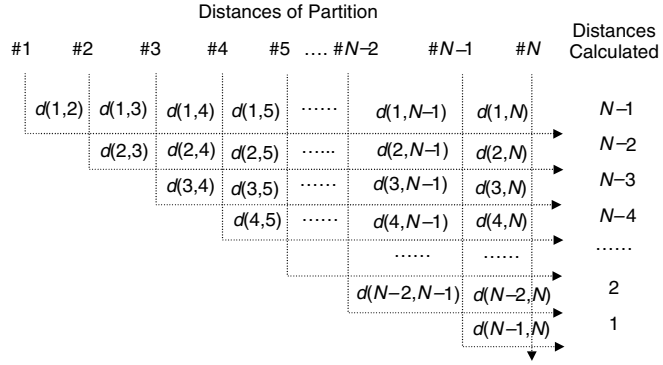


Figure 4.12 Process of calculation and organization of distances between a partition and its N – 1 neighbors (reprinted from Ref. 26).

and organized for sorting in order to reduce the number of distance calculations. For partition #1, its distances to all N 1 neighbor partitions $d(1,j)(j = 2, 3,\ldots, N)$, underlined by dotted line #1 in the figure, are calculated and then used for sorting its neighbors. For partition #2, its distance to partition #1 $d(2,1)$ is the same as distance $d(1,2)$, which has been calculated. Only N 2 distances $d(2,j)(j = 3, 4,\ldots, N)$, underlined by dotted line #2, need to be calculated; these N 2 distances together with $d(1,2)$ are used for sorting the partition's neighbors. For partition #3, only N 3 distances $d(3,j)(j = 4, 5,\ldots, N)$, underlined by dotted line #3, need to be calculated, as its distances to partition #1 and #2, $d(i,3)(i = 1, 2)$ in the vertical line for #3, have been calculated. The $N - 3$ distances plus the calculated distances $d(i,3)(i = 1, 2)$ in the vertical line for #3 [all of the distances $d(.,.)$ crossed by dotted line #3] are used for sorting the neighbors of partition 3. For partition #N, no distance needs to be calculated, as the distances to its N 1 neighbors $d(i,N)$ $(i = 1, 2, \ldots, N$ 1) [all of the distances $d(.,.)$ strikethrough by dotted line #N] have all been calculated previously. They are used for sorting the neighbors of partition N.

4.4.3 *p*-least sorting

The next step is to sort the N 1 neighbors of each partition according to their distances to the partition from nearest to farthest. "QuickSort" is a widely used fast-sorting algorithm. Its complexity is $O(n \log_2 n)$ when n values are sorted. An average of $N \times (N$ 1)$\log_2(N$ 1) comparisons and swaps are required to sort N NPSs at each iteration if QuickSort is used. This is still a large processing burden. In addition, when sorting the neighbor partitions, the identities (IDs) of the neighbor partitions need to be transposed with the distance values in order to generate the lookup table of the NPSs.

The proposed method is referred to as "p-least sorting;" it is simpler and faster than QuickSort. The important features of the data to be sorted are taken into account. That is, for N 1 neighbors of a partition, only the first p nearest neighbors need to be sorted (p is much smaller than N 1). It eliminates swap operations. Figure 4.13 illustrates the sorting process of the p-least sorting. A window containing $p = 4$ least values sorted so far is used to sort a dataset having 16 values with the ID#. The beauty of the p-least sorting is that it does not require a comparison to all of the values in the window when a value in the dataset is equal to or larger than the last value in the window. For example, the values (and their corresponding IDs) 99 (#5), 21 (#11), 92 (#12), 77 (#13), 75 (#14), 33 (#15), and 31(#16) in the dataset are sorted using only one comparison. If a value in the dataset is smaller than the value(s) in the window, only assignment(s) are required to place the value in the appropriate location in the window. For value 43 (#6) in the dataset, for example, the algorithm detects that it should be located in the second place in the window after four comparisons. Only three assignments are needed to

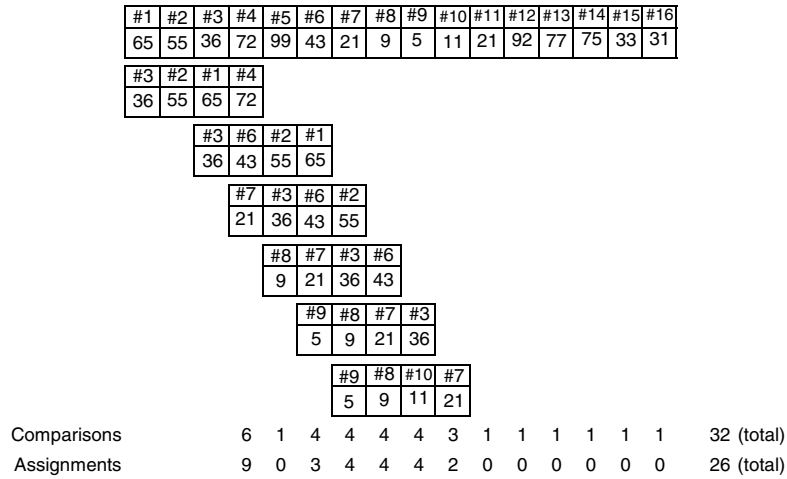


Figure 4.13 Illustration of p-least sorting method with $p = 4$ (reprinted from Ref. 26).

move 55 (#2) and 65 (#1) in the second and third place to the third and fourth place, and to place 43 (#6) in the second place.

The number of comparisons and assignments used by the p-least sorting are listed at the bottom of Fig. 4.13. Six comparisons and nine assignments (three swaps) were used to initiate the window. They are also shown at the beginning of the list. In this example, an assignment is counted as 1/3 swap, as three assignments are required to swap a value in QuickSort. A total of 32 comparisons and $26/3 \approx 9$ swaps are used to sort the dataset with the p-least sorting, whereas the QuickSort requires 75 comparisons and 16 swaps. The larger the size of the dataset to be sorted, the greater the improvement in processing time.

4.4.4 Determination of NPS sizes

The final step to create the NPSs is to determine p, the size of the NPSs. The investigation and analysis of the full-search process discovered that a training vector is placed either in a partition that was the MDP in the previous iteration or in one of the p nearest neighbor partitions of the previous MDP. The sizes of NPSs vary, but the range of variation is not large. In order to simplify the proposed method, the maximum one is used as the size for all of the NPSs:

$$p = \max\{p_i\}\ (i = 1, 2, \ldots, N), \tag{4.9}$$

where p_i is the size (i.e., maximum order number) of NPS_i ($i = 1, 2, \ldots, N$).

Unfortunately, p_i is not known before searching through all of the N 1 neighbor partitions of the MDP for each training vector. It is also observed that the sizes of NPSs decrease with the increase of iteration number. This makes sense because the codebook training error converges to the distortion

threshold ε with the increase of iterations. This observation is critical and used to determine p.

Assuming that $p' \geq p$ is the approximation size of the NPSs of the current iteration, the p_i of each NPS_i ($i = 1, 2, \ldots, N$) will be known at the end of the iteration, after the p' nearest partitions in each of the NPSs have been searched through to find the MDP for all the training vectors. The p value of the current iteration is thus obtained using Eq. (4.9). This p value is determined too late for the current iteration, but it can be used as p', the approximation size of the NPSs of the next iteration. Although it may be slightly larger than the real p value of the next iteration, it ensures that the NPSs of the next iteration contain all of the necessary candidates for finding the same MDPs as in the full-search process. This is because p values decrease with the increase of iteration number; use of the p value of the current iteration as that of the next iteration is a safe approximation. The only disadvantage is that slightly more nearest partitions in the NPSs may be searched when p' is slightly larger than the real p of the next iteration; however, the latter is unknown until the iteration is completed.

Figure 4.14 shows how p_i of each NPS_i ($i = 1, 2, \ldots, N$) is obtained at an iteration. p_i is the maximum order number of the partition that has been found being a MDP at least once within the distance sorted NPS_i. The maximum order number p_i of NPS_i is recorded during the course of codebook training at an iteration process. At the end of the iteration, the maximum p_i of all of the NPSs is extracted as p of the current iteration and used as the approximation size of the NPSs of the next iteration. In Fig. 4.14, $p = p_{N\ 1} = 10$, the

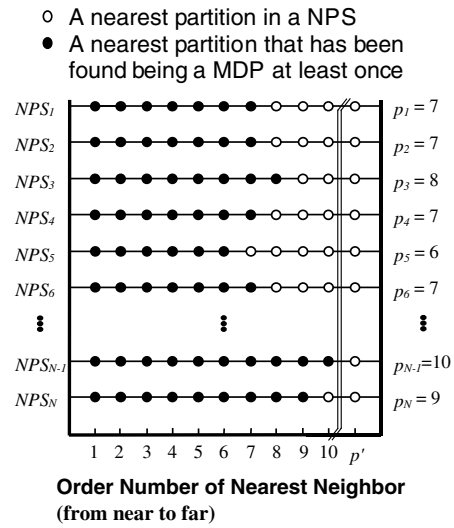


Figure 4.14 Generation of the maximum order number p_i of each NPS for determining p (reprinted from Ref. 26).

maximum order number of the current iteration is assigned to p' for the next iteration.

The first iteration is a full-search process. At the end of the first iteration, N NPSs are generated, each of which has $p' = N \quad 1$ neighbor partitions sorted from nearest to farthest. The p_i of each NPS is obtained by matching the MDP of each training vector to the NPS of its MDP to locate the maximum order number. The maximum p_i is p of the first iteration and is assigned to p' for the second iteration.

4.4.5 Two fast VQ search algorithms based on NPSs

Two algorithms are proposed in this section. Algorithm 1 searches for the MDP for a training vector in the NPS of the previous MDP plus the previous MDP. Algorithm 2 is the combination of the fast method described in Section 4.3 and the method proposed in this section to further improve the computation time.

4.4.5.1 Algorithm 1

From the second iteration onward, the full-search process for each training vector is replaced by searching only in the small NPS of the previous MDP of the training vector plus the previous MDP. The MDP found in this way is the same as that found in the full-search process because the NPS plus the previous MDP contains all necessary candidate MDPs (as in the full-search process). Because the size of the NPS is much smaller than the total number of partitions N, the codebook training process is greatly accelerated.

Figure 4.15 shows the flowchart of the codebook training process with search limited to the NPS plus the single previous MDP (the boxes with solid lines). The codebook training iteration begins after initialization of N codevectors. The proposed algorithm operates in the same way as the GLA in the first iteration, as shown in the right column of the flowchart. A full search is applied to find the MDP for each training vector $\mathbf{X}_j (j = 1, 2, \ldots, n)$. In the lower portion of the flowchart, the index I of the MDP for vector $\mathbf{X}_j$ is stored in $MDP(j)$, which will be used in the next iteration to identify a NPS for fast search of the vector. The search process for the first iteration is completed after all n vectors in the training sequence have had their MDP found and the partitions updated. Before the end of the first iteration, the algorithm updates the N codevectors, creates the N NPSs, and determines the p' for the second iteration using the method described in Sections 4.4.1–4.4.4.

After the first iteration, for each training vector $\mathbf{X}_j (j = 1, 2, \ldots, n)$, the index of the partition that was the MDP of the vector in the previous iteration stored in $MDP(j)$ is assigned as the current minimum-distance partition identity $I = MDP(j)$ (the box below the diamond-shaped box named "iteration = 1?" in the center column). The distance $d(\mathbf{X}_j, \mathbf{Y}_I)$ between the vector and the partition identified by I is calculated (the second box below the

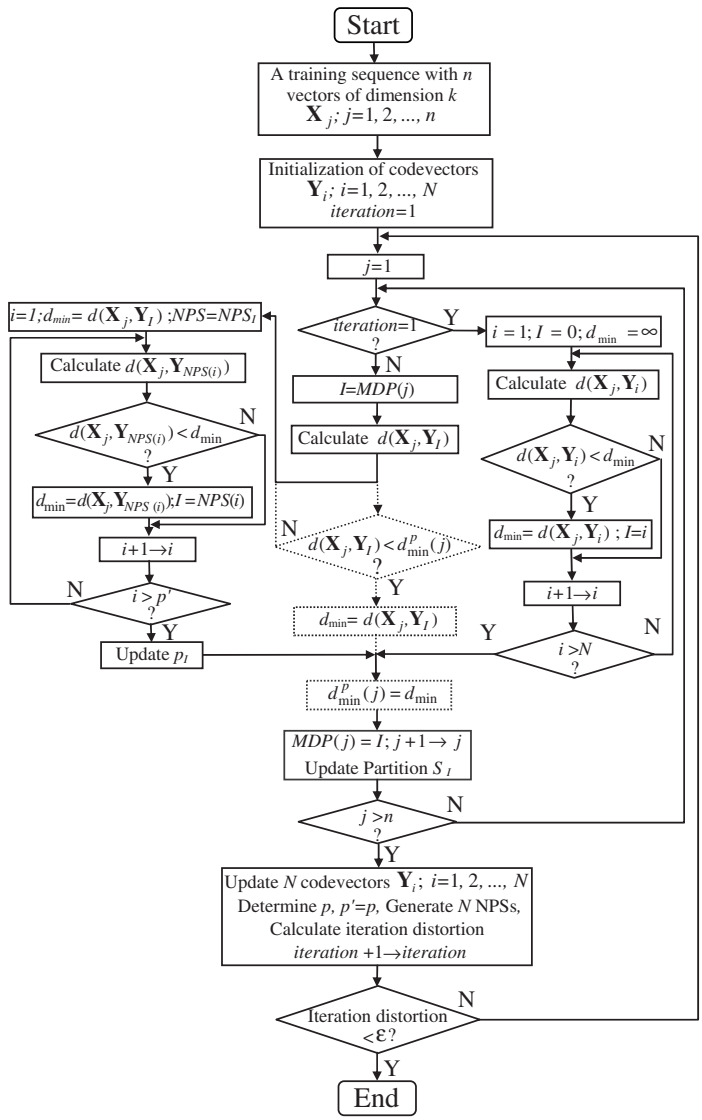


Figure 4.15 Flowchart of the proposed algorithms. Algorithm 1 includes the boxes with solid lines, and algorithm 2 includes the boxes with both solid and dotted lines (reprinted from Ref. 26).

diamond-shaped box named "iteration = 1?" in the center column) and assigned as the current minimum distance d_{min} (in the first box in the left column) because the partition has a high probability of being the MDP of the vector in the current iteration. The I, which carries the index of the MDP of the vector in the previous iteration, is used to identify the nearest partition set $NPS = NPS_I$ for searching for the MDP of the vector. The left

column of the flowchart shows that only p' nearest-neighbor partitions in NPS are searched to find the MDP for the vector $\mathbf{X}_j$. The distance $d(\mathbf{X}_j, \mathbf{Y}_{NPS(i)})(i = 1, 2, \ldots, p')$ between the vector and each of the nearest-neighbor partitions in NPS is calculated. The current minimum distance $d_{\min}$ and the index identity I are updated if the distance between the vector and the partition is smaller than the current minimum distance $d_{\min}$. This process repeats until all p' nearest partitions in the NPS are searched. Before going to the low portion of the flow chart, update p_I, the maximum order number of the MDP found in NPS for determining p at the end of the iteration.

The proposed algorithm also attempts to reduce the computation time of $d(\mathbf{X}_j, \mathbf{Y}_I)$ using partial distance.[24] Compared to the GLA, the additional memory required by Algorithm 1 is $N \times p'$, which is used for storing the lookup table of the p' nearest neighbors for each of the N NPSs, where p' is the size of the NPSs, and N is the codebook size. The TIE-based fast search algorithms[9 16] require $2N^2$ additional memory. The search algorithms in Refs. 18 and 22 require $4(n + N)$ and $(N + N/4 + N/16)$ additional memory, respectively, where n is the total number of vectors in a training sequence.

4.4.5.2 Algorithm 2

Section 4.3 describes a fast codebook training method that slightly modifies the GLA such that a training vector does not require a full-search to find the MDP if its distance to the partition is improved in the current iteration compared to that of the previous iteration. Over half of the total vectors in a training sequence can be directly placed in the same partition as in the previous iteration as the MDP of the current iteration. The codebook generated using this method might not be identical to the codebook generated by GLA, although it is as good as the GLA codebook. In Section 4.3, a full search is still required to find the MDP for a training vector if its distance to the previous MDP is not improved in the current iteration. This full-search can be replaced with the NPS search method proposed in this section to further accelerate the training process. Algorithm 2 is the combination of the two methods proposed in Section 4.3 and this subsection. A codebook generated using Algorithm 2 will not be identical to that generated using the GLA due to the use of the method proposed in Section 4.3.

The codebook training process of Algorithm 2 is shown in Fig. 4.15 (the boxes with solid and dotted lines). The operation of the first iteration is the same as that of Algorithm 1 except that the minimum distance $d_{\min}$ of each vector $\mathbf{X}_j$ needs to be stored in $d^p_{\min}(j)$ (the dotted-line box in the lower portion of the flowchart), which will be used in the next iteration for determining whether the distance between the training vector and the MDP is improved.

After the first iteration, for each vector $\mathbf{X}_j(j = 1, 2, \ldots, n)$ in the training sequence, the distance $d(\mathbf{X}_j, \mathbf{Y}_I)$ between the vector and the partition that was the MDP in the previous iteration, identified by $I = MDP(j)$, is calculated

first. The same partition as in the previous iteration is assigned to the vector as the MDP if the distance is improved, i.e., $d(\mathbf{X}_j, \mathbf{Y}_I) < d^p_{\min}(j)$, where $d^p_{\min}(j)$ is the minimum distance between the vector and the partition in the previous iteration. No search is applied to the vector, as shown in the center column of the flowchart (through the dotted-line diamond-shape box and the dotted-line rectangular box to the lower portion of the flowchart). Otherwise, a search process within the NPS of the previous MDP is applied to find the MDP for the vector (going to the left column of the flowchart). The search process for MDP within the NPS is the same as described for Algorithm 1.

In addition to the memory requirements of Algorithm 1 of $N \times p'$ for storing the lookup table of the p' nearest neighbors for each of the N NPSs, the additional memory requirement of Algorithm 2 is the n for storing n minimum distances $d^p_{\min}(j)$ of the previous iteration, where n is the total number of vectors in a training sequence.

4.4.6 Experimental results

Two hyperspectral AVIRIS datasets used in Section 4.3 were tested. The experiments were performed using the same Sun Fire 280R workstation with a 900-MHz UltraSPARC-III+ CPU. The PSNR defined by Eq. (4.4) was used to measure the quality of codebooks and compression performance.

The iteration distortion threshold ε was set to 0.0001 in the experiments to attain good codebook fidelity. The codebook training and compression results (compression ratio, number of iterations, computation time, and PSNR/SNR) of the two test datasets using the GLA with codebook sizes $N = 16$ to 2048 appear in Tables 4.1 and 4.2. Figure 4.16 shows the improvement in computation time of the two fast algorithms compared to the GLA for the two datasets. The improvement in computation time of the fast algorithm described in Section 4.3 to the GLA is also shown in the figure for the sake of comparison.

It can be seen that Algorithm 1 improved the computation time of codebook training by factors from 6.6 to 50.7 for the Jasper Ridge dataset when the codebook size is from $N = 16$ to 2048, and factors from 5.8 to 70.4 for the Cuprite dataset. The time used to create the NPSs has been included. Algorithm 1 produces exactly the same codebook as that produced by the GLA because the NPS together with the MDP of the previous iteration contains the best-match partition found by the full-search process. The MDP found by Algorithm 1 is identical to that found by the GLA. The larger the codebook size N is, the greater the improvement in computation time. This is because the size of the NPS, which is sought by the MDP, represents a small percent of the total partitions when the codebook size N is large.

It can also be seen from Fig. 4.16 that the time improvement stops increasing at a certain codebook size ($N = 1024$ for the Jasper Ridge dataset,

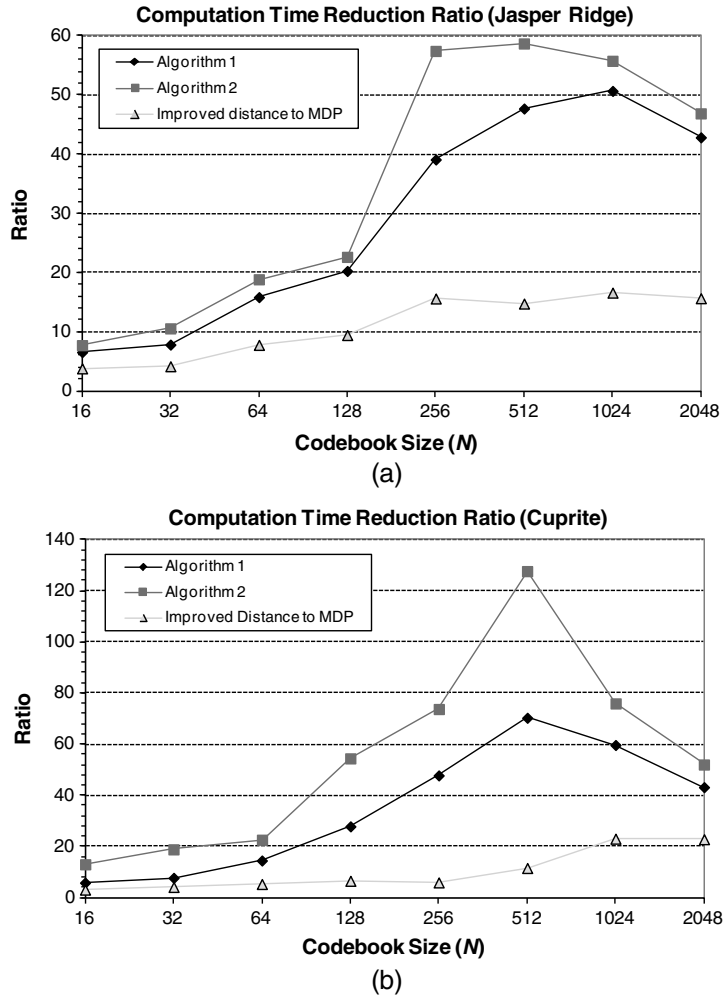


Figure 4.16 Improvement of codebook training time of the NPS-based Algorithms 1 and 2 for the (a) Jasper Ridge dataset and (b) Cuprite dataset; and a comparison to the fast search algorithm based on improved distance to MDP (reprinted from Ref. 26).

and $N = 512$ for the Cuprite dataset) with the increase in the codebook size N. This is because at that codebook size the computation time used to prepare the N NPSs starts to affect the overall computation time. In training a codebook of that size, the computation time saved by searching in the small NPS is significantly consumed by the overhead of preparation of the N NPSs. For instance, when the codebook size is $N = 512$, a total of $N(N \quad 1)/2 = 130{,}816$ vector distances need to be calculated at the end of each iteration in order to sort the neighbor partitions for each of the N partitions. Meanwhile, the total number of calculations of vector distances required

to find the MDP for all the training vectors in the test datasets of size $n = 256 \times 256 = 65{,}536$ is $n(p' + 1)/f_{partial\text{-}distance} = 65{,}536 \times (24 + 1) / 2.5 = 655{,}360$ (where an average NPS size over all iterations $p' = 24$ and a partial distance factor $f_{partial\text{-}distance} = 2.5$ are used). The number of calculations of vector distances used for preparation of N NPSs represents 17% of the total number of calculations of vector distances of the codebook training process. The percentage goes up to 70% when the codebook size $N = 2048$. In general, the time overhead for preparation of the N NPSs starts to significantly affect the overall computation time when the codebook size goes above 1% of the training set size n.

Algorithm 1 greatly outperforms the fast search algorithm that is based on the improved distance to MDP described in Section 4.3. Not only does it generate a codebook exactly the same as the GLA, but it also attains much more improvement in computation time. It is faster than the method described in Section 4.3 by factors from 1.7 to 3.2 for the Jasper Ridge data and by factors from 1.9 to 8.0 for the Cuprite data.

Algorithm 2 combines the method described in Section 4.3 and the NPS-based method. A codebook generated using the fast search method might not be identical to that generated using the GLA when the same training set is used to train a same-size codebook. The improvement in computation time of codebook training by Algorithm 2 is superior to Algorithm 1 for all codebook sizes. Speed improvement factors from 7.7 to 58.7 were attained for the Jasper Ridge data when the codebook size was from $N = 16$ to 2048, and factors from 13.0 to 127.7 were attained for the Cuprite data.

Codebooks generated using Algorithm 2 are as effective as those generated using the GLA when they are used to compress the datasets. Figure 4.17 shows the differences of PSNRs attained using codebooks generated by the GLA and those generated by Algorithm 2 for the two datasets. The differences in the PSNRs are within ± 0.1 dB, except $N = 512$ for the Cuprite dataset.

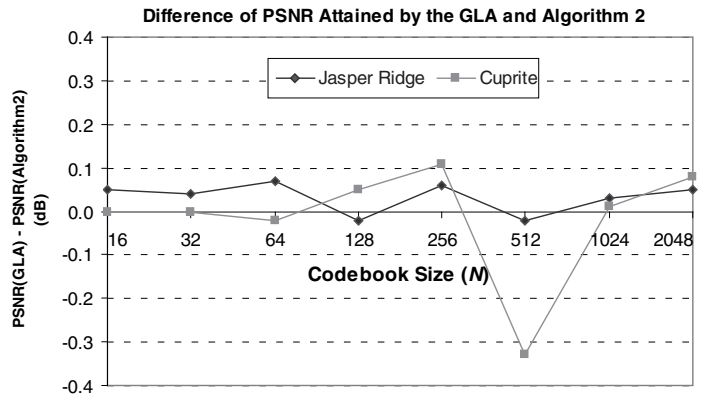


Figure 4.17 Difference of PSNR attained by GLA and Algorithm 2 (reprinted from Ref. 26).

A negative difference implies a slight improvement of PSNR. When the codebook size is $N = 512$ for the Cuprite dataset, the PSNR is better by 0.33 dB using Algorithm 2. The PSNR difference was caused by a disagreement in the MDP found by the GLA and by the method described in Section 4.3.

It is possible that the partition that was the MDP of a training vector in the previous iteration may not be the closest partition of the vector in the current iteration, even though the distance between the vector and the partition is improved in the current iteration. If this is the case, the training vector is forced to stay in the same MDP at the current iteration, whereas the GLA will not place the training vector in the same MDP of the previous iteration if it is not the closest. This forced assignment of MDP can cause some training vectors to be classified suboptimally in an iteration process, introduce certain iteration errors compared to the GLA, and ultimately take more iteration loops to reach the predefined threshold. The experimental results shown in Section 4.3 (Tables 4.1–4.4) have indicated that the search method in Section 4.3 took 1–2 more iteration loops on average than the GLA in training a codebook. Because the fast search method in Section 4.3 takes more iteration loops to train a codebook, it is possible that the overall training error of a codebook trained using this method is slightly smaller than that using the GLA when they exit the iteration loop by comparing the overall training error with the iteration threshold ε. For the case of the codebook of size $N = 512$ trained on the Cuprite dataset, the GLA took 52 iterations to train the codebook and exited the iteration loop with the overall training error of 0.0000914 ($<\varepsilon = 0.0001$), whereas Algorithm 2 took 55 iterations to train the codebook and exited the iteration loop with the overall training error of 0.0000681. From this point of view, the codebook trained using Algorithm 2 is slightly more accurate to span the training vectors than that using the GLA.

The improvement in computation time of the fast search method is independent of the number of dimensions of the vectors, as it achieves the improvement by reducing the search space from searching all N partitions to the small NPS of the previous MDP of a training vector plus the single previous MDP. A simplified theoretical speedup factor of the fast search method can be expressed as $f = N/(p_{\text{average}} + 1)$, where p_{average} is the average size of the NPSs of all the iteration loops. In reality, the speedup factor also depends on the codebook size N due to the overhead used for preparing the NPSs as discussed above. Experiments with a small vector size of 32 were carried out for the Cuprite dataset. The experimental results gave similar conclusions.

4.4.7 Comparison with published fast search methods

It is worth the effort to compare the improvement in computation time achieved by the proposed algorithms in this book with reported fast codevector search methods in the literature. It would be difficult to judge

the performance speedup of published fast methods by comparing their computation time reported, since the sizes of training sets, the dimensions of vectors, size of codebooks trained, and the computation platform used were different. That is why most of the published papers reviewed in Section 4.2 evaluated the improvement in computation time of the developed fast methods by comparing their computation time with that of the GLA, as the GLA is well known and easy to implement. A reader can easily judge the performance of different fast methods by comparing their ratios of computation time improvement to the GLA. This approach is adopted here to evaluate the performance of the proposed algorithms in this book.

In Kaukoranata et al.,[23] the developed fast method, which is referred to as grouping, together with three earlier published fast methods—partial distance search (PDS),[24] triangle inequality elimination (TIE),[9] and mean-distance-order partial search (MPS)[13]—were compared with the GLA for the codebook size of $M = 256$. Three widely used images—Bridge, Miss America, and House—were used. In order to evaluate the computation time reduction of the proposed algorithms in this book, Table 4.5 was created here based on the information in the article.[23] It shows the time reduction compared to the GLA obtained by each of the published methods. It can be seen that the time reduction obtained by a single published fast method is between 1.9 and 20.3 for the three test images when the codebook size is 256. Because the grouping was applied to PDS, TIE, and MPS in Ref. 23 to further reduce the computation time, Table 4.5 also shows the overall time reduction obtained by combining

Table 4.5 Time reduction compared to the GLA obtained by four published fast algorithms and their combination (codebook size $M = 256$).

		A method Alone		With Grouping	
Image	**Method**	**Running Time**	**Time Reduction**	**Running Time**	**Overall Time Reduction**
Bridge	GLA	127.6			
	Grouping	46.1	2.8		
	PDS	33.4	3.8	13.0	9.8
	TIE	21.4	6.0	13.5	9.5
	MPS	12.4	10.3	4.8	26.6
Miss America	GLA	1344.5			
	Grouping	336.3	4.0		
	PDS	311.1	4.3	75.8	17.7
	TIE	135.2	9.9	44.6	30.1
	MPS	97.3	13.8	21.5	62.5
House	GLA	874.8			
	Grouping	138.8	6.3		
	PDS	460.7	1.9	85.0	10.3
	TIE	43.2	20.3	28.8	30.4
	MPS	55.2	15.8	15.5	56.4

grouping with each of the earlier published fast methods. The overall time reduction is between 9.5 and 62.5 for the three test images when the codebook size is 256. The time reductions obtained by Algorithm 1 proposed in this book are 39.1 and 47.8 for Jasper Ridge and Cuprite datasets, respectively, when the codebook size is 256; and the time reductions obtained by Algorithm 2 proposed in this book are 57.4 and 73.8 (the sizes are much larger than that obtained by a single published fast method and comparable to the overall time reduction obtained by their combinations).

4.5 3D VQ Compression Using Spectral-Feature-Based Binary Code

Two fast VQ search algorithms have been discussed in Sections 4.3 and 4.4. These fast VQ search algorithms significantly reduce the search space of the conventional full-search GLA in the codebook training process to accelerate the processing.

The remaining sections of this chapter describe fast vector quantization algorithms that exploit other effective means to significantly reduce the computational complexity of the conventional vector quantization algorithms for bringing VQ algorithms to practical uses in the compression of satellite images, especially hyperspectral datacubes.

This section introduces an effective binary coding method to map the spectra of a hyperspectral datacube to be compressed into very simple and short binary codes. The heavy VQ codebook training and codevector matching are then performed on these binary codes. This method speeds the training and matching processing.

A spectral-feature-based binary code (SFBBC) was developed to represent the vectors $\{\mathbf{X}_i\}\, i = 1, 2, \ldots, N_r \times N_c$ of a hyperspectral datacube.[36] The SFBBC-coded vectors allow the distance $d(\mathbf{X}_i, \mathbf{Y}_l)$ between a training vector $\mathbf{X}_i$ and a codevector $\mathbf{Y}_l$ to be calculated using the Hamming distance[37] rather than the Euclidean distance. The Hamming distance is a sum of binary bit-wise exclusive-or operations and is a much-faster computation than the Euclidean distance. Because the calculation of distance $d(\mathbf{X}_i, \mathbf{Y}_l)$ is the most frequent operation in the VQ process, this speeds up processing.

4.5.1 Spectral-feature-based binary coding

When the spectrum $\mathbf{X}_{i,j} = \{x_{i,j}(\lambda)(\lambda = 1, 2, \ldots, N_b)\}$, corresponding to a ground sample at location (i, j) in a datacube, is defined as a vector, in fact, the codebook generation and nearest-codevector searching for each of vectors are operations of spectral feature matching. If a simple and efficient way to express a spectrum can be found, the operation of spectrum matching can be performed rapidly.

Qian et al. extended the binary spectral encoding algorithm[38] and developed a SFBBC[36] that can effectively express a spectrum using binary code. The SFBBC encodes an N_b-band spectrum $\{x_{i,j}(\lambda)(\lambda = 1, 2, \ldots, N_b)\}$ into three binary codevectors:

1. N_b-bit amplitude binary codevector $\mathbf{X}^a_{i,j}$,
2. $(N_b \quad 2)$-bit slope binary codevector $\mathbf{X}^s_{i,j}$, and
3. N_b-bit mean deviation binary codevector $\mathbf{X}^{MD}_{i,j}$.

The three binary codevectors are then concatenated to form a single $(3N_b \quad 2)$-bit SFBBC code:

$$\widetilde{\mathbf{X}}_{i,j} = \mathbf{X}^a_{i,j}\mathbf{X}^s_{i,j}\mathbf{X}^{MD}_{i,j} \quad (i = 1, 2, \ldots, N_r; j = 1, 2, \ldots, N_c). \tag{4.10}$$

The amplitude binary codevector $\mathbf{X}^a_{i,j}$ expresses variation of amplitude of each spectral element about its mean and is constructed from

$$\begin{gathered}\mathbf{X}^a_{i,j} = \left[X^a_{i,j}(1), X^a_{i,j}(2), \ldots, X^a_{i,j}(N_b)\right]^T, \\ (i = 1, 2, \ldots, N_r; j = 1, 2, \ldots, N_c)\end{gathered} \tag{4.11}$$

$$\mathbf{X}^a_{i,j}(\lambda) = \begin{cases} 1 & [x_{i,j}(\lambda) \quad \mu_{i,j}] \geq 0 \\ 0 & [x_{i,j}(\lambda) \quad \mu_{i,j}] < 0 \end{cases} \lambda = 1, 2, \ldots, N_b, \tag{4.12}$$

where the scalar $\mu_{i,j}$ is defined as the spectral mean of pixel (i, j):

$$\mu_{i,j} = \frac{1}{N_b}\sum_{\lambda \quad 1}^{N_b} x_{i,j}(\lambda) \quad (i = 1, 2, \ldots, N_r; j = 1, 2, \ldots, N_c). \tag{4.13}$$

The slope binary codevector $\mathbf{X}^s_{i,j}$ represents variation of amplitude of each spectral element about its slope related to two adjacent elements and is constructed from

$$\begin{gathered}\mathbf{X}^s_{i,j} = \left[X^s_{i,j}(1), X^s_{i,j}(2), \ldots, X^s_{i,j}(N_b)\right]^T, \\ (i = 1, 2, \ldots, N_r; j = 1, 2, \ldots, N_c)\end{gathered} \tag{4.14}$$

$$\mathbf{X}^s_{i,j}(\lambda) = \begin{cases} 1 & [x_{i,j}(\lambda + 1) \quad x_{i,j}(\lambda \quad 1)] \geq 0 \\ 0 & [x_{i,j}(\lambda + 1) \quad x_{i,j}(\lambda \quad 1)] < 0 \end{cases} \quad \lambda = 2, 3, \ldots, N_b \quad 1. \tag{4.15}$$

The mean deviation binary codevector $\mathbf{X}^{MD}_{i,j}$ describes the variation of amplitude of each spectral element about its mean deviation and is constructed from

$$\begin{gathered}\mathbf{X}^{MD}_{i,j} = \left[X^{MD}_{i,j}(1), X^{MD}_{i,j}(2), \ldots, X^{MD}_{i,j}(N_b)\right]^T, \\ (i = 1, 2, \ldots, N_r; j = 1, 2, \ldots, N_c)\end{gathered} \tag{4.16}$$

$$\mathbf{X}_{i,j}^{MD}(\lambda) = \begin{cases} 1 & [x_{i,j}(\lambda) \quad \mu_{i,j}] \geq MD_{i,j} \\ 0 & [x_{i,j}(\lambda) \quad \mu_{i,j}] < MD_{i,j} \end{cases} \qquad \lambda = 1, 2, \ldots, N_b, \quad (4.17)$$

where the scalar $MD_{i,j}$ is defined as the spectral mean deviation of pixel (i, j):

$$MD_{i,j} = \frac{1}{N_b} \sum_{\lambda \ 1}^{N_b} |x_{i,j}(\lambda) \quad \mu_{i,j}| \quad (i = 1, 2, \ldots, N_r; j = 1, 2, \ldots, N_c). \quad (4.18)$$

With the spectral vectors being expressed using the SFBBC code, the distance between two vectors $\mathbf{X}_{i,j}$ and $\mathbf{Y}_{m,n}$ or the similarity measure for determining signature matches between them can be implemented by calculating the Hamming distance[38] between the two SFBBC-coded vectors $\widetilde{\mathbf{X}}_{i,j}$ and $\widetilde{\mathbf{Y}}_{m,n}$, which is computed as follows:

$$D_h(\widetilde{\mathbf{X}}_{i,j}, \widetilde{\mathbf{Y}}_{m,n}) = \sum_{\lambda \ 1}^{3N_b \ 2} \widetilde{\mathbf{X}}_{i,j}(\lambda) \mathrm{XOR} \widetilde{\mathbf{Y}}_{m,n}(\lambda). \quad (4.19)$$

This is just a sum of binary bit-wise exclusive-or operations. Because perfect matches rarely occur with real data, the two vectors are considered identical if the distance D_h between them is smaller than an acceptable threshold distance d.

4.5.2 Fast 3D VQ using the SFBBC

In fact, a SFBBC code can be viewed as a dimension-reduction code. Let us still use the AVIRIS datacube as an example. In VQ compression of 3D datacubes, a raw vector has $N_b = 224$ elements, and takes 224 units of 16-bit long memory, while a binary vector uses $\widetilde{\mathbf{X}}_{i,j}$ just $3N_b \quad 2 = 670$ bits. It takes only 42 units to put them in a 16-bit-long memory. Therefore, the dimension of the vectors is reduced from 224 to 42. A more significant reduction in computational complexity is the distortion measure between two vectors. For the raw vectors, the distortion measure uses Eq. (4.2), where $N_b = 224$ products and $2N_b \quad 1 = 447$ additions are required. For the SFBBC binary vectors, however, the distortion measure uses Eq. (4.19), which measures the Hamming distance between two binary vectors, and requires only $(3N_b \quad 2)/16 = 42$ exclusive-or logical operations. The latter is much faster than the former.

One can now discuss codebook training with the SFBBC binary vectors. Each vector of training sequence $\mathbf{X}_{i,j}(i = 1, 2, \ldots, N_r; j = 1, 2, \ldots, N_c)$ is encoded to a binary codevector $\widetilde{\mathbf{X}}_{i,j}$. In codebook training, the MDP for all vectors of the training sequence is found using SFBBC binary vectors instead of using raw vectors. For a spectral vector $\mathbf{X}_{i,j}$, if

$$D_h(\widetilde{\mathbf{X}}_{i,j}, \widetilde{\mathbf{Y}}_m) \leq D_h(\widetilde{\mathbf{X}}_{i,j}, \widetilde{\mathbf{Y}}_l) \quad (4.20)$$

for $l = 1, 2, \ldots, N$, then $\mathbf{X}_{i,j}$ is assigned to partition S_m. Here, $\widetilde{\mathbf{Y}}_m$ and $\widetilde{\mathbf{Y}}_l$ are SFBBC binary vectors of codevectors $\mathbf{Y}_m$ and $\mathbf{Y}_l$. The computational complexity to find the minimum distance partition S_m for one vector is now reduced from $N \times N_b$ products plus $N \times (2N_b \quad 1)$ additions to $N \times \left(\frac{3N_b \quad 2}{len}\right)$ exclusive-or operations (here, *len* is the word-length of memory, usually 16 for AVIRIS data).

In the LBG algorithm, during codebook training for each vector in the training sequence, a search is performed over all N codevectors to find the MDP; it is very time consuming. To overcome this problem, it is suggested to use the mean $\mu_{i,j}$ of the current vector as an auxiliary parameter to decrease the range of search. It has been shown in the previous discussion that the mean of a vector plays an important role in SFBBC coding. It has a direct influence on the binary vector $\mathbf{X}^a_{i,j}$, which expresses the variation of amplitude of the vector, and on the binary vector $\mathbf{X}^{MD}_{i,j}$. But these binary vectors do not explicitly give the value of the mean. The probability of a good match between two vectors whose means are similar is much higher than that between two vectors whose means are very different. That is to say that it is not worthwhile to match the codevectors whose means are far from that of the current vector. Only the codevectors whose means are close to that of the current vector are searched to find the minimum distance partition. Thus, the search range can be greatly decreased. To implement this idea, the codevectors need to be sorted according to the values of their means. When searching for the nearest codevector for an input vector, the codevector whose mean is closest to that of the current vector is located first and then taken as the center of the searching range N_o; the nearest codevector is then sought in the range $(N_o \quad r_1, N_o + r_2)$, where the r_1 and r_2 can be determined by the tolerance of mean bias.

In order to increase the accuracy of vector matching with SFBBC-coded vectors, a scheme of two-level searching is adopted. In the first level, several candidate codevectors are found by using SFBBC vectors and the methodology discussed earlier. In the second level, the best one is selected by using the mean square criteria from these candidates. Although it takes a little bit more time to do this, it improves the accuracy of the match. The total time for two-level searching is still much less than that for the LBG and conventional VQ.

4.5.3 Experimental results of the SFBBC-based VQ compression algorithm

Hyperspectral data acquired using CASI were tested. The test datacube has 144 spectral bands within the visible and near-IR region, and each band has 150 rows and 200 columns. The data has been calibrated and is stored with 16-bit long precision, although the data occupies only about half of the stored dynamic range. The results obtained by both conventional VQ (i.e., LBG algorithm)[2] and the SFBBC-based algorithm are given to evaluate their performance. The experiments

Table 4.6 Comparison of codebook generation using original vectors and SFBBC vectors.

		Codebook Size			
Vector Type	**Results**	**256**	**512**	**1024**	**4096**
Original vectors	Processing time (sec)	107.59	251.18	419.72	1436.72
	Distortion Threshold ε	0.0013	0.0014	0.0011	0.0013
	Number of iterations	12	10	10	8
SFBBC vectors	Processing time (sec)	2.68	5.29	11.07	46.65
	Distortion Threshold ε	0.0006	0.0040	0.0015	0.0012
	Number of iterations	6	5	7	11
	Improvement in processing time	40.2	47.5	37.9	30.8

reported in Ref. 36 have been redone with a more-recent computational platform, and the processing times reported in that source have been updated.

The results of codebook generation using the LBG algorithm on the original spectral vectors and the SFBBC vectors are listed in Table 4.6. It can be seen from the table that generating an identical-size codebook is on average 39 times faster with the SFBBC vectors than with the original vectors. In addition, it shows that the number of training iterations with the SFBBC vectors are usually smaller than with the original vectors, when their distortion thresholds ε are very close.

The compression ratio in VQ is determined by the codebook size N if the dimension of the vectors is fixed [it has been defined in Eq. (4.1)]. For the CASI datacube, the dimension of the vectors is $N_b = 144$, and the word-length $L = 16$ bits. The PNSR defined in Eq. (4.4) is used to measure the fidelity of the reconstructed data from the compression data with respect to its original data. The value of *peak signal* in Eq. (4.4) used in the experiments is the maximum value measured from the raw data (29,841 in this case) instead of 65,535, the maximum value of a 16-bit unsigned word.

The compression performance of the conventional LBG algorithm and the SFBBC-based fast algorithm with four different-size codebooks is listed in Table 4.7. The processing time listed in the table includes both coding and

Table 4.7 Comparison of VQ compression (including codebook training and coding) using LBG algorithm and the SFBBC-based fast algorithm.

		Codebook Size			
Vector Type	**Results**	**256**	**512**	**1024**	**4096**
LBG	Compression ratio	288	256	230	192
	Processing time (sec)	10.23	20.45	40.91	177.27
	PSNR (dB)	43.03	44.14	45.43	47.99
SFBBC	Compression ratio	288	256	230	192
	Processing time (sec)	0.34	0.68	1.16	4.16
	PSNR (dB)	42.28	43.51	44.44	46.83
	Improvement in processing time	30.0	30.0	35.3	42.6

decoding. A LBG algorithm that uses full-search codevector matching requires ~177 s to compress a 3D datacube of 150 rows $\times$ 200 columns $\times$ 144 bands when the codebook size is $N = 4096$. Using the SFBBC-based algorithm, the processing time is 30–42 times faster than that using conventional LBG, whereas the reduction in the fidelity of reconstructed data is only between 0.9–1.9 dB.

4.6 Correlation Vector Quantization

As illustrated in Fig. 4.1, an index map is formed after VQ compression. An index located at (x, y) of the index map is the address of the codevector in the codebook that best represents the spectrum of the ground sample located at the same location in the scene of the datacube. Because of spatial correlation, the adjacent indices in the index map may be identical, especially in regions of relatively small variation between ground samples. The spatial correlation could be exploited by coding the index map using a conventional lossless technique, such as differential pulse code modulation (DPCM) and entropy coding. However, this method does not offer any benefit to processing speed. A correlation vector quantization (CVQ) method was proposed by Qian et al.[39] that performs VQ compression on 3D hyperspectral data and index map coding simultaneously to remove the correlation in both the spatial and spectral domains, with the additional benefit of reduced computational complexity.

4.6.1 Process of CVQ

A movable window that covers a block of 2 $\times$ 2 adjacent vectors (i.e., spectra) in a datacube is used in CVQ. The vector quantizer scans over the scene of the datacube from left to right, top to bottom. Figure 4.18 shows the upper-left corner of an index map. This figure is used here to describe the principle of

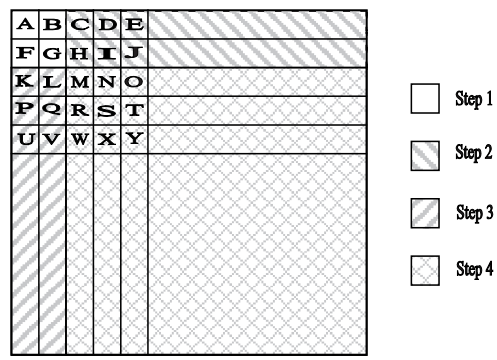


Figure 4.18 The upper-left corner of an index map, the area in which vectors are coded by each step.

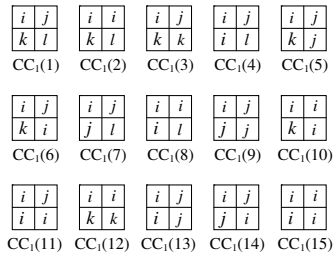


Figure 4.19 Step 1 correlation codevectors.

the CVQ. The capital letters represent vectors. The coding process of CVQ is composed of four steps: (1) initialize the quantizer by starting in the block at the top-left corner; (2) extend quantization across the first two rows of the scene; (3) similarly extend the process down the first two columns; and (4) recursively quantize the remaining imagery. The vectors that are coded by each step are indicated by different shades in the figure.

In step 1, the window is located at the upper-left corner of the index map, covering vectors **A**, **B**, **F**, and **G**, as shown in Fig. 4.18. The index of the codevector for **A** is obtained by searching through the codebook to find the best-match codevector. The indices for **B**, **F**, and **G** are obtained by directly comparing them with **A** instead of searching through the codebook. The same index for **A** is assigned to **B** if the distance measure between **A** and **B** is smaller than a given threshold d, thereby avoiding a search of the entire codebook to encode **B**. If the distance measure exceeds the threshold, the index for **B** is obtained by searching for the best codevector in the codebook. The same procedure is applied for **F** and **G**.

There are 15 possible combinations of the four indices in step 1; they are expressed by the 15 correlation codevectors (CCs) shown in Fig. 4.19. In the figure, the lowercase letters i, j, k, and l denote different indices. Table 4.8 gives the meaning of each CC and the number of bits required to encode the four vectors. $\log_2 N$ is the number of bits required to express an index of codevector in a codebook of size N, and four bits are used to express a CC.

Table 4.8 Symbolism of CCs in step 1 and the number of bits used to code four vectors in the window.

Correlation Codevector	Symbolism	Bits Required to Code 4 Vectors in the Window
$CC_1(1)$	All 4 indices are different	$4\log_2 N + 4$
$CC_1(2)$ $CC_1(7)$	2 indices are identical	$3\log_2 N + 4$
$CC_1(8)$ $CC_1(11)$	3 indices are identical	$2\log_2 N + 4$
$CC_1(12)$ $CC_1(14)$	2 pairs indices are identical	$2\log_2 N + 4$
$CC_1(15)$	All 4 indices are identical	$\log_2 N + 4$

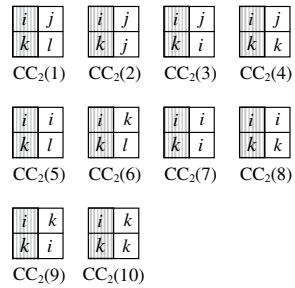


Figure 4.20 Step-2 correlation codevectors.

In step 2, the window covers the first two rows (but not the first column) of the index map and moves over the index map from left to right, for example, blocks **BCGH**, **CDHI**, **DEIJ**, etc. Only two vectors in each block need to be coded. For example, in block **BCGH**, only **C** and **H** need to be coded because **B** and **G** were already coded in step 1. The index of the coded vector is assigned to the new vector if the distance measure between the new vector and the coded vector is smaller than the threshold d. There are ten possible states for the two new indices; ten CCs are used to represent them, as shown in Fig. 4.20. The known indices are shaded. The symbolism of ten CCs and the number of bits required to code the two new vectors are listed in Table 4.9.

Step 3 is similar to step 2 except that the window is moved vertically. That is, the window covers the first two columns but not in the first row of the index map, for example, blocks **FGKL**, **KLPQ**, **PQUV**, etc. There are also ten CCs.

In step 4, the window covers neither the first row nor the first column in the index map, i.e., blocks **GHLM**, **HIMN**, **LMQR**, etc., as shown in Fig. 4.18. Only one vector in the window needs to be coded when the window scans to a new location. For example, when the window is located

Table 4.9 Symbolism of CCs in step 2, and the number of bits used to code two new vectors in the window.

Correlation codevector	Symbolism	Bits Required to Code Two New Vectors in the Window
$CC_2(1)$	2 new indices are different from 2 known indices	$2\log_2 N + 4$
$CC_2(2)$	2 new indices are identical, but different from 2 known indices	$\log_2 N + 4$
$CC_2(3)$ $CC_2(6)$	1 new index is identical to 1 of the 2 known indices	$\log_2 N + 4$
$CC_2(7)$ $CC_2(10)$	2 new indices are identical to 2 known indices	4

at block **GHLM**, only **M** needs to be coded because the vectors **G**, **H**, and **L** have already been coded in previous steps, and their indices are known. Three distance measures between the new vector and each coded vector are computed, and the smallest of the three distance measures is compared with the threshold d. If it is less than d, the index of the coded vector, which corresponds to the smallest distance measure, is assigned to the new vector. Otherwise, the index of the new vector is obtained by searching the codebook. There are only four possible states for the new index: identical to one of the three known indices or different from all of them; thus, four CCs are used to express them. Two bits are sufficient to code the new vector if the new index is identical to one of the three known indices. $\log_2 N+2$ bits are required if the new index is different from all three known indices.

The CVQ performs VQ compression and index map coding simultaneously. No additional index-map-coding step is required. It is faster than the conventional VQ algorithm because the codebook is not searched in many cases. The threshold d is the only parameter of the CVQ. If it is set to be too small, the process reverts to the conventional VQ algorithm. Similarly, if there is no spatial correlation, there is no additional compression gain.

4.6.2 Performance of CVQ

If the spatial size of a datacube is N_r rows by N_c columns, the number of scans in steps 1 through 4 are 1, N_c 2, N_r 2, and (N_c 2)(N_r 2), respectively. Step 4 is the most frequent one. This section discusses only step 4 for simplicity. If the Euclidean distance defined by Eq. (4.2) is used as the distance measure, the computation of one distance measure requires k products plus k 1 additions and k subtractions, where the vector dimension k is equal to the number of spectral bands N_b.

For simplicity, only products are considered when assessing the computation complexity. In CVQ, the quantizer first computes three vector distances between the new vector and each coded vector in the window. $3N_b$ products are required to encode a vector if the distance meets the threshold d. Otherwise, the quantizer further searches through the N codevectors in the codebook to find the closest one, and a total of (3 + N)N_b products are required. Let it be supposed that α is the probability that a new index is different from all three known indices; 1 α is the probability that a new index is identical to one of the three known indices. The factor of improvement in coding time (FICT) of the CVQ can be estimated as

$$FICT = \frac{CT_{VQ}}{CTC_{VQ}} = \frac{NN_b}{(1 \quad \alpha)3N_b + \alpha(3+N)N_b} = \frac{N}{3+\alpha N} \approx \frac{1}{\alpha}, \qquad (4.21)$$

where N is the codebook size. It is usually 256 or larger; αN is much greater than 3.

The compression ratio of CVQ is calculated as

$$Cr_{(CVQ)} = \frac{N_r N_c N_b L}{(\alpha \log_2 N + 2) N_r N_c + N N_b L}, \tag{4.22}$$

where L is the word-length of the datacube in bits. The factor of improvement in compression ratio (FICR) can be estimated as

$$FICR = \frac{Cr_{(CVQ)}}{Cr_{(VQ)}} = \frac{\log_2 N + c}{\alpha \log_2 N + 2 + c}, \tag{4.23}$$

where

$$c = \frac{NNbL}{NrNc} \tag{4.24}$$

is the overhead for transmission of the codebook.

As an example, if the test data in Section 4.5 is compressed using CVQ with a codebook of size $N = 256$, and assuming the frequency of occurrence of each correlation codevector in step 4 is identical (i.e., $\alpha = 0.25$), the *FICT* and *FICR* can be estimated using Eq. (4.21) and (4.23). They are $FICT = 3.8$ and $FICR = 1.8$, respectively. The greater the α value is, the smaller the *FICT* and *FICR*. In the limit, when $\alpha = 1$, no new vector in step 4 can be replaced by its adjacent vectors, and no improvement in compression ratio is made: the compression ratio obtained by CVQ is slightly worse than that obtained by conventional VQ. This is caused by the use of 2 additional bits to code a correlation codevector.

4.7 Training a New Codebook for a Dataset to Be Compressed

A challenge to VQ compression algorithms for satellite hyperspectral data in terms of operational use is that they require large computational resources, particularly for the codebook training phase. Because the size of hyperspectral datacubes can be hundreds of times larger than those for traditional remote sensing, the processing time required to train a codebook or to encode a datacube using the codebook could also be hundreds of times larger. In 2D VQ compression applications, the problem of training time is reduced by training a codebook only once and henceforth applying it repeatedly to all subsequent images to be compressed. This codebook is a universal codebook or dictionary. It is normally trained from a large representative training set that spans as many scene types as possible. This approach may not work for satellite data compression because a training set cannot span many types of satellite datasets. The so-called universal codebook trained from this training set cannot compress a satellite dataset well.

In satellite remote sensing, it is generally very difficult to obtain a universal codebook that spans many datasets to the required degree of

accuracy, partly because of the high potential variability of the acquired datacubes in terms of their location, spectral and spatial resolution, illumination, viewing angle, atmospheric effects, season, spectral band configuration, SNR of the instruments, etc., and partly because of the high reconstruction fidelity that is required of the data in its downstream use. For these reasons, the preferred strategy is to train a new codebook for a dataset to be compressed by using the dataset itself as the training set and to apply the trained codebook to compress the dataset. This gets rid of the difficulty of constituting a representative training set. The codebook trained in this way is more effective. Because the codebook is trained by using the dataset itself as the training set and is applied to compress the dataset only (not universal), it needs to be transmitted to the decoder side together with the index map as the compressed data in order for the decoder to reconstruct the compressed data. Figure 4.21 illustrates the scheme of the VQ data-compression system for satellite data.

Because the codebook is part of the compressed data, the compression ratio defined by Eq. (4.1) for the conventional VQ compression algorithms needs to be revised by taking into account the additional bits used to encode the codebook. The revised compression ratio is as follows:

$$C_r = \frac{N_r N_c N_b L}{N_r N_c \log_2 N + N N_b L} = \frac{N_b L}{\log_2 N + \dfrac{N N_b L}{N_r N_c}}, \tag{4.25}$$

where NN_bL/N_rN_c is the overhead for the codebook. By looking at Eq. (4.25) itself, it seems that the compression performance is reduced as the compression ratio is reduced if the codebook size N remains the same. In reality, based on experimental results, this is often not the case. Although

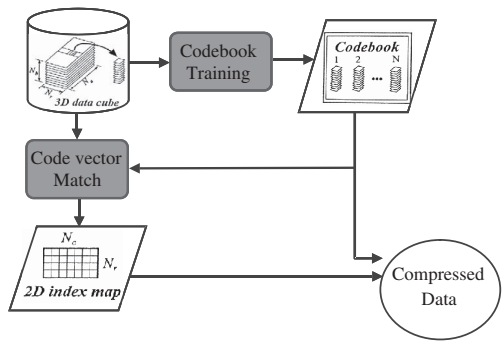


Figure 4.21 Scheme of VQ data compression system for satellite data, where a new codebook is trained for each input dataset and is transferred to the decoder together with the index map.

the compression ratio is reduced due to the overhead of transferring the codebook, the compression fidelity (normally measured by PSNR or SNR) is improved conversely. This is because the codebook trained from the dataset itself is much more accurate than the codebook of the same size trained from the large, so-called representative training set. Thus, a small-size such codebook will achieve the same compression fidelity as a large universal codebook. The small codebook size N trained from the dataset itself compensates for the overhead of the codebook for compression ratio loss. The more-important gain of this strategy is that training a small codebook from a small training set is faster because the VQ training processing time is proportional to the codebook size N and the training size n, as discussed in Section 4.1.

All of the codebooks in the following sections are trained from the datasets to be compressed themselves and are transferred to the decoder as part of compressed data. For the calculation of compression ratio, Eq. (4.25) is used.

4.8 Multiple-Subcodebook Algorithm Using Spectral Index

This section describes a method that uses the information contained in a hyperspectral datacube to be compressed to improve the performance of the VQ compression algorithm.[41 43] A spectral index (SI) image of a datacube to be compressed is generated first. This SI image is then employed to divide the datacube into s subsets, each of which represents a cover type that occurs within the scene of the datacube. This strategy is based on the knowledge that the codebook generation time (CGT) and coding time (CT) are reduced when the codebook size is reduced. Accordingly, while keeping the same total number of codevectors, we generate not only a single codebook but also s smaller codebooks of equal size called subcodebooks from each of the subsets. This VQ compression algorithm is referred to as the multiple-subcodebook algorithm (MSCA).

4.8.1 Spectral indices and scene segmentation

The most-accurate methods for segmenting scene spectra into similar clusters, such as supervised classification and isoclustering, lead to large computational overheads when performed on hyperspectral datacubes, require operator intervention, or both. Therefore, they do not suit the goal of creating a more-efficient algorithm in terms of processing time; a faster solution is required. In the VQ-based hyperspectral data compression techniques presented here, a spectrum corresponding to a ground sample in the scene is treated as a vector. A spectral index is introduced as a means to capture one or more spectral characteristics for scene segmentation of a hyperspectral datacube; it is computed vector by vector, and the resulting SI image is provided as input to a

segmentation procedure. The segmentation is performed rapidly because it operates only on the SI image instead of the entire datacube.

By way of example, the normalized difference vegetation index (NDVI) is chosen for its simplicity and its familiarity to the remote sensing community. The role of NDVI in land-cover classification is well documented in the literature.[40] The data range of NDVI is between −1.0 and 1.0 by definition. Spectra with similar NDVI values often share many spectral similarities, especially for vegetated scenes at visible and NIR wavelengths. For other types of scenes, other indices are preferable. For instance, a water index should be used over ocean scenes. For datacubes involving other wavelength regimes, such as the short-wave infrared (SWIR), a more-complex index or a combination of indices should be used.

If the segmentation regions are chosen such that their members share certain spectral characteristics, each subcodebook will be more likely to represent that region effectively, and the fidelity penalty incurred will be kept small. Four segmentation methods used on the SI image are studied.

4.8.1.1 Manual multithresholding

The multithresholding (MT) method manually segments a SI image into n clusters by simply slicing the SI (e.g., NDVI) value into s intervals of equal width. If a spectrum's SI value lies in the first interval, it is classified to region 1; if it lies in the second interval, it is classified to region 2; and so on. In practice, in order to facilitate coding and maximize the compression ratio, it is preferable to set the number of regions s to a power of 2, such as 8 or 16. The first row of Table 4.10 lists an example for a datacube called *cube*1 (which is described later), where the vegetation cover type dominates (described in

Table 4.10 An example of scene segmentation using the four segmentation methods on the SI image (NDVI) of test datacube *cube*1 when the number of regions $s = 8$. (The data range of SI value and the number of pixels in each region are listed in the table. The size of the SI image is $128 \times 2200 = 281{,}600$.)

Segmentation Method	Region Number							
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
MT	[1.0, 0.15)	[0.15,0)	[0,0.15)	[0.15,0.30)	[0.30,0.45)	[0.45,0.60)	[0.60,0.75)	[0.75,1.0)
	36600	4674	5846	6145	7828	128248	82969	9290
IC	42278	59662	60938	43440	28087	23864	14954	8377
HS	[1.0, 0.27)	[0.27,0.49)	[0.49,0.53)	[0.53,0.56)	[0.56,0.59)	[0.59,0.63)	[0.63,0.69)	[0.69,1.0)
	35206	34424	36772	35756	34808	34794	35418	34242
MHS	[1.0, 0.27)	[0.27,0.06)	[0.06,0.27)	[0.27,0.45)	[0.45,0.54)	[0.54,0.58)	[0.58,0.65)	[0.65,1.0)
	35206	8376	8662	8843	54958	56457	54811	54287

Section 4.8.4). When it is segmented into eight regions, all of the spectra whose NDVI values are less than 0.15 are lumped into a single region (which represents water), and all of the spectra whose NDVI values exceed 0.75 are lumped into another single region. The remaining range $0.15 \sim 0.75$ is divided into six regions with an equal NDVI width of 0.15.

4.8.1.2 Isoclustering

Isoclustering (IC) is carried out on the SI image using the PCI/EASI-PACE routine ISOCLUS in the PCI software tool. The algorithm examines a large number of unknown values in the SI image and divides them into a number of clusters based on natural groupings present in the image. The number of clusters s produced by this algorithm is determined by the nature of the scene. It is often the case that the output s is not equal to a number of power 2. It is necessary to constrain it by adjusting the parameters of the algorithm. Unsupervised classification is an iterative and relatively time-consuming operation. The second row of Table 4.10 gives the segmentation result of this method.

4.8.1.3 Histogram-based segmentation with same-size regions

The size of each region produced by both manual MT and IC is different. Some regions are very large, while others may be very small. Because the MT method segments the SI image in terms of the given intervals of SI value, it does not take into account the population of the regions. If the occurrence frequency of SI value is high in a certain interval, the corresponding region is large. Similarly, IC segments the SI image by automatically searching for different clusters based on a certain criterion. The size of a cluster is determined by its characteristics.

In the MSCA, the size of each subcodebook is set to be identical, i.e., N/s, to facilitate coding and keep the same compression ratio as the conventional VQ, which uses a single codebook of size N. From this point of view, the sizes of regions are expected to be the same, so that each subcodebook can efficiently span the same amount of spectra. Otherwise, the small-size regions are well spanned, whereas the large-size regions are poorly spanned. This will result in uneven error distribution for each subcodebook and ultimately reduce the fidelity of the system. The histogram-based segmentation (HS) method segments a SI image into regions of the same size or as close as possible.

In order to implement this segmentation method, the histogram of the SI image is used and divided into s sections of equal area. Figure 4.22, for example, shows the histogram of the SI image of the test datacube *cube*1. The width of each section is different in order to keep the area constant. The width is wider in a section with low frequency of SI value, while the width becomes shorter in a section with high frequency of SI value. In fact, this method is an

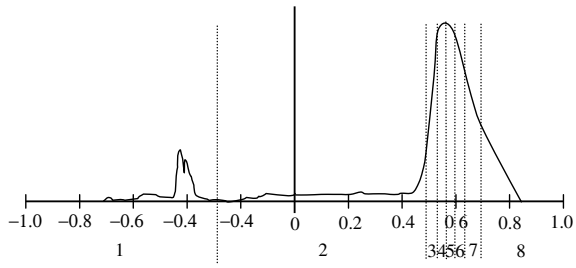


Figure 4.22 Illustration of histogram-based segmentation with a same-size region.

adaptive MT segmentation under the condition of equal-size regions. In practice, it is difficult to find the boundaries in the horizontal axis that make the area of each section identical. Thus, the sizes of the regions are only roughly equal. The third row of Table 4.10 gives the segmentation result using this method.

4.8.1.4 Modified histogram-based segmentation

The segmentation method with same-size regions might not provide good segmentation if the histogram of the SI image has more than one spaced peak whose distributions are different. For example, in Fig. 4.21, the big peak on the right side represents vegetation cover types, and the two small peaks on the left side represent nonvegetated-cover-type water (lakes). The low-frequency part between them represents other nonvegetated cover types (e.g., roads, etc.). If the segmentation method described in Section 4.8.1.3 were used to segment this SI image, the width in this part would be very wide (e.g., section 2), whereas the width in the vicinity of the vegetation peak would be very narrow (e.g., sections 3–7). It is known that different SI values represent different spectra in the scene. A region with a wide range of SI values implies that there is less similarity between the spectral characteristics of the spectra in that region. The region will therefore be less well spanned than a narrow region of the same size if the same-size subcodebooks are used.

There are two alternative ways to handle this problem. The first alternative uses variable-size subcodebooks: A region with a narrow range of SI values is spanned by a small-size subcodebook, whereas a region with a wide range of SI values is spanned by a large-size subcodebook. This alternative is not adopted in this book because it increases the complexity of the MSCA for managing different-size subcodebooks.

Another alternative divides the histogram into p parts, each of which covers a length of histogram with roughly similar distribution and applies the same-size segmentation method described in Section 4.8.1.3 to each of them. The sizes of regions are set to be smaller in the part with a low occurrence frequency,

while the sizes of regions are set to be larger in the part with a high occurrence frequency. If the number of regions in each part is $s_i (i = 1, 2, \ldots, p)$, the total number of regions s is

$$s = \sum_{i\ 1}^{p} s_i. \tag{4.26}$$

This alternative is referred to as modified histogram-based segmentation (MHS).

For example, when we segment the histogram of Fig. 4.22 into $s = 8$ regions using this method, it is divided into three parts. Part 1 covers a range of SI value 1.0 to 0.27 (i.e., section 1 delimited by the same-size segmentation method), and is segmented into $s_1 = 1$ region for water cover type. Part 2 covers a range of SI value 0.27 to 0.45, and is segmented into $s_2 = 3$ regions for nonvegetated cover types. Part 3 covers the area of the big peak and is segmented into $s_3 = 4$ regions for vegetation cover types. The fourth row of Table 4.10 lists the segmentation result of this method.

4.8.2 Methodology of the MSCA

The methodology of the MSCA is shown in Fig. 4.23. The three constituent parts of the formation of SI-based training subsets are shown explicitly and are bounded by a dashed-line rectangle. The SI calculation consists of generating a SI image for an input datacube. The block of SI segmentation

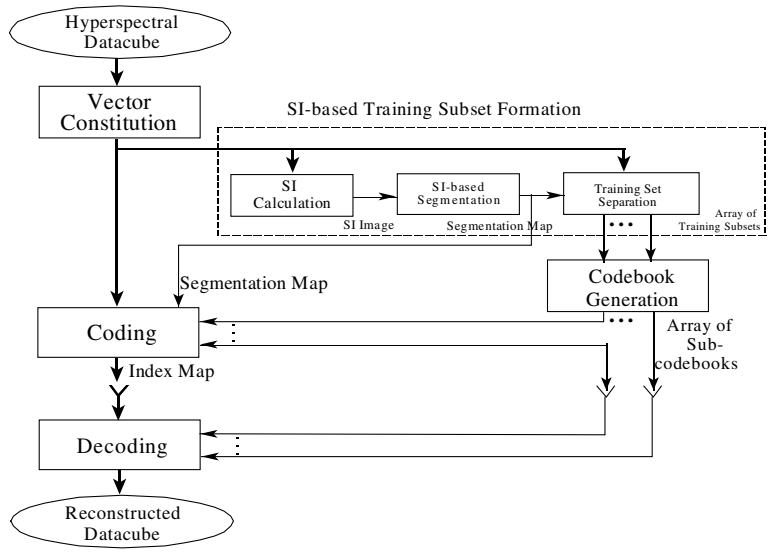


Figure 4.23 Block diagram of the spectral-index-based multiple-subcodebook algorithm.

delimits the SI image into a number of distinct clusters of spectra that share certain spectral characteristics, and a segmentation map is created. The block of training-set separation divides the datacube into s training subsets, each of which corresponds to a region in the segmentation map instead of only a single training set. Each training subset is used independently for training its own subcodebook in the codebook-generation step, and s subcodebooks of equal size N/s are generated. The s subcodebooks are passed to the coding step, where the segmentation map is used to match each vector to be compressed with the best codevector from the appropriate subcodebook.

4.8.3 Improvement in processing time

To gain an appreciation for the expected effect on processing time, let us suppose that the sizes of the segmented regions are identical and that the speed of convergence is always consistent. In addition, suppose that the computational overhead associated with maintaining the multiple subcodebooks can be neglected.

It has been shown in previous discussion that to train a codebook with N codevectors of dimension k from a training set of size n, a total of $n \times N \times k$ products are required for one iteration loop. The number of iterations depends on the convergence speed. In the coding step, $N \times k$ products are required to search for the best codevector in a codebook of size N in order to code an input vector. In MSCA, codebook generation consists of s repetitions of a training process whereby both the training set size and the subcodebook size are reduced by a factor of s; thus, the conventional one-loop iteration time becomes

$$s \times \left(\frac{n}{s} \times \frac{N}{s} \times k \right) = \frac{1}{s} \times (n \times N \times k). \tag{4.27}$$

In other words, the codebook generation time is reduced by a factor of s compared to the conventional VQ. Meanwhile, the coding step in conventional VQ consists of searching the entire codebook to find the best codevector to an input vector. However, in MSCA, only one of the s subcodebooks, whose size is N/s, needs to be searched. Thus, the CT also improves by a factor of s.

4.8.4 Experimental results of the MSCA

Four datasets, called *cube*1 through *cube*4, were tested. These datacubes were acquired using CASI flown in "enhanced spectral" mode. They contain calibrated spectral radiance in 72 spectral bands covering the spectral range of 404 to 913 nm at a spectral interval of approximately 7.2 nm. They were extracted from flight lines acquired in July 1994 during a field campaign of the Boreal Ecosystem-Atmosphere Study.[44] The scene contains mainly black spruce trees with some tamarack and birch. There are also some roads and a

few lakes. *cube*1 is used as the example to describe the test results. Its size is 128 pixels by 2200 lines by 72 spectral bands of 16-bit. The PSNR defined in Eq. (4.4) is used to assess the compression fidelity. Three codebook sizes of $N = 4096$, 1024, and 256 were used. Because these codebooks are trained specifically for the datacube to be compressed, they all need to be transferred to a decoder. Equation (4.25), described in Section 4.7, is used to calculate the compression ratios for them. These codebooks correspond to compression ratios of 40:1, 81:1, and 127:1, respectively, for both the single-codebook 3D VQ algorithm and the MSCA.

The compression results of *cube*1 obtained using the MSCA with the segmentation maps generated in Section 4.8.1 are shown in Fig. 4.24. Each codebook consists of $s = 8$ subcodebooks of same size, when the scene of the datacube is segmented into $s = 8$ regions. The conventional 3D VQ[39] is taken as the reference; the compression results obtained by the reference are also shown in the figure for comparison. The graph on the left of Fig. 4.24 shows the PSNR of compressed data using the MSCA with each of the four segmentation methods.

In the figure, the CGTs using the MT and the IC methods did not take into account the time spent for segmentation by an operator. The processing time for generating the segmentation maps varied over a large range. The IC method took 5 min. It can take a few hours for an operator to segment the SI

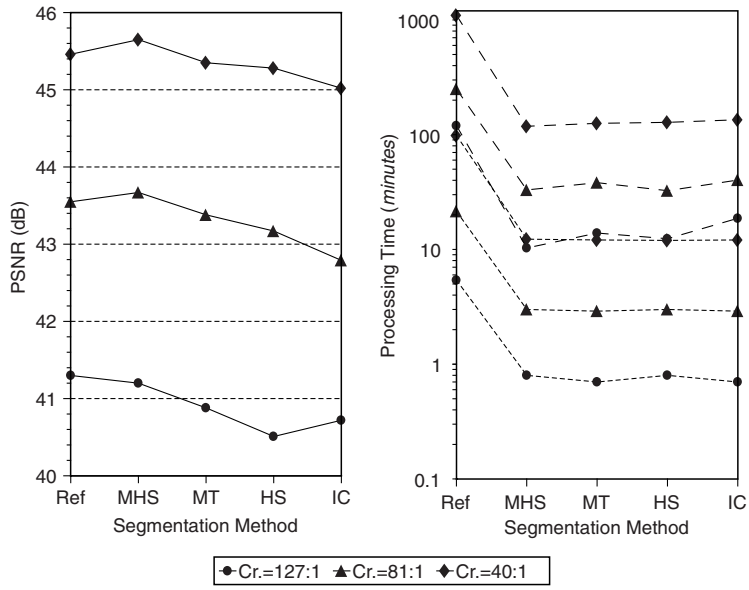


Figure 4.24 Compression results of *cube*1 using MSCA as a function of different segmentation methods when the number of regions is $s = 8$. The left graph shows the PSNR, and the right graph shows the CGT (dashed lines) and CT (dotted lines).

image using the MT method depending on the experience of the operator. The two histogram-based methods spent less than 1 s. The right graph of Fig. 4.24 shows that the improvement in CGT and CT are approximately $s = 8$, as analyzed in Section 4.8.3.

The PSNRs yielded by the MSCA with MHS are the best. Compared with those of the reference, they are slightly improved for smaller compression ratios 40 and 81 (0.19 and 0.12 dB better). This is because the spectra in the datacube are well delimited, and each subcodebook spans almost the same number of distinct spectra, thus the system error is uniformly distributed.

The MSCA with MT segmentation produces the second-best PSNRs. The PSNRs are decreased by 0.27 dB on average. These losses in fidelity result from the mismatch between identical-size subcodebooks and uneven-size regions.

The PSNRs produced by MSCA with the histogram-based segmentation are decreased by 0.45 dB on average. It can be seen from the histogram of the SI image shown in Fig. 4.22 that the bandwidth of sections is sharply increased in the low-frequency part in order to keep the size of the regions close to the constant size. For instance, the bandwidth of section 2 is as wide as 0.76, which is 15 to 19 times wider than the bandwidth of 0.04 to 0.05 for sections 3–7. The error distribution is uneven if identical size subcodebooks are used to span these regions. This results in decrease of fidelity of the overall system. In order to evaluate the performance of each subcodebook, the reconstruction fidelity by each subcodebook is calculated. The fidelities of region 2 are the worst in the eight regions: Corresponding to the three compression ratios 40, 81, and 127, they are 40.53, 37.92, and 34.69 dB, respectively, which are about 5 dB worse than their overall PSNRs 45.28, 43.17, and 40.51 dB.

The MSCA with IC produces the worst results of the four segmentation methods. The PSNRs are decreased by 0.53 dB on average. This is probably caused by constraining the number of regions s to being a power of 2.

It has been shown in Section 4.8.3 that in the MSCA, the larger the number of regions s is, the greater the improvement in the CGT and CT. Is it true that the larger the number of regions s is, the better the performance? The number of regions in *cube*1 has been known between 12 and 15 after analysis using isoclustering without constraint to the number of clusters. The MSCA with the number of regions beyond this range was tested.

The SI image of *cube*1 is segmented into $s = 16$ and 32 regions by the MHS method, as it produces the best performance. Two arrays of training subsets are formed based on these two segmentation maps and are used to generate 16 and 32 subcodebooks of the same size at an overall codebook size $N = 4096$, 1024, and 256. These codebook sizes correspond to the same compression ratios 40:1, 81:1, and 127:1 as when $s = 8$ regions. The compression results using these two

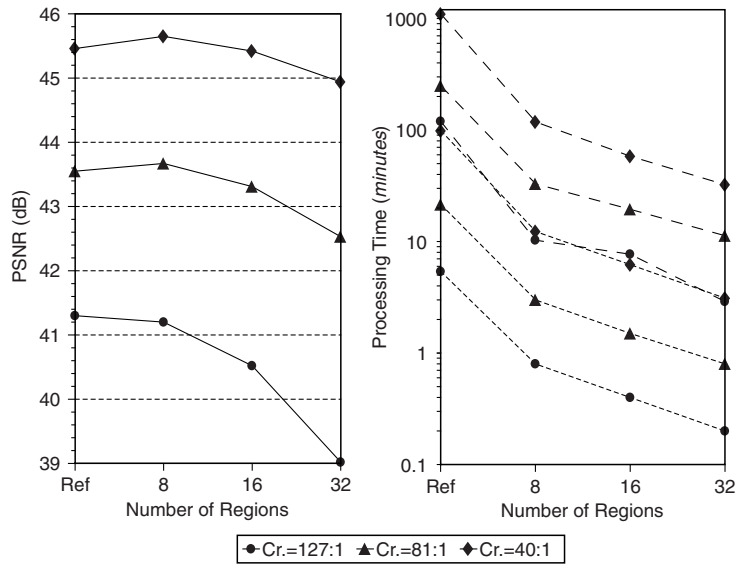


Figure 4.25 Compression results of *cube*1 using the MSCA with the MHS as a function of the number of regions s. The left graph shows the PSNR, and the right graph shows the CGT (dashed lines) and CT (dotted lines).

arrays of subcodebooks are shown in Fig. 4.25. For the purpose of comparison, the reference case as well as the results using a segmentation map of $s = 8$ are also shown in the figure. When the number of regions $s = 16$, the CGT and CT are improved by a factor of 15.7 and 14.6, respectively, on average, while the average PSNR loss is 0.35 dB. When $s = 32$, the average improvement in the CGT and CT is 32.5 and 28.9, respectively, but the average PSNR loss is increased to 1.27 dB. Although the improvement in the CGT and CT increases with the increase of s, the reconstruction fidelity decreases. Especially when s is far beyond the real number of regions of the dataset, the fidelity rapidly decreases. Therefore, the number of regions s cannot be selected arbitrarily. In this example, it is appropriate to select $s = 8$ or 16.

In order to test the robustness and applicability of the MHS method, each of the previous experiments were repeated on three different test datacubes whose scenes contain cover types similar to *cube*1. The results showed similar trends.

4.8.5 MSCA with training set subsampling

The next three subsections describe three improved VQ hyperspectral-data-compression systems that integrate the MSCA, training set subsampling, and the SFBBC binary coding:

1. the MSCA with training set subsampling,

2. the MSCA with training set subsampling plus SFBBC codebook training, and
3. the MSCA with training set subsampling plus SFBBC for both codebook training and coding.

(The effect of these three compression systems on remote sensing products is described in Section 8.7.)

The *cube*1 from previous simulation tests will be used as test datacube. The previous discussion has shown that the MSCA using the MHS with the number of regions $s = 8$ produced the best performance. (This MSCA configuration is adopted in the next three subsections. The same codebook sizes as in Section 4.8.4 will be used, and each codebook has eight subcodebooks.)

In VQ hyperspectral-data-compression techniques, the CGT is much larger than the CT. It is roughly m times as large as the CT if the number of iterations is equal to m because one iteration takes the same amount of time as coding, as discussed in Section 4.1. It is often the case that a codebook training process takes $m = 10$ iterations or more. Thus, it is necessary to reduce the CGT in order to speed up the overall processing time. Because the CGT is proportional to the size of the training set, it follows that the CGT can be reduced by subsampling the training set. The difficulty is that as the training set size is reduced, its ability to span the data declines, and beyond a certain point the fidelity penalty incurred begins to outweigh the speed increase.

In early work,[42] the experimental results demonstrated that the subsampling rate of a training set of hyperspectral data that allows optimal tradeoff between efficiency and fidelity appears to lie in the 2–4% range. This section uses a subsampling rate of 2%.

Before showing the simulation results of the MSCA with subsampled training subsets, two different methods of subsampling need to be described.

- *Method 1* (*equal percentage*): Each training subset is subsampled at the same percentage of 2%. The size of each subsampled training subset is different, as the size of each training subset is different. They are listed in the second line of Table 4.11. The size of each training subset is also listed in the table (first line). Sometimes, for a small training subset, the size of the subsampled training subset can be smaller than that of the subcodebook to be trained from it. In Table 4.11, for example, the sizes of the subsampled training subsets of regions 2, 3, and 4 are 170, 176, and 180, respectively. These are all smaller than the subcodebook size, $N/s = 4096/8 = 512$ (for the compression ratio of 40:1), which is not desirable. If this happened, the reconstruction fidelity will be very poor for that subcodebook.
- *Method 2* (*equal size*): Each training subset is subsampled at a different rate to obtain same-size subsampled training subsets. The size depends

Table 4.11 Size of subsampled training subsets of *cube*1 subsampled at the rate of 2% (the number of regions $s = 8$).

	Region Number (Region Size)							
Segmentation Method	**1 (35206)**	**2 (8376)**	**3 (8662)**	**4 (8843)**	**5 (54958)**	**6 (56457)**	**7 (54811)**	**8 (54287)**
Equal Percent	718	170	176	180	1121	1151	1118	1107
Equal Size	718							

on the overall subsampling rate. The sizes of subsampled training subsets of *cube*1 using this method at a rate of 2% are listed in the second line of Table 4.11. Each subsampled training subset contains the same number of vectors (718).

The MSCA was tested with two arrays of subsampled training subsets extracted using the two subsampling methods. They produced almost the same improvement in the CGT but produced no improvement in the CT because the training-set subsampling technique does not have influence on the CT. The MSCA with subsampling method 2 yielded better reconstruction fidelity than with subsampling method 1. This is because some subsampled training subsets obtained with method 1 are even smaller than the sizes of subcodebooks.

Figure 4.26 shows the simulation results of the MSCA with the equal-size subsampled training subsets. The left graph shows the PSNR of the three compression ratios, and the right graph shows the CGT (dashed lines), the CT (dotted lines), and overall processing time (PT) (solid lines). The PT includes the CGT, CT, and time for creating a segmentation map. For the purpose of comparison, the PSNR, CGT, CT, and PT produced by the conventional 3D VQ are also shown as circles in the same graph. CGTs are reduced to 2.3, 0.54, and 0.12 min (the dashed line marked by squares at bottom of the right graph), respectively, from 1098, 250, and 120 min (the dashed line marked by circles at top of the right graph) at the three compression ratios of 40, 81, and 127. They are improved by factors of 477, 463, and 1000, respectively; these improvements result from two contributions. One comes from the methodology of the MSCA, which brings an improvement of approximately $s = 8$ times. Another comes from the reduction of subsampling training subsets to 2%, which brings an improvement of around 50 times. Thus, the improvement in the CGT is $8 \times 50 = 400$ times, assuming that the training convergence speed remains constant. But the real CGT improvements are all larger than this.

The CTs are improved by a factor of around $s = 8$ compared with the reference. In the reference case, the PTs are dominated by the CGT because the CGTs are two orders of magnitude larger than the CTs. That is why the

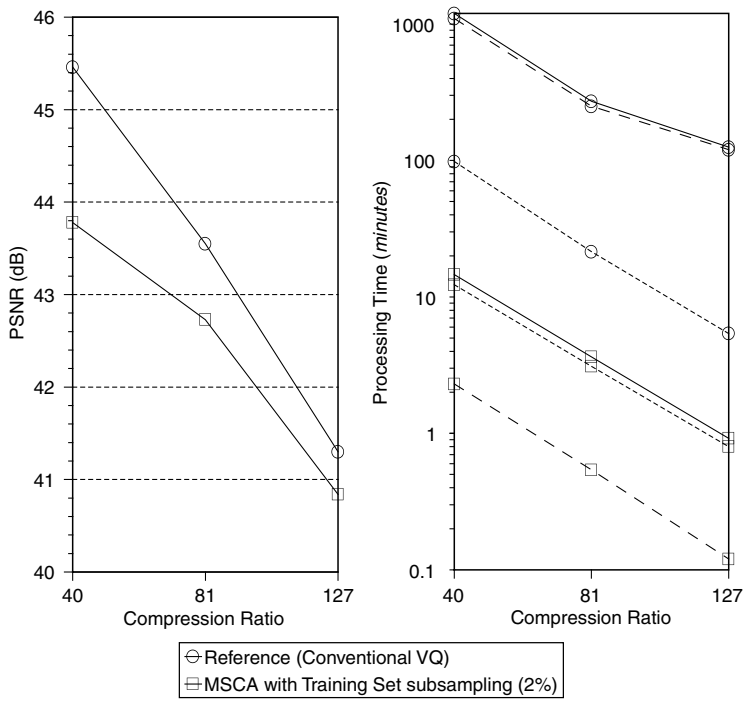


Figure 4.26 Performance comparison of the MSCA with training set subsampling (2%) against the conventional VQ. The left graph shows PSNR curves, and the right graph shows the curves of the CGT (dashed lines), the CT (dotted lines), and the overall processing time (solid lines).

PT curve (solid line marked by circles at top of the right graph) is only slightly above the CGT curve (dashed line marked by circles at the top of the right graph). Meanwhile, in the MSCA with 2% STSSs, the CGTs are reduced to around six times as small as the CTs. Thus, CTs dominate the PT (solid line marked by squares in the middle of the right graph). The overall processing speed is improved by a factor of 82, 75, and 136, respectively, at the three compression ratios. The costs for all of these improvements are losses of PSNR of 1.68, 0.82, and 0.46 dB, respectively.

4.8.6 MSCA with training set subsampling plus SFBBC codebook training

Section 4.5 describes SFBBC binary coding: it converts a spectral vector $\mathbf{X}_{i,j} = \{x_{i,j}(\lambda)(\lambda = 1, 2, \ldots, N_b)\}$ of size $16N_b$ bits into a binary code of size ($3N_b$ 2) bits, where N_b is the number of elements of the spectral vector, and 16 is the word-length of each element. A SFBBC vector is much shorter than an original spectral vector. Furthermore, with the SFBBC vectors, the distance measure between two vectors in both the codebook training and

coding processes can be implemented by using the Hamming distance instead of the Euclidean distance. The former is much faster to compute than the latter because the Hamming distance is just a sum of bitwise exclusive-or operations. Using this algorithm, both the CGT and CT were reduced by a factor of 30–40, as reported in Section 4.5.

This section describes the VQ compression system that integrates the MSCA, training set subsampling, and SFBBC codebook generation. Figure 4.27 shows the experimental results of this system. Three arrays of subcodebooks of 4096, 1024, and 256 codevectors were generated from 2% subsampled training subsets by the SFBBC fast-training algorithm. The CGTs are further reduced to 0.10, 0.05, and 0.04 min, respectively (the dashed line at bottom of the right graph). Compared to the reference, they are improved by factors of 10,980, 5,000, and 3,000, respectively. The codebooks are then used to compress the datacube by the MSCA using the conventional coding algorithm. The CTs are improved by a factor of $s = 8$, similar to that in Section 4.8.5. The CTs dominate the overall processing time because they are

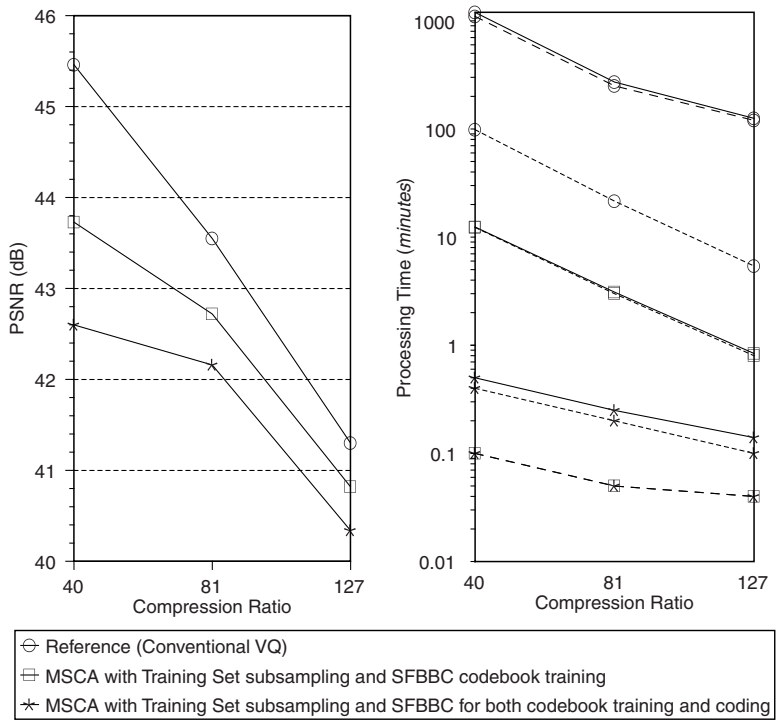


Figure 4.27 Performance comparison of the MSCA with training set subsampling (2%) and the SFBBC algorithms. The left graph shows PSNR curves, and the right graph shows the curves of the CGT (dashed lines), the CT (dotted lines), and the overall processing time (solid line).

tens of times as large as the CGTs, shown by the solid line and dotted line curves marked by squares in the middle of the right graph of Fig. 4.27. The overall processing speeds increase by 96, 88, and 150 times, respectively, at three compression ratios. The fidelity losses are 1.73, 0.83, and 0.48 dB, respectively (the curve marked by squares in the left graph of Fig. 4.27).

4.8.7 MSCA with training set subsampling plus SFBBC for both codebook training and coding

The same codebooks generated in Section 4.8.6 are used, but coding is performed with the SFBBC coding algorithm. The CTs are improved by factors of 246, 108, and 54, respectively, at the three compression ratios. These improvements result from a joint contribution of the MSCA and the SFBBC coding. They are still several times as large as the CGTs and are the major contributor to the overall processing time. The overall processing speeds are improved by factors of 2393, 1086, and 895, respectively, but the fidelity losses increase to 2.86, 1.39, and 0.96 dB, respectively (the curve marked by asterisks in the left graph of Fig. 4.27).

In order to test the robustness and applicability, each of the previous simulations was repeated on three different test datacubes. The experimental results showed similar trends.

4.9 Successive Approximation Multistage Vector Quantization

4.9.1 Compression procedure

In order to overcome the shortcomings of the conventional VQ algorithm for hyperspectral datacubes, a successive approximation multistage vector quantization (SAMVQ) algorithm for encoding multidimensional data was developed.[45] SAMVQ is built on a multistage VQ framework, which is a general approach to lowering the computation complexity in VQ methods.[5] However, the existing multistage VQ is severely limited in use, and the fidelity is substantially poorer than the full-search VQ methods; thus it is not well adopted. SAMVQ is a novel multistage VQ algorithm and brings this framework to practical use. SAMVQ achieves a fidelity that approaches or exceeds the full-search VQ method. There is no limit to multistage in SAMVQ.

Figure 4.28 shows the block diagram of SAMVQ. In a first-approximation stage, a first codebook having a small number of codevectors (normally $N_m = 8$ or 16) is generated from an input datacube. A codevector in the codebook corresponds to a cluster of spectral vectors and is the centroid of the cluster. Selection of such a codebook to accurately reflect the vectors of a given datacube is unlikely, and as such, less care needs to be taken to ensure high fidelity of the results. Next, the spectral vectors of the datacube are encoded

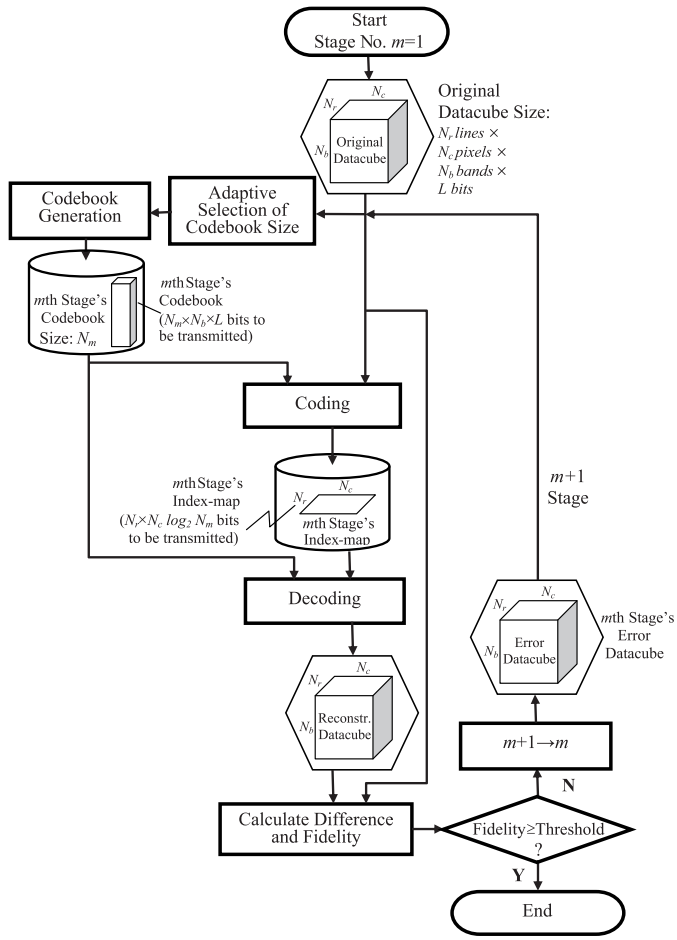


Figure 4.28 Block diagram of the SAMVQ algorithm.

using the small codebook first by determining a codevector within the first codebook that best approximates the spectral vector within the datacube. Generation of a first index map replaces each spectral vector with an index indicative of the codevector's location within the first codebook. Encoding the datacube with a small codebook will result in encoded data having a low fidelity. In preparation for a following approximation stage, the difference based on the original datacube and the encoded datacube is determined. The difference data is obtained as a difference between the original datacube and reconstructed datacube after decoding. The difference data becomes input data for a subsequent approximation stage.

In the second subsequent approximation stage, a new approximation is created by generating a new small codebook and encoding each error vector of the difference data by determining a codevector within the new small

codebook that best approximates the error vector within the difference data. A second index map is created by replacing the error vectors with an index indicative of the codevector's location within the new small codebook. New difference data based on the encoded datacube and the encoded difference data is then determined. The new difference data become input data for a subsequent approximation stage.

The approximation stages comprising the encoding of the difference data are continued until a control error of the difference data is smaller than a given threshold or a predefined number of approximation stages M is reached.

4.9.2 Features

In SAMVQ, the computational burden present in conventional VQ algorithms for 3D datacubes is no longer a problem, as the codebook size N_m of an approximation stage is over two orders of magnitude smaller. Codebook generation time is much faster than with conventional VQ algorithms because the codebook at each approximation stage is much smaller. To generate a codebook for a given datacube, the CGT is proportional to the size of the codebook. In conventional VQ algorithms a single codebook of size N is generated. The CGT is proportional to N:

$$CGT_{VQ} \propto N, \tag{4.28}$$

whereas for SAMVQ, the CGT is proportional to the cumulative sum of the sizes of the codebooks generated at each approximation stage:

$$CGT_{SAMVQ} \propto N' = \sum_{m\ 1}^{M} N_m, \tag{4.29}$$

where N_m is the size of the codebook at the mth approximation stage; in general, $N_m \ll N$, and M is the number of approximation stages.

Because the CT and decoding time (DT) are also proportional to the codebook size, the coding and the decoding are much faster for SAMVQ than conventional VQ algorithms. This improvement in processing speed has no fidelity penalty because the SAMVQ method spans the data with a virtual codebook proportional to the product of the sizes of the codebooks generated at each approximation stage. Therefore, the size of the virtual codebook is

$$\tilde{N} = \prod_{m\ 1}^{M} N_m. \tag{4.30}$$

For example, assume that SAMVQ comprises a four-stage approximation with codebooks having an identical size of $N_m = 8$ codevectors at each stage. The virtual codebook—the equivalent codebook for conventional VQ algorithms—would then have $\tilde{N} = 8^4 = 4096$ codevectors to achieve the

same reconstruction fidelity as SAMVQ having only $N' = 4 \times 8 = 32$ codevectors. Both the CGT and the CT are improved by a factor of

$$\frac{\tilde{N}}{N'} = \frac{8^4}{4 \times 8} = 128. \tag{4.31}$$

By varying the size of the codebook N_m used at each approximation stage and the number of approximation stages m, the compression ratio of the encoding as well as the fidelity of the reconstructed data is controllable. Therefore, SAMVQ is highly advantageous for various applications.

The compression ratio obtained by SAMVQ is greater than that by the conventional VQ algorithms while the fidelity of the reconstruction data remains the same because the total number of codevectors is smaller ($N' = 4 \times 8 = 32$, instead of $N = 8^4 = 4096$).

The compression ratio of SAMVQ with m-stage approximation of a hyperspectral datacube is expressed as

$$C_r = \frac{N_r N_c N_b L}{\sum_{m=1}^{M} (N_r N_c \log_2 N_m + N_m N_b L)}, \tag{4.32}$$

where N_r and N_c are the number of lines and the number of pixels in the scene of the hyperspectral datacube, N_b is the number of spectral bands, and L is the word-length of data value. In Eq. (4.32), the overhead of codebooks at each approximation stage is included.

The set of codebooks and index maps of the multiple approximation stages is easily combined into a single codebook and a single index map similar to those obtained from conventional VQ algorithms. For example, a single index map is optionally constructed by concatenating the indices indicating the same spatial location in the index maps, allowing fast calculation of a corresponding combined codevector for decoding.

Another highly advantageous feature of the SAMVQ method is the creation of encoded data (codebook and index map) in subsequent approximation stages. This allows encoding, transmission, and decoding of the same original image data in stages of increasing reconstruction fidelity. For example, the most-significant image information is encoded, transmitted, and decoded first, followed by less-significant image information. Furthermore, if the number of codevectors in the first codebook is determined such that it is close to the number of real classes present in the scene of an input hyperspectral datacube, then the first index map provides spatial information and spectral classification information of the hyperspectral datacube.

Given the multiple-stage approximation nature in the compression, SAMVQ can operate in either a fixed-fidelity or fixed-compression-ratio mode. In the former mode, given a desirable reconstruction fidelity threshold

(such as PSNR), SAMVQ will automatically compress one stage after another until the reconstruction fidelity reaches the fidelity threshold. During the course of compression, the SAMVQ algorithm adaptively selects the codebook size at each approximate stage to yield the best compression ratio and the fastest processing time. This mode has been examined and demonstrated on eight hyperspectral test datasets. In the fixed-compression-ratio mode, given the number of stages and the size of the codebook at each stage in terms of the compression ratio needed, the algorithm compresses the data until the compression of all approximation stages are completed. The fidelity of the compression varies with the input data.

Lossy compression can be viewed as near-lossless compression from the point of view of applications if the level of error (noise) introduced by a lossy compressor is smaller than that of the intrinsic noise of the original data. Using the fixed-fidelity mode, one can implement near-lossless compression by setting the value of the fidelity threshold to be slightly less than the SNR of the original data. (Near-lossless compression is discussed in detail in Chapter 5.)

In SAMVQ, a graph of fidelity (such as PSNR) versus compression ratio [which is related to the number of approximation stages and codebook size at each stage as defined in Eq. (4.32)] reaches an asymptote. That is, despite increasing the number of approximation stages, after a certain point the reconstruction fidelity ceases to increase as the compression ratio decreases. This point is referred to as an inflection point of the graph. In order to avoid this, another operating mode called asymptotic compression ratio estimation has been developed and put into use. In this operating mode the inflection point of the compression ratio versus PSNR graph is detected, and compression proceeds automatically stage by stage until the reconstruction fidelity is not improved significantly (i.e., reaches the inflection point). In this mode, the algorithm adaptively selects the codebook size at each approximate stage to yield the best compression ratio and the fastest processing time. This mode has been examined and demonstrated on eight hyperspectral test datasets.

Because spectral information is the signature information for hyperspectral applications, it is critical to prevent loss in lossy data compression. The strategy here gives spectral information integrity a high weight in the compression process. In SAMVQ, the compression fidelity is checked for each single spectral band rather than the overall datacube. If a spectral band whose fidelity has reached the fidelity threshold or the inflection point, the process of approximating to the fidelity threshold or the process of detecting the inflection point ceases for that spectral band. This method guarantees that the fidelity of the reconstructed data is better than the fidelity threshold in each spectral band rather than the overall datacube.

Due to the fact that the convergence speed of each band to the fidelity threshold is different, the bands with a fast convergence speed are no longer involved in the processing of the compression approximation with those bands whose convergence speed is slow. This saves processing time for both codebook generation and coding.

In the process of automatic detection of the inflection point, this strategy produces an inflection point for each spectral band and continues approximate compression on spectral bands whose inflection points have not been reached. This not only saves processing time for both codebook generation and coding, but it also produces a high compression ratio compared to the standard method that checks the fidelity of the overall datacube because it avoids further approximations of spectral bands whose inflection points have been reached in the process, which reduces the number of codevectors.

4.9.3 Test results

The same test data for *cube*1 to *cube*4 acquired using CASI (included in Section 4.8) is used to assess the performance of SAMVQ. Table 4.12 tabulates the experimental results of SAMVQ on *cube*1. For the purpose of comparison, the experimental results produced by the MSCA with different combinations with subsampling of the training set and SFBBC coding (see Sections 4.8.5–4.8.7), denoted as Systems 1, 2, and 3, are also listed in the table. For the SAMVQ algorithm, two cases are tested. The SAMVQ algorithm was applied to the test datacubes without subsampling the training set. This case is referred to as SAMVQ 1 in the table. The SAMVQ algorithm was applied the test datacubes with 2% subsampling of training set in order to compare with the experimental results by MSCA. This case is referred to as SAMVQ 2 in the table.

Table 4.12 Comparison of PSNR and processing time of SAMVQ with the conventional VQ compression algorithm for the similar compression ratio.

Improved VQ System 1			Improved VQ System 2			Improved VQ System 3			SAMVQ 1			SAMVQ 2		
CR	PSNR (dB)	PT (min)	CR	PSNR (dB)	PT (min)	CR	PSNR (dB)	PT (min)	CR	PSNR (dB)	PT (min)	CR	PSNR (dB)	PT (min)
127	40.80	1.2	127	40.70	0.8	127	40.25	0.2	125	40.85	2.8	125	40.42	0.1
81	42.75	4.0	81	42.70	3.5	81	42.19	0.3	81	43.75	4.5	81	43.29	0.2
40	43.80	15.0	40	43.65	13.0	40	42.58	0.5	57	45.90	4.8	57	45.58	0.4

System 1: MSCA + 2% subsampling (Section 4.8.5),
System 2: MSCA + 2% subsampling + SFBBC codebook generation (Section 4.8.6),
System 3: MSCA + 2% subsampling + SFBBC for both codebook generation and coding (Section 4.8.7),
SAMVQ 1: without subsampling of training set,
SAMVQ 2: with 2% subsampling of training set.

It can be seen from the table that SAMVQ outperforms all of the combined compression cases of MSCA in terms of the PSNR and processing time (PT) at similar compression ratios. For simplicity, compare System 3 and SAMVQ 2, as they are the most-compatible cases in terms of subsampling the training set and processing speed. For the lowest-compression-ratio cases, SAMVQ 2 achieved a compression ratio of 57:1 with a PSNR of 45.58 dB, whereas System 3 achieved a compression ratio of 40:1 with a PSNR of 42.58 dB. Both the compression ratio and PSNR of System 3 are worse than those of SAMVQ 2. In addition, the processing time of SAMVQ 2 is faster than System 3. The test results on the other three test datacubes show the similar conclusions.

It is evident that SAMVQ is a superior VQ-based compression algorithm in terms of its efficacy and simplicity for operational use. Two patents on SAMVQ compression techniques have been filed and granted.[45,46] One patent[45] is on the subject discussed in this section, and another patent[46] is on the subject of onboard near-lossless compression, which is discussed in Chapter 5.

4.10 Hierarchical Self-Organizing Cluster Vector Quantization

Another novel and fast VQ compression algorithm, referred to as hierarchical self-organizing cluster vector quantization (HSOCVQ), has been developed by Qian et al.[47,48] The HSOCVQ algorithm merges the codebook training and coding phases into one. Given a desired fidelity threshold, such as a RMSE, HSOCVQ compresses the spectral vectors in a datacube until each of them is encoded with fidelity better than the threshold, which is the main difference between SAMVQ and HSOCVQ in terms of compression fidelity. Reduction of the computational complexity of the search for the nearest codevector is no longer an issue in HSOCVQ, as both the training set size and the codebook size are orders of magnitude smaller than in the conventional VQ algorithms.

4.10.1 Compression procedure

Figure 4.29 shows the block diagram of the HSOCVQ algorithm. It first trains an extremely small number of tentative codevectors (usually $N_1 \leq 8$) using an input datacube as the training set and then uses these tentative codevectors to classify spectral vectors in the datacube into N_1 clusters. Each tentative codevector is the kernel of one of the clusters. The HSOCVQ algorithm then encodes the N_1 clusters one after another.

In encoding cluster #1, the spectral vectors in the cluster are coded using the index of the tentative codevector associated to the cluster (i.e., 1) if the compression fidelity is better than the predetermined threshold. Tentative codevector #1 becomes a formal codevector and is added to the codebook. If the compression fidelity is not better than the threshold, the HSOCVQ

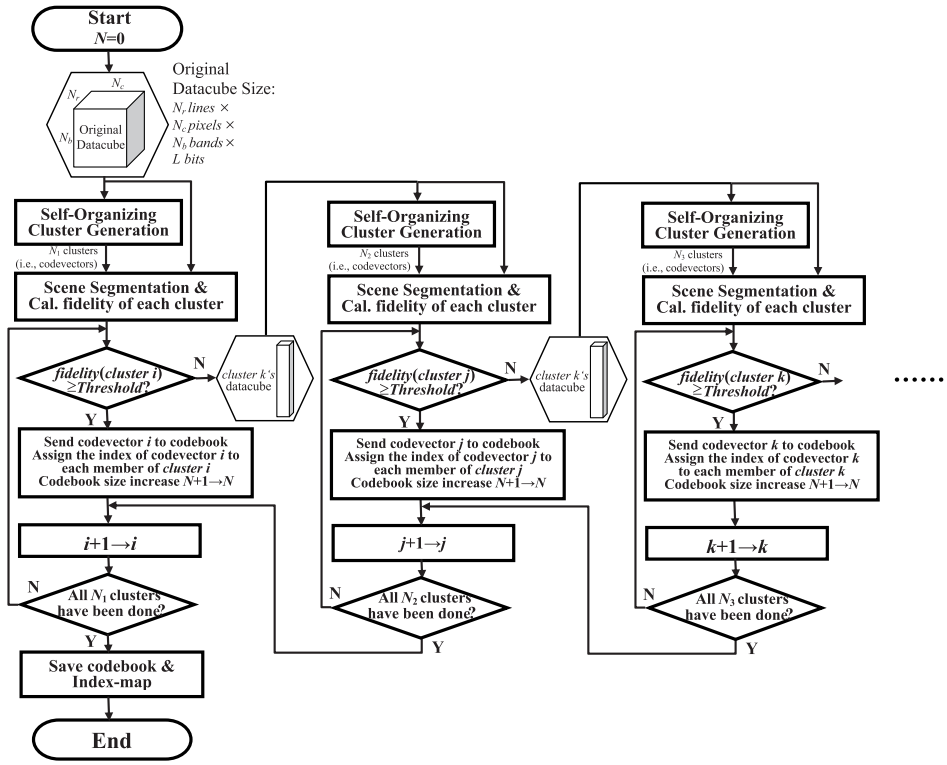


Figure 4.29 Block diagram of HSOCVQ.

algorithm generates an extremely small number (N_2) of tentative codevectors by using cluster #1 as a training set. Cluster #1 is then split into N_2 subclusters, each of which associates to a tentative codevector. The number of tentative codevectors is determined adaptively. The distance between the spectral vectors in cluster #1 and the kernel of the cluster (i.e., tentative codevector #1) is calculated and used to estimate the number of the tentative codevectors. A relatively larger number of tentative codevectors (e.g., 8) are generated if the distance is large. A relatively small number of tentative codevectors (e.g., 2) are generated if the distance is small. HSOCVQ then encodes spectral vectors in each of the subclusters using the tentative codevector associated to the subcluster and checks if each of the vectors has compression fidelity better than the threshold. If a spectral vector whose compression fidelity associated with the tentative codevector is better than the threshold, it is encoded using the tentative codevector and excluded from the subcluster. The tentative codevector associated to the subcluster becomes a formal codevector and is added to the codebook. The remaining spectral vectors whose fidelity associated with the formal codevector is not better than the threshold are further split by adaptively generating new tentative

codevectors and clustering the subcluster using the same method described earlier. The tentative codevector associated with a subcluster will not become a formal codevector and be added in the codebook if it does not encode a single spectral vector in the subcluster. This process repeats until all of the N_2 subclusters are encoded for cluster #1, and then the HSOCVQ algorithm encodes cluster #2, and so on until all N_1 clusters have been processed.

4.10.2 Features

One of the unique features of HSOCVQ is that it guarantees that the reconstruction fidelity of each spectrum in the compressed datacube is better than the predetermined fidelity. This feature allows HSOCVQ to preserve small targets or "golden" spectra in a hyperspectral datacube. HSOCVQ operates only in Fidelity mode. Similar to SAMVQ, setting the compression fidelity to a level better than that of the instrument that captured the data can result in a near-lossless compression (or a lossless compression if accounting for the instrument noise).

HSOCVQ can be viewed as a traditional tree-structured VQ with at least the following four main differences:

1. Online design of the tree structure and transmission of codevectors;
2. Flexible tree structure (i.e., adaptive breadth and depth of tree);
3. Encoding converged vectors of a cluster at any given depth of the tree using their kernel as codevector and excluding the vectors from the cluster for further training; and
4. A stopping criterion based on maximum error.

The first point enables HSOCVQ to be a real-time algorithm, whereas the fourth point allows it to control the compression fidelity for near-lossless compression. The second point results from the adaptive selection of the number of codevectors to be generated for a cluster and from the different convergence rates of the clusters. It results in compression to a desired fidelity with the fewest terminal nodes (i.e., codevectors). Its cost is the calculation of the fidelity of each vector in the cluster to the kernel of the cluster. The third point is inconsistent with the traditional tree-structured VQ because it assigns the kernel of a cluster as the codevector of the node if at least a vector in the cluster is converged to the compression fidelity. This may increase the total number of codevectors. The benefit is that the clusters become smaller and smaller when the splitting goes deep. This speeds up the processing of the clusters and decreases the total breadth and depth of the tree to reduce the total number of codevectors.

In the VQ technique, to generate a codebook for a given datacube, the CGT is proportional to the size of codebook and the size of the training set. In HSOCVQ, the clusters, subclusters, and sub-subclusters generated are disjoint, and their sizes are decreased as the hierarchical splitting levels increase. Thus

the compression process is extremely fast and efficient, as both the codebook size and training set (cluster or subclusters) size are small, and the spectral vectors in each cluster or subcluster are not trained twice. CGT is much faster than in the conventional 3D VQ because the size of codebook is much smaller in general. CT (codevector match) and DT are also much faster than conventional 3D VQ because CT and DT are also proportional to the codebook size.

In HSOCVQ, the codebook is generated in real time without any *a priori* knowledge or user interaction. This feature allows the compression technology to be applied in real time aboard a spacecraft.

Moreover, each codevector in the codebook that is associated with a specific cluster is trained only from the data (vectors) that belongs to the cluster. In other words, the size of training set for training each codevector is much smaller than the entire datacube. The improvement in processing speed offered by HSOCVQ against conventional 3D VQ has no fidelity penalty.

The compressed data—the index map—produced by this compression technique is actually a classification map of the datacube. In addition, the clusters in the index map are well ordered. Similar clusters have close class numbers. Moreover, controlling the fidelity of compression can be used to control the accuracy of the classification. This is a unique property not possessed by supervised or unsupervised classification methods. The processing time required to compress a datacube using HSOCVQ is faster than that required to classify the datacube using a supervised or an unsupervised classification method.

Though an increased fidelity requirement demands more processing time (because additional recursions are needed), the increase is not so substantial, and the trade-off is mainly between fidelity and compression ratio. This allows the general trade-off between fidelity and compression ratio to be tailored to the system application.

References

1. Gray, R. M., "Vector quantization," *IEEE ASSP. Mag.* **1**, 4–29 (1984).
2. Linde, Y., A. Buzo, and R. M. Gray, "An algorithm for vector quantizer designs," *IEEE Trans. Comm.* **28**, 84–95 (1980).
3. Nasrabadi, N. M. and Y. Feng, "Vector quantization of images based upon the Kohonen self-organizing feature maps," *Proc. IEEE Int. Conf. Neural Networks 1988*, 101–108 (1988).
4. Abut, H., *Vector Quantization*, IEEE Reprint Collection, IEEE Press, Piscataway, NJ (1990).
5. Gersho, A. and R. M. Gray, *Vector Quantization and Signal Compression*, Kluwer, Boston, MA (1992).

6. Equitz, W. H., "A new vector quantization clustering algorithm," *IEEE Trans. Acoust., Speech, Signal Process.* **37**, 1568–1575 (1989).
7. Sproull, R. F., "Refinements to nearest neighbor searching in k-dimensional trees," *Algorithmica* **6**, 579–589 (1991).
8. Ramasubramanian, V. and K. Paliwal, "Fast K-dimension tree algorithm for nearest neighbour search with application to vector encoding," *IEEE Trans. Signal Process.* **40**, 518–531 (1992).
9. Chen, S.-H. and W. M. Hsieh, "Fast algorithm for VQ codebook design," *Proc. Inst. Elect. Eng.* **138**, 357–362 (1991).
10. Orchard, M. T., "A fast nearest-neighbor search algorithm," *Proc. IEEE ICASSP 1991*, 2297–2300 (1991).
11. Huang, C.-M., Q. Bi, G. Stiles, and R. W. Harris, "Fast full-search equivalent encoding algorithms for image compression using vector quantization," *IEEE Trans. Image Process.* **1**, 413–416 (1992).
12. Ramasubramanian, V. and K. Paliwal, "Fast nearest-neighbour search base on Voronoi projections and its application to vector quantization encoding," *IEEE Trans. Speech Audio Process.* **7**, 221–226 (1997).
13. Ra, S. W. and J. K. Kim, "A fast mean-distance-oriented partial codebook search algorithm for image vector quantization," *IEEE Trans. Circuits System II* **40**, 576–579 (1993).
14. Li, W. and E. Salari, "A fast vector quantization encoding method for image compression," *IEEE Trans. Circuits System Video Technology* **5**, 119–123 (1995).
15. Chen, T.-S. and C.-C. Chang, "Diagonal axes method (DAM): A fast search algorithm for vector quantization," *IEEE Trans. Circuits Syst. Video Technol.* **7**, 555–559 (1997).
16. Hsieh, C.-H. and Y.-J. Liu, "Fast search algorithms for vector quantization of images using multiple triangle inequalities and wavelet transform," *IEEE Trans. Image Process.* **9**, 321–328 (2000).
17. Wu, K.-S. and J.-C. Lin, "Fast VQ encoding by an efficient kick-out condition," *IEEE Trans. on Circuits and Systems for Video Tech.* **10(1)**, 59–62 (2000).
18. Mielikainen, J., "A novel full-search vector quantization algorithm based on the law of cosines," *IEEE Signal Processing Letters* **9**(6), 175–176 (2002).
19. Lu, Z.-M. and S.-H. Sun, "Equal-Average Equal-Variance Equal-Norm Nearest Neighbour Search Algorithm for Vector Quantization," *IEICE Trans. INF. & SYST.* **E86-D**(3), 660–663 (2003).
20. Lee, C. H and L.H. Chen, "A fast search algorithm for vector quantization using mean pyramids of codewords," *IEEE Trans. Commun.* **43**, 1697–1702 (1995).

21. Pan, J. S., Z. M. Lu, and S. H. Sun, "Fast codeword search algorithm for image coding based on mean-variance pyramids of codewords," *Electron. Lett.* **36**(3), 210–211 (2000).
22. Song, B. C. and J. B. Ra, "A fast search algorithm for vector Quantization using L_2-norm pyramid of codewords," *IEEE Trans. Image Process.* **11**, 10–15 (2002).
23. Kaukoranta, T., P. Franti, and O. Nevalainen, "A fast exact GLA based on codevector activity detection," *IEEE Trans. Image Process.* **9**(8), 1337–1342 (2000).
24. Bei, C. D. and R. M. Gray, "An improvement of the minimum distortion encoding algorithm for vector Quantization," *IEEE Trans. Comm*. **33**, 1132–1133 (Oct. 1985).
25. Qian, S.-E., "Hyperspectral data compression using a fast vector quantization algorithm," *IEEE Trans. Geosci. Remote Sens.* **42**(8), 1791–1798 (2004).
26. Qian, S.-E., "Fast vector quantization algorithms based on nearest partition set search," *IEEE Trans. Image Process.* **15**(8), 2422–2430 (2006).
27. Neville, R. A., N. Rowlands, R. Marois, and I. Powell, "SFSI: Canada's First Airborne SWIR Imaging Spectrometer," *Canadian J. Remote Sen.* **21**, 328–336 (1995).
28. Hu, B., S.-E. Qian, D. Haboudane, J. R. Miller, A. Hollinger, and N. Tremblay, "Retrieval of crop chlorophyll content and leaf area index from decompressed hyperspectral data: the effects of data compression," *J. Remote Sens. Environ.* **92**(2), 139–152 (2004).
29. Qian, S.-E, A. B. Hollinger, M. Dutkiewicz, H. Zwick, and J. Freemantle, "Effect of lossy vector quantization hyperspectral data compression on retrieval of red edge indices," *IEEE Trans. Geosci. Remote Sens.* **39**(7), 1459–1470 (2001).
30. Hu, B., S.-E. Qian, and A. B. Hollinger, "Impact of lossy data compression using vector quantization on retrieval of surface reflectance from CASI imaging spectrometry data," *Canadian J. Remote Sen.* **27**(1), 1–19 (2001).
31. Qian, S.-E., B. Hu, M. Bergeron, A. Hollinger, and P. Oswald, "Quantitative evaluation of hyperspectral data compressed by near lossless onboard compression techniques," *Proc. IGARSS 2002*, 1425–1427 (2002).
32. Qian, S.-E., M. Bergeron, C. Serele, and A. Hollinger, "Evaluation and comparison of JPEG 2000 and VQ based on-board data compression algorithm for hyperspectral imagery," *Proc. IGARSS 2003*, 1820–1822 (2003).

33. Hu, B., S.-E. Qian, D. Haboudane, J. R. Miller, A. B. Hollinger, and N. Tremblay, "Double blind evaluation of the impact of VQ data compression on hyperspectral data in the retrieval of crop chlorophyll content and leaf area index in precision agriculture application," *Remote Sens. Environ.* **92**(2), 139–152 (2004).
34. Staenz, K., R. Hitchcock, S.-E. Qian, and R. A. Neville, "Impact of on-board hyperspectral data compression on mineral mapping products," *Int. Conference ISPRS* (Dec. 2002).
35. Qian, S.-E, A. Hollinger, M. Bergeron, I. Cunningham, C. Nadeau, G. Jolly, and H. Zwick, "A Multi-disciplinary User Acceptability Study of Hyperspectral Data Compressed Using Onboard Near Lossless Vector Quantization Algorithm," *Int. J. Rem. Sens.* **26**(10), 2163–2195 (2005).
36. Qian, S.-E., A. B. Hollinger, D. Williams, and D. Manak, "Fast 3D data compression of hyperspectral imagery using vector quantization with spectral-feature-based binary coding," *Opt. Eng* **35**(11), 3242–3249 (1996).
37. Viterbi, A. J. and J. K. Omura, "*Principles of Digital Communication and Coding,*", p. 81, McGraw-Hill, New York (1979).
38. Alan Mazer, S. et al., "Image Processing Software for Imaging Spectrometry Data Analysis," *Rem. Sens. of Environment* **24**, 201–210 (1988).
39. Qian, S.-E., A. B. Hollinger, D. Williams, and D. Manak, "A near lossless 3-dimensional data compression system for hyperspectral imagery using correlation vector quantization," *Proc. 47th International Astronautical Congress*, Beijing, China (1996).
40. Bannari, A., D. Morin, F. Bonn, and A. Huete, "A Review of Vegetation Indices," *Rem. Sens. Rev.* **13**, 95–120 (1995).
41. Qian, S.-E., A. B. Hollinger, D. Williams, and D. Manak, "3D data compression of hyperspectral imagery using vector quantization with NDVI-based multiple codebooks," *Proc. IEEE Int. Geosci. & Remote Sens.*, 2680–2684 (1998).
42. Manak, D., S.-E. Qian, A. B. Hollinger, and D. Williams, "Efficient hyperspectral data compression using vector quantization and scene segmentation," *Canadian J. Remote Sens.* **24**, 133–143 (1998).
43. Qian, S.-E., A. B. Hollinger, D. Williams, and D. Manak, "Vector quantization using spectral index based multiple subcodebooks for hyperspectral data compression," *IEEE Trans. Geosci. Remote Sens.* **38**(3), 1183–1190 (2000).
44. Sellers, P. J. et al., "The Boreal Ecosystem-Atmosphere Study (BOREAS): An Overview and Early Results," *22nd Conference on Agricultural and Forest Meteorology*, 1–4 (1996).

45. Qian, S.-E. and A. Hollinger, "System and Method for Encoding/ Decoding Multi-dimensional Data Using Successive Approximation Multistage Vector Quantization (SAMVQ)," *U.S. Patent No. 6,701,021 B1, issued on March* **2**, 2004.
46. Qian, S.-E. and A. Hollinger, "Method and System for Compressing a Continuous Data Flow in Real-Time Using Cluster Successive Approximation Multistage Vector Quantization," *U.S. Patent No. 7,551,785, issued on June* **23**, 2009.
47. Qian, S.-E. and A. Hollinger, "System and Method for Encoding Multi-dimensional Data Using Hierarchical Self-Organizing Cluster Vector Quantization (HSOCVQ)," *U.S. Patent No. 6,724,940 B1, issued on April* **20**, 2004.
48. Qian, S.-E. and A. Hollinger, "Method and System for Compressing a Continuous Data Flow in Real-Time Using Recursive Hierarchical Self-Organizing Cluster Vector Quantization," *U.S. Patent No. 6,798,360 B1, issued on September* **28**, 2004.

Chapter 5
Onboard Near-Lossless Data Compression Techniques

5.1 Near-Lossless Satellite Data Compression

The vector-quantization-based lossy compression algorithms discussed in Chapter 4 can easily achieve compression ratios of 50:1 or more if some loss in fidelity of the reconstructed data can be tolerated in exchange for the higher compression ratio. Caution must be taken when a lossy data compression algorithm is applied to satellite data. For example, hyperspectral data contains rich spectral information for remote sensing applications. If a hyperspectral datacube is compressed using a lossy method, any information loss due to the compression can reduce the value of the data. Conventional lossy compression methods developed for 2D or 3D images are not suitable for hyperspectral imagery because they were not designed to preserve the spectral information in hyperspectral imagery.

A scientific dataset acquired by a satellite is not noise free. It contains all kinds of instrument noise, such as thermal noise, shot noise, salt-and-pepper noise, quantization noise, etc. The thermal noise is caused by the detector array and amplifiers of the instrument, and is independent of the signal intensity. The shot noise of the detector array is dependent on signal intensity; it is caused by statistical quantum fluctuations, that is, variation in the number of photons sensed at a given exposure level. Shot noise is proportional to the square root of the signal intensity, and the noises at different pixels of the detector array are independent of one another. Shot noise follows a Poisson distribution. In addition to photon shot noise, there can be additional shot noise from the dark leakage current in the detector array. This noise is sometimes known as dark-current shot noise. The salt-and-pepper noise is impulsive noise that can be caused by analog-to-digital converter errors. The quantization noise is caused by quantizing the analog electronic signal of the sensed pixels to digital counts; it has an approximately uniform distribution and can be signal dependent. Due to the existing instrument noise, scientific

datasets acquired by a satellite instrument have a SNR that quantifies how much the signal has been corrupted by the noise.

In addition, raw digitized satellite datasets need to be processed before they are delivered to a user community to derive application products. Raw datasets need to be converted to radiance data in the radiometric calibration process to remove all of the artifacts caused by the instrument and the atmosphere. This process introduces uncertainty or errors to the scientific datasets. After that, the radiance data is corrected to remove atmospheric effects and converted to reflectance data. The atmospheric correction is another source of introducing errors to the scientific datasets.

This chapter defines all of the noises (from different sources: instrument noise, calibration-process, and atmospheric correction) contained in an original satellite dataset as "intrinsic noise" for the purpose of distinguishing the noise (errors) introduced by a compression algorithm.

In order to preserve the scientific information of satellite data, a lossy compression algorithm should be designed to restrict the error introduced during the compression process to a level consistent with or lower than the level of the intrinsic noise of the original data. This kind of lossy compression is defined as "near lossless," as this level of compression error is expected to have a small-to-negligible impact on remote sensing applications of the satellite data compared with the intrinsic noise.

This chapter describes two near-lossless VQ-based compression techniques for onboard processing: cluster SAMVQ and recursive HSOCVQ.[1] As described in Chapter 4, these two compression techniques are capable of controlling the compression error introduced in the compression process to the level consistent with that of the intrinsic noise of the original datasets by setting the compression fidelity threshold to a value smaller than the intrinsic noise. The compression errors introduced by these two compression techniques are expected to have a small-to-negligible impact on remote sensing applications in comparison with the intrinsic noise of the original data. This kind of near-lossless compression is different from the visually near-lossless compression[2,3] for medical images and the virtual near-lossless compression[4,5] for hyperspectral imagery.

5.2 Cluster SAMVQ

5.2.1 Organizing continuous data flow into regional datacubes

Unlike ground applications where a full datacube is available in advance, in real-time compression aboard a hyperspectral satellite, only a 2D focal plane frame (i.e., 2D detector array, as shown in Fig. 1.7) and the earlier frames are available at a given moment when the satellite looks at the cross-track line on the ground. One dimension of the focal plane frame corresponds to the

ground sample cells in the cross-track line, and another dimension of the frame corresponds to the spectrum expansion of each ground sample in the wavelength dimension. The spectrum expansion of a ground sample is referred to as a spectrum or spectral vector herein. There are N_c spectra (i.e., spectral vectors) in a focal plane frame if a cross-track line has N_c ground samples. The second spatial dimension of a hyperspectral datacube is obtained by the satellite flight in the along-track direction. This operating concept of a hyperspectral sensor is illustrated in Fig. 1.7. A series of such 2D frames collected in a certain period of time covers an instantaneous scene on the ground (referred to as a region herein) and can be treated as a real-time regional datacube for the purpose of dividing continuous 2D frame series into manageably sized datacubes.

Because fixed-size datacubes are unavailable in advance aboard a satellite, real-time compression of hyperspectral imagery has to be carried out region-by-region by organizing the continuous data flow of 2D frames into regional datacubes. Figure 5.1 illustrates how a series of continuous 2D focal plane

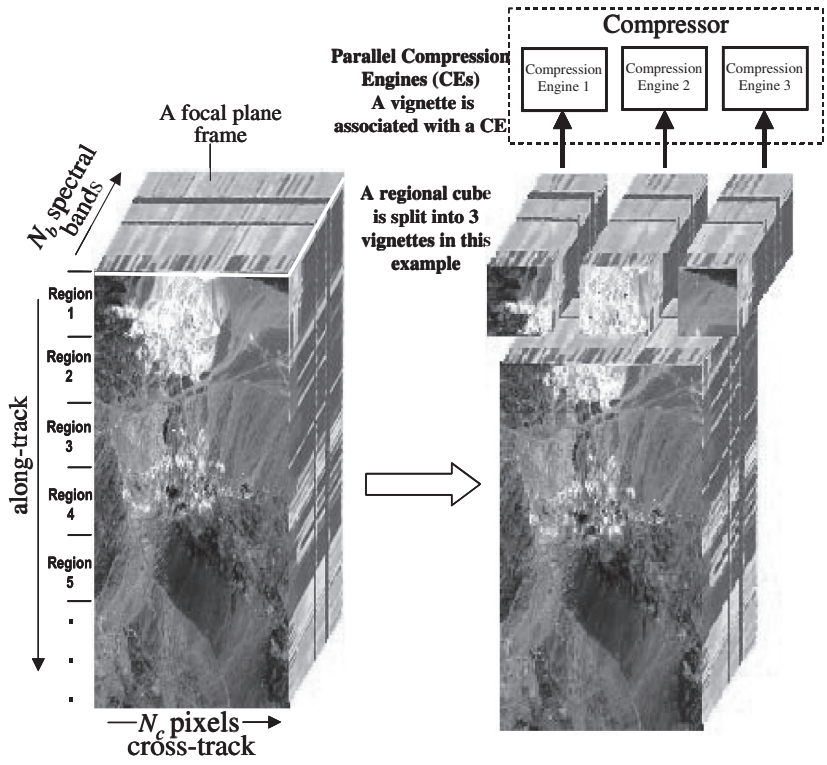


Figure 5.1 Illustration of (a) organizing continuous 2D focal plane frames into regional datacubes and (b) splitting a regional datacube into vignettes to facilitate parallel hardware implementation. For a color version of this figure, see Plate 4 in the color plate section of this book.

frames are organized into a regional datacube on the left graph and how a regional datacube is split into multiple vignettes (subdatacubes) on the right graph for facilitating parallel processing to overcome the constraints of the CPU processing and memory limitation. In the figure, a regional datacube is split into $M = 3$ subdatacubes. In the real case, the number of subdatacubes to be split depends on the volume of a regional datacube, the compression processing throughput, and the memory capacity. Each compression engine (CE) compresses a subdatacube independently. This approach solves the problem of limited processing speed and memory capacity in real-time data compression aboard a satellite.

5.2.2 Solution for overcoming the blocking effect

However, visible spatial quilts (blocks) occur within a regional datacube because the compression of each subdatacube is compressed independently by each compression engine. Figure 5.2 shows an example of the blocking effect of the difference image between the original and the reconstructed datacubes when a datacube is split into subdatacubes for parallel processing and when each subdatacube is compressed by a compression engine independently. This blocking effect of the compressed datacube is not acceptable.

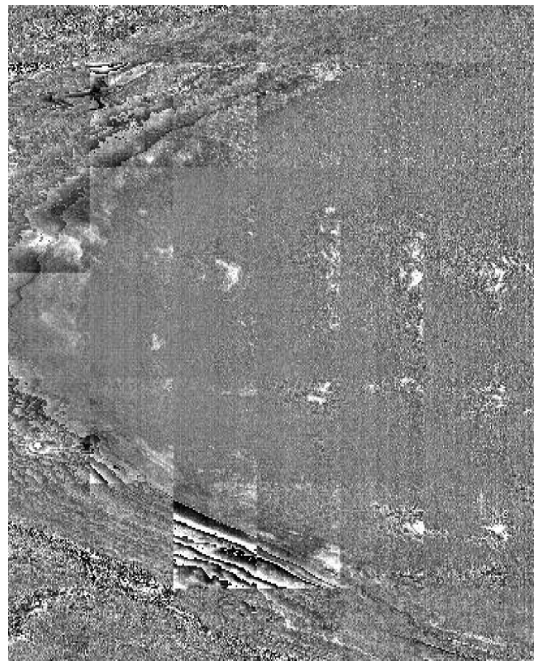


Figure 5.2 An example of spatial quilt in the difference image between the original and the reconstructed datacube when a datacube is split into subdatacubes for parallel processing and when each subdatacube is compressed by a compression engine independently.

To overcome this blocking effect, an innovative method referred to as "cluster SAMVQ" herein has been developed.[6] It separates a regional datacube into manageable subsets for parallel processing by classifying a regional datacube into M clusters based on the similarity of spectra within the datacube rather than cutting the datacube into M vignettes. Similar spectra of the ground samples are more likely to be associated with the same clusters of particular targets in the scene (e.g., vegetation and water body, etc.). A subset consists of a cluster of spectra that are similar but not associated with specific locations across track. This method not only gets rid of the cross-track boundaries in a regional datacube but also slightly improves the performance of the compression. Because the spectra in a cluster are similar, they can be more easily compressed. Fewer codevectors or fewer approximation stages are required to attain the same reconstruction fidelity as that achieved using the vignette approach. Thus, a higher compression ratio can be achieved if the fidelity remains constant.

The classification processing is a preprocessing step that classifies a regional datacube to form subsets and distributes them to each CE for compression using the SAMVQ technique. A complex classification processor is not necessary, as the purpose is to divide a regional datacube into subsets without inducing spatial blocking effects and to facilitate parallel processing. Any existing classification method can be used to divide a regional datacube into clusters. The classification method chosen is called the spectral vector partition algorithm; it classifies spectral vectors in a regional datacube into partitions based on the minimal distances between the spectral vectors and the centroids of the partitions, which is similar to the codebook training. This method is simple, fast, and easy to implement in hardware.

The sizes of the clusters yielded by the selected classification method are not the same. Some of them are large, and others are small. This is because the size of the clusters depends on the scene of the datacube to be compressed. However, it is desirable to have approximately equal sizes for each cluster in order to make full use of the hardware capacity of each CE. For example, a CE designed to compress a set of 4096 spectral vectors in a parallel compression system would not be fully used if the size of the cluster assigned to the CE were much smaller than 4096 spectral vectors. There is also an upper limit because the CE cannot handle a cluster if its size is larger than 4096 spectral vectors. A classification method that can adaptively control the size of each cluster was developed so that they are approximately equal in size by splitting a large cluster and merging small clusters during the course of classification.

5.2.3 Removing the boundary between adjacent regions

The solution described in Section 5.2.2 eliminates the blocking effect occurring within a regional datacube. However, there is still a boundary between two adjacent regions in the along-track direction, as compression is

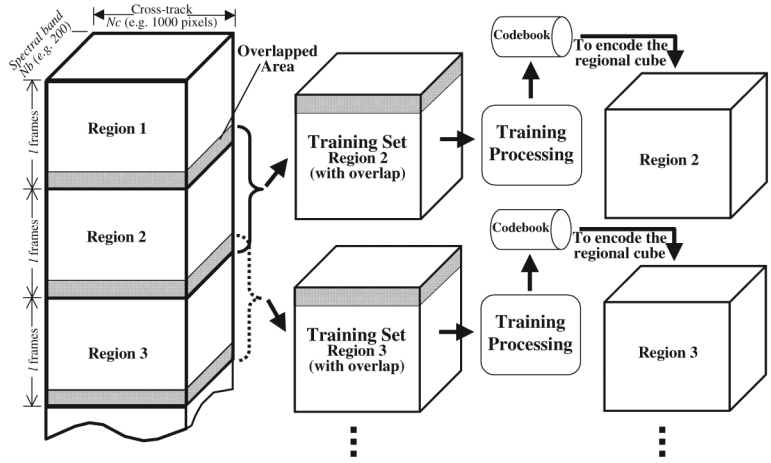


Figure 5.3 Overlap of two adjacent regions for codebook training to remove the boundary between two regions.

performed independently region by region. In order to overcome this problem, a method of overlapping two adjacent regions for codebook training has been introduced. A number of cross-track lines from the previous region, which are closest to the current region, are selected and included in the training set used for the current region during codebook training process, as shown in Fig. 5.3. Because of the correlation and similarity among the spectral vectors of the current region and those from the overlapped area of the previous region, the codevectors trained for the current region will be highly correlated with the codevectors trained in the previous region, especially in the juncture area of the two regions. No boundary occurs between the two adjacent regions when the codevectors trained in this way are used to encode the spectral vectors of the current region.

5.2.4 Attaining a fully redundant regional datacube for preventing data loss in the downlink channel

The data loss due to a single-bit event in the downlink channel is an issue in the development of spaceborne hyperspectral imagers, especially when an onboard data compressor is used. Data is more sensitive to bit errors after it is compressed. Compressed data of a regional datacube is encapsulated into source packages and ultimately placed in multiple transfer frames before it is transmitted via the downlink channel to the ground. Single-bit errors can cause a transfer frame to be corrupted or lost; if one occurs in a transfer frame that contains the index map and/or codebook, the reconstructed data for that regional datacube can be subject to error. If that transfer frame is lost, then the regional datacube is lost.

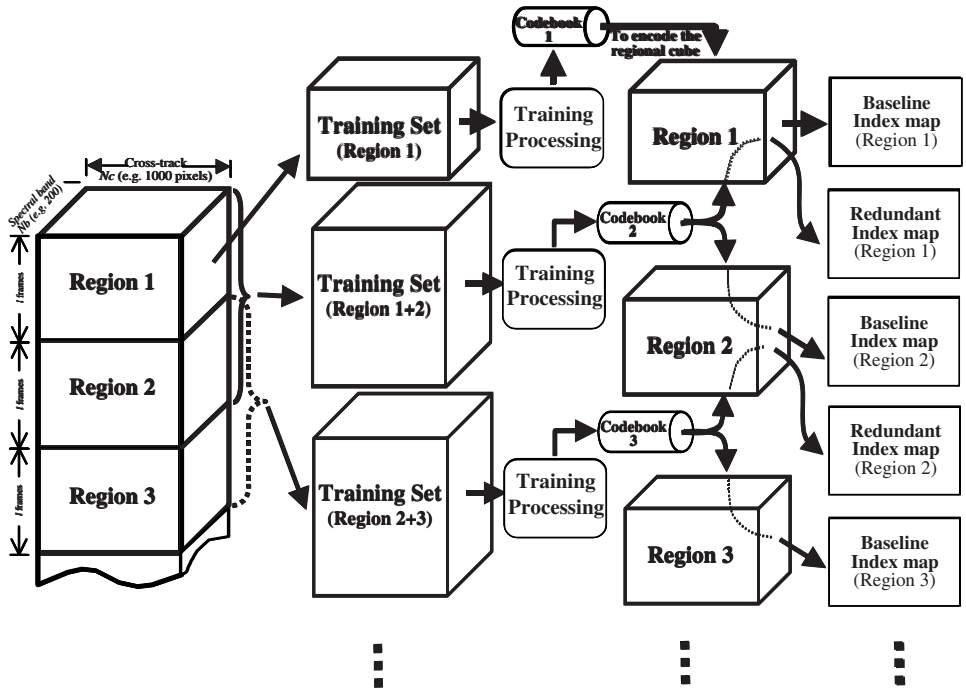


Figure 5.4 Combining a current region and an entire previous region to attain a full redundancy.

Full redundancy of a regional datacube is obtained when the previous region and current region are combined as the training set for codebook training. The codebook trained from the combined regions can be used to encode both regions. Figure 5.4 illustrates how a combined training set is formed and how a codebook is created for regions $1 + 2$, $2 + 3$, etc. A regional datacube is associated with two codebooks. Region 1 is associated with codebook 1 and 2, region 2 with codebook 2 and 3, and so on. In this way, a regional datacube has a fully redundant codebook. There is no penalty to the compression ratio for this redundancy, as the number of codevectors remains the same. In order to attain a redundant index map, a current codebook is used to encode the current region as well as the previous region to produce a baseline index map for the current region and a redundant index map for the previous region. In the example, the codebook 2 (trained from regions $1 + 2$) creates a baseline index map for region 2 and a redundant index map for region 1. This doubles the size of the index map. The overhead of the compression ratio is very low because an index map is usually a tenth of the size of a codebook.

This method attains full redundancy for a region. An entire regional datacube can be perfectly restored if its codebook or index map are corrupted

or lost due to single-bit errors. This feature is extremely important to protect against such errors aboard a satellite or in the downlink channel. Because the compressed data (i.e., codebook and index map) is fully redundant, the reconstruction process is less sensitive to any errors, including single-bit errors that could be introduced after applying a lossless compression algorithm to the compressed data. This method enables the application of a lossless compression algorithm to the codebooks and index maps to further increase the compression ratio.

It is observed that in SAMVQ compression, the amplitude of the elements of the codevectors generated in each stage decreases with the increasing number of stages. For example, in the early stages, the amplitude of a codevector may occupy the most-significant bits, whereas the amplitude of a codevector after the second stage may occupy only the four least-significant bits, even though the image dataset to be compressed has a 16-bit dynamic range. In cluster SAMVQ, this fact is taken into account by storing the codebook generated in a stage. The word-length of a codebook generated in a stage is not fixed. Two schemes are utilized to encode a stage codebook: (1) using the shortest-fitting word-length for each codevector of the stage codebook, and (2) using the shortest-fitting word-length for all of the codevectors of the stage codebook. Both schemes save storage memory for codebooks in the multistage VQ and help increase the compression ratio while retaining the same fidelity. More importantly, the codebooks encoded in this way have much-higher bit-error immunity and protect against bit-error propagation that can be caused by single-bit flips. For example, assume that a codebook generated for stage 5, occupying only the 2 least-significant bits (i.e., the amplitude dynamic range of 0–3), has 1 bit of a codevector within the codebook flipped; the amplitude of the corrupted codevector that was encoded using the conventional SAMVQ approach would become between 32,768 (i.e., 2^{15}) and 32,771 if the most-significant bit of the codevector is flipped. In cluster SAMVQ, with the same assumption, the amplitude of the corrupted codevector will be between 0 and 3. This scheme greatly reduces the amplitude of the resulting error propagation.

5.2.5 Compression performance comparison between SAMVQ and cluster SAMVQ

Table 5.1 shows an example of the compression performance comparison of cluster SAMVQ with conventional SAMQ that divides a datacube into vignettes. A BOREAS datacube acquired using the CASI hyperspectral sensor was used as test data. The test datacube is in a raw digital number (DN) with 12-bit resolution (data range: 0–4024). The datacube size is 405 pixels in the cross-track direction by 2852 lines by 72 spectral bands (file size 166 MB). The experimental results show that cluster SAMVQ attains better reconstruction

Table 5.1 Comparison of compression performance between cluster SAMVQ and conventional SAMVQ using vignette splitting approach.

	SAMVQ (with vignettes)			Cluster SAMVQ		
Compression ratio	20:1	30:1	50:1	20:1	30:1	50:1
PSNR (dB)	55.8	51.30	47.14	57.5	54.72	51.57
RMSE	5.51	11.00	17.68	5.38	7.34	10.62

fidelity than SAMVQ with vignettes when the compression ratios remain constant.

5.3 Recursive HSOCVQ

The SAMVQ and HSOCVQ methods described in Chapter 4 were designed for different applications. The HSOCVQ algorithm merges the codebook training and coding phases into one. Given a desired compression fidelity threshold (such as PSNR or a RMSE), HSOCVQ compresses the spectral vectors in a datacube until each of them is encoded with fidelity better than the threshold. It is suitable for preserving single- or small-population spectral signatures, whereas SAMVQ compresses data in a multistage approximation manner by checking the fidelity of the entire dataset stage-by-stage until the overall fidelity of the compressed data reaches the fidelity threshold at an approximation stage. The spectral signature carried by a single sample or small population samples in a scene may not be preserved. Normally, SAMVQ produced relatively higher PSNRs of the reconstructed data compared to HSOCVQ for the same compression ratio.

Similarly, for onboard real-time data compression using HSOCVQ, the continuous data flow of the 2D focal plane frames also needs to be organized into regional datacubes and compressed one after another. In order to deal with the similar problems that occurred in onboard real-time data compression, the conventional HSOCVQ method has been updated. A novel onboard, real-time data-compression technique called recursive HSOCVQ has been developed.[7]

5.3.1 Reuse of codevectors of the previous region to attain a seamless conjunction between regions

As described in Chapter 4, HSOCVQ first trains an extremely small number of tentative codevectors (usually $N_1 \leq 8$) using the datacube to be compressed as the training set and then uses these tentative codevectors to classify spectral vectors in the datacube into N_1 clusters. It then encodes the N_1 clusters one after another. This is equivalent to the cluster SAMVQ classification of a regional datacube into M clusters based on the similarity of spectra within the datacube to solve the blocking effect. HSOCVQ, by its nature, divides a

datacube to be compressed into multiple small-size clusters. There will be no blocking effect when HSOCVQ is applied to compress a regional datacube. This is another difference between HSOCVQ and SAMVQ. In onboard real-time compression, HSOCVQ only needs to find a way to overcome the boundary between the adjacent regions.

To overcome the boundary between the adjacent regions, our innovative approach is to reuse the codevectors trained for the previous region to encode the spectral vectors in the current region to attain a seamless conjunction of the two adjacent regions. The last few frames (especially the last frame) of the previous region and the first few frames (especially the first frame) of the current region are referred to as the boundary area. Because the boundary is artificial, spectral vectors in the boundary area must be similar. The same codevectors are used to encode the spectral vectors of both regions in the boundary area. No visible spatial boundary between the two regions will occur because the same codevectors are used. The reused codevectors of the previous region are carried forward to the next region.

5.3.2 Training codevectors for a current frame and applying them to subsequent frames

In real-time onboard VQ compression, codevectors have to be trained for a relatively small instantaneous scene (or region) in order to reduce the requirements for CPU, memory, complexity, power consumption, and volume of the system. In recursive HSOCVQ, a single 2D focal plane frame acquired in a moment by a hyperspectral sensor is used to train codevectors for compression of that frame. When the next frame data comes, new codevectors are trained by using this frame as the new training set. In other words, recursive HSOCVQ trains codevectors and compresses data frame by frame. This method greatly simplifies hardware implementation and requires less memory for onboard operation. As shown in Fig. 5.5, for example, the spectral vector of a ground sample cell in a focal plane frame has 200 bands with 12-bit data resolution. The memory required to accommodate a frame data is only 1000 such spectral vectors (a total of $1000 \times 200 \times 12$ bits); this amount of data can be easily accommodated. However, this method greatly reduces the ability of HSOCVQ to compress data because it does not benefit from using the correlation that exists among spectral vectors within a big scene.

In order to benefit from the correlation that exists among spectral vectors of the ground samples within a scene, recursive HSOCVQ reuses codevectors trained for a current frame in the subsequent frames to increase codevector efficiency and save training time. Correlation is the strongest between the spectral vectors in the current frame and those in the next frame. Codevectors trained for the current frame can be used to encode most of the spectral

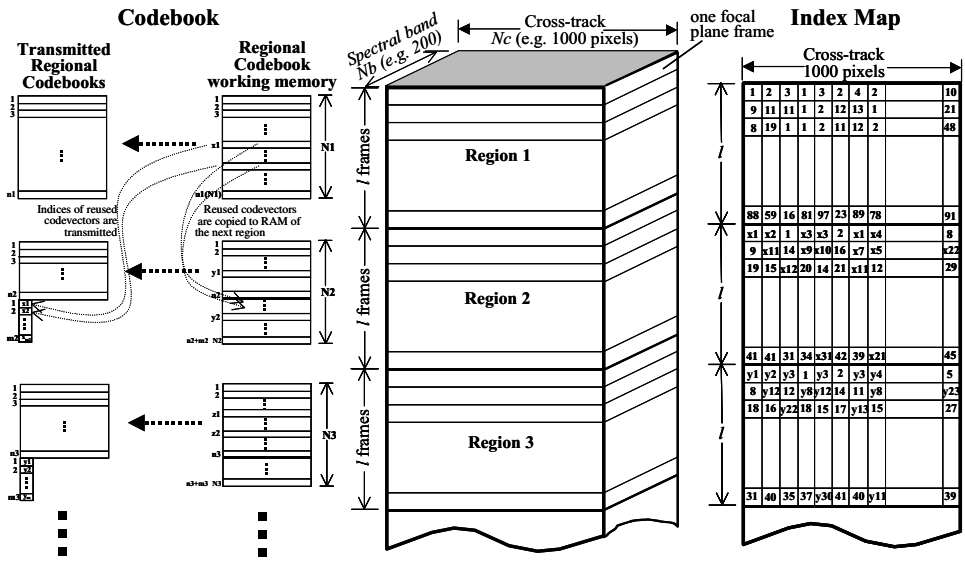


Figure 5.5 Illustration of recursive HSOCVQ generating an associated codebook in each region with reuse of codevectors trained for the previous region.

vectors in the next frame at the same desired fidelity threshold. Correlation decreases with the separation of the subsequent frames from the current frame; that is, fewer codevectors trained for the current frame are used to encode the spectral vectors in widely separated frames. The number of codevectors that are reused depends on the desired fidelity threshold. This method has following advantages:

1. Codevector training is extremely fast because the training set is very small (a single frame rather than a datacube).
2. Codevectors trained in this way are used very efficiently. Codevectors are used not only to encode the spectral vectors in the frame from which they are trained but also to encode the spectral vectors in the subsequent frames.
3. It is a recursive approach to encoding a large datacube by training only a small portion of data.

Because reusing codevectors leaves some spectral vectors in the subsequent frames that cannot be encoded at the desired fidelity, these spectral vectors need to be encoded separately. Given the small number of uncoded spectral vectors in the next frame (as most of the spectral vectors have been encoded), the uncoded spectral vectors cannot be compressed efficiently. Not only is there little correlation among the uncoded spectral vectors but also experimental results show that the population of the uncoded spectral vectors derived from subsequent frames is too small to be

trained and encoded. As such, only a compression ratio between 1:1 and 2:1 could be achieved if they were encoded immediately. The approach presented here collects all uncoded spectral vectors in each of the subsequent frames and then encodes them at the end of a regional datacube, after a preset number of frames is reached, or after a sufficient population of uncoded spectral vectors is gathered. In this way, a sufficiently high correlation among the spectral vectors can be accumulated when the spectral vectors are encoded, and a relatively high compression ratio can be attained.

5.3.3 Two schemes of carrying forward reused codevectors trained in the previous region

Two schemes of carrying forward the reused codevectors have been proposed and implemented in recursive HSOCVQ. The first scheme transmits only the index of each reused codevector together with the codevectors trained for a current regional datacube. Figure 5.5 illustrates the concept of this scheme. In the figure, the leftmost column shows the regional codebooks transmitted. The second-leftmost column shows the working memory of the regional codebooks. Codevector x1 is reused in the next region. Only its index (or address) in that codebook "x1" (8 bits, assuming a codebook size of 256 codevectors) is transmitted rather than the entire codevector (size is 200×12 bits, assuming that a spectral vector has 200 bands with 12-bit data resolution). In the decompression process on the ground, its recorded index (x1) will point to the corresponding codevector in the previous regional codebook. In this way, all of the reused codevectors need not be transmitted, which greatly increases the compression ratio because an index uses only 1 byte rather than using 300 bytes for a codevector. This scheme is referred to as the "associated codebook" scheme because the recursive decompression process requires a current regional codebook and the previous regional codebook (the current regional codebook is always associated with that of the previous region).

Upon completing the compression of the current region, all of the reused codevectors from the previous region are copied and appended to the working memory of the current regional codebook. In this way, all codevectors used for the current region are organized into a whole codebook in order to facilitate the processing of the next region. In the compression of the next region, all of the codevectors residing in the working memory of the region (whether trained or carried forward) will be checked to verify if they can be reused to encode the new region.

The second scheme carries a codevector from the previous region forward to the working memory of the current regional codebook as soon as it is reused. Upon completing the compression of the current region, the codevectors in the working memory of the current region are a mixture of

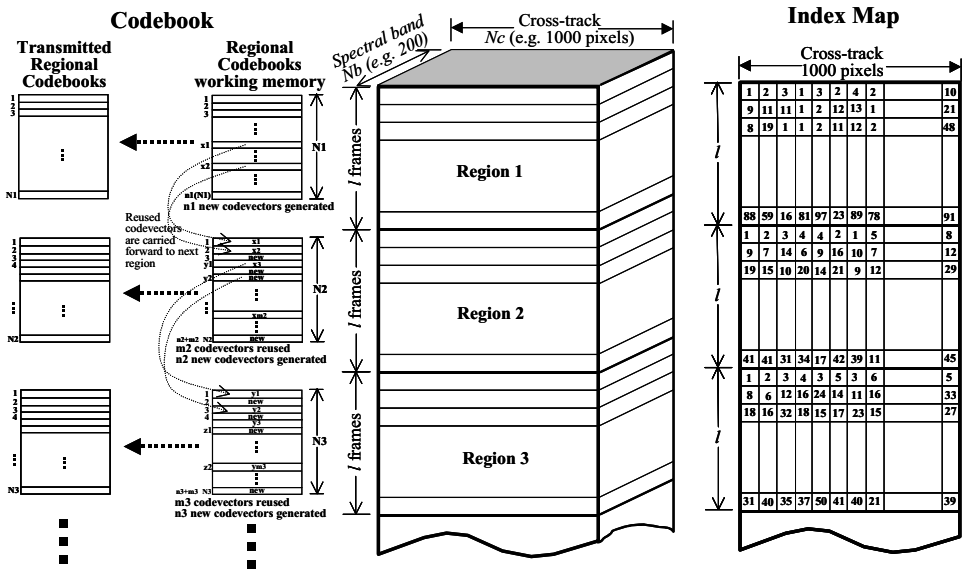


Figure 5.6 Illustration of recursive HSOCVQ generating an independent codebook in each region with reuse of codevectors trained for the previous region.

reused and newly trained codevectors. All of the codevectors in the working memory are transmitted. In this scheme, the codebook of each region is distinct and independent, as all of the reused codevectors are already included. Figure 5.6 illustrates this scheme. Reused codevectors "x1" and "x2" are carried forward to the working memory of the current region as soon as they are reused and immediately become the codevectors of the new regional codebook. In the decompression process on the ground, a region that is reconstructed requires only the codebook of that region. Although this scheme does not make use of the redundancy of the reused codevectors in the codebooks of the two neighboring regions, it has the advantage of protecting against error propagation. The errors will be constrained to a regional datacube if bit errors occur within that codebook or index map.

The order of reusing existing codevectors is important to achieve high compression performance. When spectral vectors in a current region are encoded, there are two kinds of codevectors that exist and can potentially be used: current-region, newly trained codevectors for the previous frames, and codevectors in the previous regional codebook. As mentioned in Section 5.3.2, HSOCVQ trains codevectors frame by frame. If a spectral vector in the current frame can be encoded by a reused codevector, it is from one of the two sources. Once a spectral vector is successfully encoded from one source, codevectors in another source will not be searched.

Experimental results show that the codevectors newly trained from the current region should be checked first, followed by those from the previous regional codebook. This order produces better compression performance for three reasons:

1. The codevectors newly trained from the frames of the current region have higher correlation to spectral vectors in the current frame than those trained for the previous region. They yield better fidelity when they are reused to encode spectral vectors of the current frame.
2. If a spectral vector can be encoded by a codevector from the newly trained codevectors of the region, there is no need to search the codebook of the previous region. This will save compression time for encoding the spectral vector because the size of the current regional codebook is normally small.
3. If a spectral vector could be encoded equally well using either a codevector from the current or previous region, the former is preferred because the overhead of transferring a codevector from the previous region is removed. Otherwise, the reused codevector in the previous regional codebook needs to be carried forward to the current regional codebook. This will increase the size of the current regional codebook and reduce the compression ratio.

5.3.4 Compression performance comparison between baseline and recursive HSOCVQ

Table 5.2 shows an example of the compression performance comparison of baseline and recursive HSOCVQ for hardware implementation. A radiance datacube of the Cuprite mining district of Nevada was used as test data; it was acquired using the AVIRIS hyperspectral sensor in June 1996, with an approximately 20-m ground resolution in 224 spectral bands, each about 10 nm wide, in the 400–2500-nm wavelength range, with a spatial size of 614 pixels by 512 lines (12.3 km $\times$ 10.2 km). The data file size is 140 MB. The experimental results show that recursive HSOCVQ modified to facilitate parallel hardware performs as well as baseline HSCOVQ, although the correlation that benefits compression is greatly reduced due to the training in a single frame.

Table 5.2 Comparison of compression performance between baseline and recursive HSOCVQ.

	HSOCVQ				Recursive HSOCVQ			
Compression ratio	10:1	20:1	30:1	40:1	10:1	20:1	30:1	40:1
PSNR (dB)	58.35	56.2	55.23	54.64	57.95	56.1	55.05	54.32
SNR (dB)	41.95	39.83	38.83	38.12	41.55	39.66	38.62	37.92
RMSE	39.64	50.57	56.67	60.65	41.50	51.58	58.13	63.06

5.4 Evaluation of Near-Lossless Performance of SAMVQ and HSOCVQ

5.4.1 Evaluation method and test dataset

One of the unique features of SAMVQ and HSOCVQ allows control of the errors introduced during the compression process. For remote sensing applications, compression is considered near lossless provided that the error introduced by the compression is not larger than the intrinsic noise in the original data caused by the instrument noise and other noise sources from preprocessing, such as calibration and atmospheric correction. Because an original datacube is not exempt from intrinsic noise and other noise sources, these errors propagate into the remote sensing products derived from the original data.

In order to evaluate the near-lossless feature of the two compression techniques, the error introduced by the two compression algorithms was analyzed band by band and pixel by pixel. The compression errors were compared with the intrinsic noise of the original data to see if they were consistent with the level of intrinsic noise in the original data.

A low-altitude AVIRIS dataset acquired in the Greater Victoria Watershed District, Canada on August 12, 2002 was used (information on the dataset is available at http://aviris.jpl.nasa.gov/ql/list02.html). The ground sample distance (GSD) of the dataset is 4 m $\times$ 4 m with an AVIRIS nominal SNR of 1000:1 in the visible and near-infrared (VNIR) region. A spectral subset was selected to remove redundant and bad bands, which reduced the data from 224 bands to 204 bands, including the original bands 6–31 (423.04–664.79 nm, VNIR), bands 35–96 (673.64–1258.39 nm), and bands 98–213 (1263.72–2399.48 nm) for the short-wavelength infrared. A 28 m $\times$ 28 m GSD datacube was derived by spatially aggregating the 4 m $\times$ 4 m GSD dataset. The SNR of the aggregated datacube is $1000 \times \sqrt{49} = 7000{:}1$. Figure 5.7(a) shows the aggregated datacube, whose spatial size is 292 lines with 121 pixels per line. The datacube is encoded in 16-bit digital numbers (DNs). This datacube is considered noise-free because the noise is too small to have a significant impact on the evaluation. Figure 5.7(a) is a RGB image with bands 38 (702.2 nm), 20 (557.9 nm), and 2 (432.6 nm) being displayed as red, green, and blue.

Figure 5.7(b) shows a datacube identical to Fig. 5.7(a) except that its SNR is 600:1, generated by adding simulated instrument noise and other possible noise sources to (a). An additive noise (Gaussian model) was used. This noise-added datacube was used as an original datacube for compression and for evaluation of the near-lossless feature of the compression algorithms here; it is considered to be representative of a real satellite hyperspectral dataset because the SNR for such an instrument is likely to be approximate to that level.[2] The noise of the original datacube caused by the instrument noise and other noise

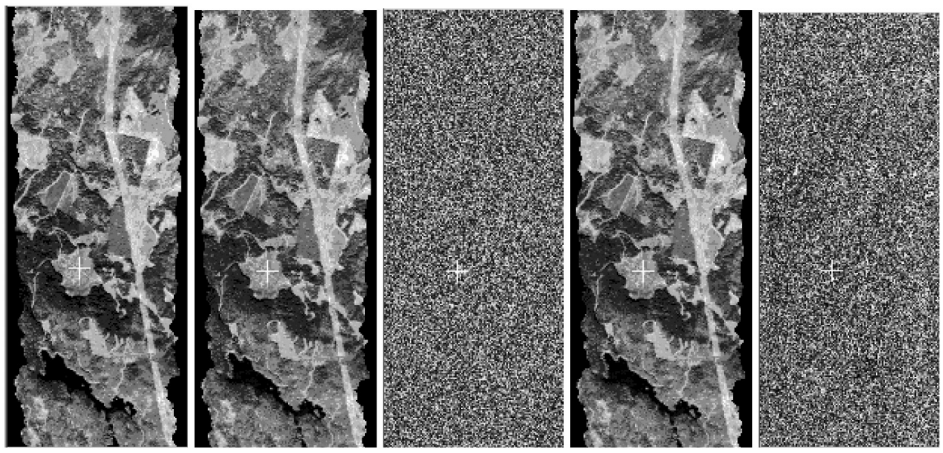


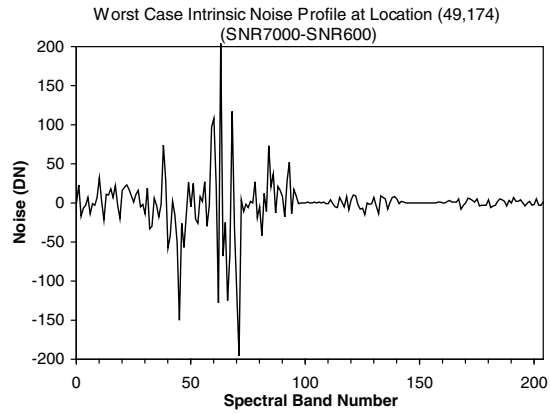
Figure 5.7 AVIRIS Greater Victoria Watershed District datacube: (a) noise-free datacube, (b) noise-added datacube (uncompressed), (c) intrinsic-noise datacube, (d) compressed datacube, and (e) compression-error datacube (reprinted from Ref. 1). For a color version of this figure, see Plate 5 in the color plate section of this book.

sources is referred to as intrinsic noise in this section. Figure 5.7(c) shows an intrinsic noise image at band 38 (702.2 nm) of the noise datacube, which was obtained by subtracting the datacube with a SNR of 600:1 (b) from the noise-free datacube (a). The display scale is stretched to between the minimum (–155 DN) and maximum (149 DN) amplitude of the noise image for better contrast.

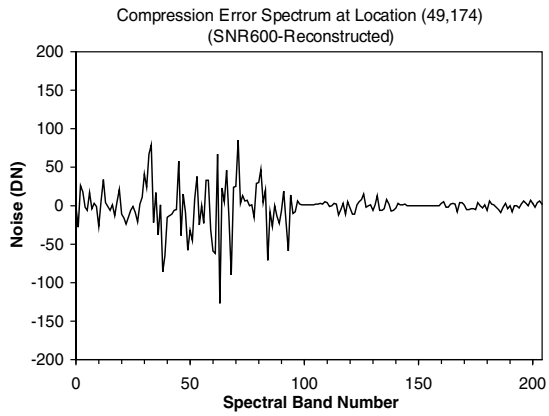
The original datacube was compressed using SAMVQ at a compression ratio 20:1, and then the compressed data was decompressed to obtain the reconstructed datacube, as shown in Fig. 5.7(d), for evaluation. It is difficult to visually distinguish the difference between the original and the reconstructed datacubes. Figure 5.7(e) shows the compression-error image (difference between the original datacube and the reconstructed datacube) at band 38. The display scale is stretched to the same range as Fig. 5.7(c). The pattern of the compression error image looks similar to that of the intrinsic noise image; also, there are no apparent structures.

5.4.2 Evaluation of a single spectrum

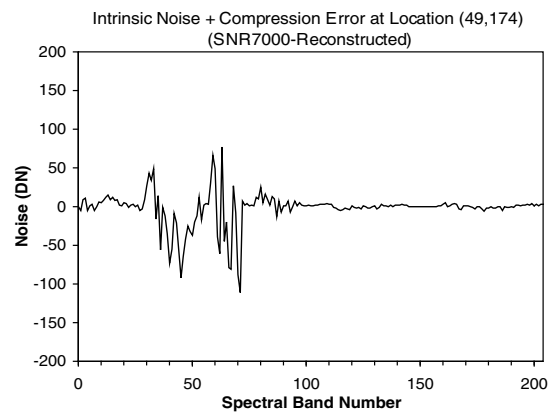
The intrinsic noise of the original datacube has been analyzed; Figure 5.8(a) shows the worst-case noise profile of a ground sample pixel at location (49, 174) of the intrinsic-noise datacube as a function of spectral band number. The noise magnitudes for the VNIR bands and the beginning of the SWIR bands are large. The maximum value of the noise is 204 DN at band 66 (1050.45 nm), and the minimum value of the noise is –196 DN at band



(a)



(b)



(c)

Figure 5.8 Profiles of (a) the worst-case intrinsic noise of the original data at location (49, 174), (b) compression error introduced by SAMVQ at a compression ratio of 20:1, and (c) overall noise (i.e., intrinsic noise with compression error) at the same location (reprinted from Ref. 1).

71 (1097.72 nm). The noise magnitudes are between –15 DN and 15 DN between bands 100 (1363.50 nm) and 204 (2399.48 nm).

Figure 5.8(b) shows the compression error (or noise) profile of the reconstructed datacube compressed by SAMVQ with a compression ratio of 20:1 at the same location as Fig. 5.8(a). The error introduced due to the compression is not larger than the intrinsic noise of the original datacube across the spectral bands. The maximum value of the compression error is 85 DN at band 71 (1097.72 nm), and the minimum value of the compression error is –127 DN at band 63 (1022.09 nm). The compression-error magnitudes between bands 100 (1363.50 nm) and 204 (2399.48 nm) are in the same range as the intrinsic noise of the original datacube: between –15 DN and 15 DN.

After compression/decompression, the reconstructed data contains both the intrinsic noise of the original datacube and the compression error (or compression noise). This section refers to the combination of intrinsic noise and compression error as overall noise, which is the final noise budget of the datacube if the reconstructed data is sent to a hyperspectral data user for deriving their products. Figure 5.8(c) shows the overall noise profile at the same location (49, 174); it was obtained by subtracting the spectrum of the reconstructed datacube from the spectrum of the noise-free datacube at the same location. Interestingly, the overall noise profile shows that the maximum value of the noise is reduced to 77 DN at band 63 (1022.09 nm), and the minimum value of the noise is increased –111 DN at band 71 (1097.72 nm). The magnitudes of the overall noise between bands 100 (1363.50 nm) and 204 (2399.48 nm) are reduced to between –5 DN and 5 DN rather than between –15 DN and 15 DN, as in the original datacube. The range of the overall noise magnitudes in the VNIR bands is also smaller than for the intrinsic noise of the original datacube, which is probably due to the random error introduced by the compression algorithm that cancelled the intrinsic noise in the original data. These results show that the VQ-based compression algorithm evaluated here can act as a low-pass filter, suppressing high-frequency noise during compression.[8]

Figure 5.9 shows the noise profiles of the intrinsic noise of the original data, compression error introduced by SAMVQ at a compression ratio of 20:1, and overall noise as a function of spectral band number for a randomly selected ground pixel at location (94, 90). The profiles of the compression error and the overall noise show better results than in Fig. 5.8.

5.4.3 Evaluation of an entire datacube

In order to assess the compression error of the entire reconstructed datacube, the standard deviation of each band image of the compression-error datacube was calculated and plotted as a function of spectral band number. These standard deviations were used to estimate the noise level of the compressed data. The standard deviations of each band image of the intrinsic noise and of

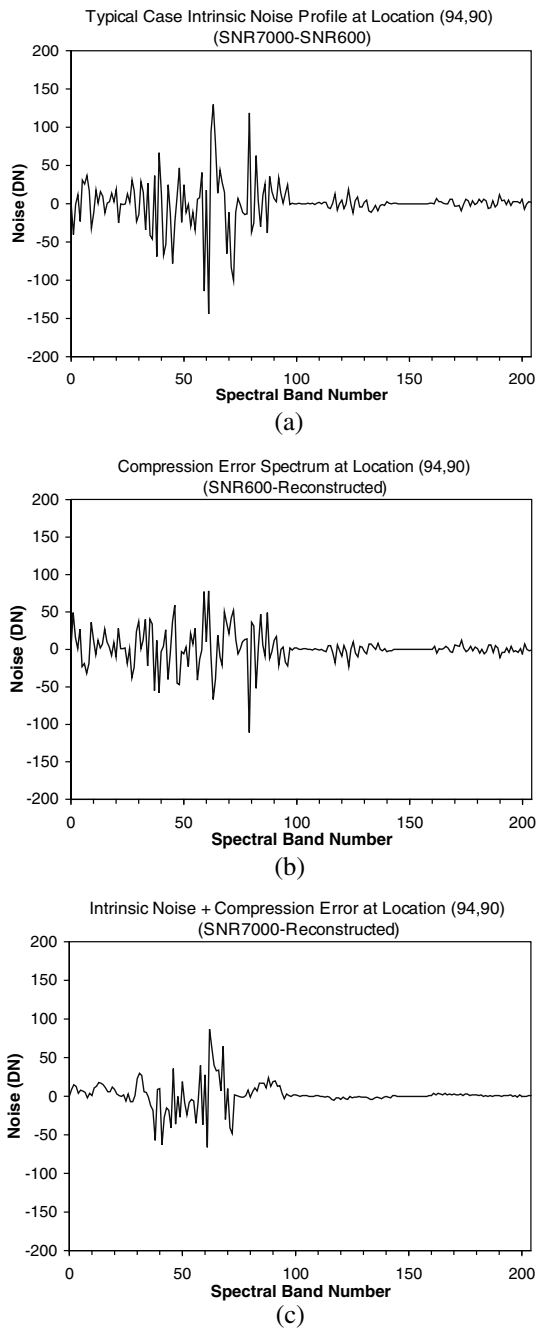


Figure 5.9 Profiles of (a) intrinsic noise of the original data of a randomly selected ground-sample pixel (94, 90), (b) compression error introduced by SAMVQ at a compression ratio of 20:1, and (c) overall noise (i.e., intrinsic noise with compression error) at the same location (reprinted from Ref. 1).

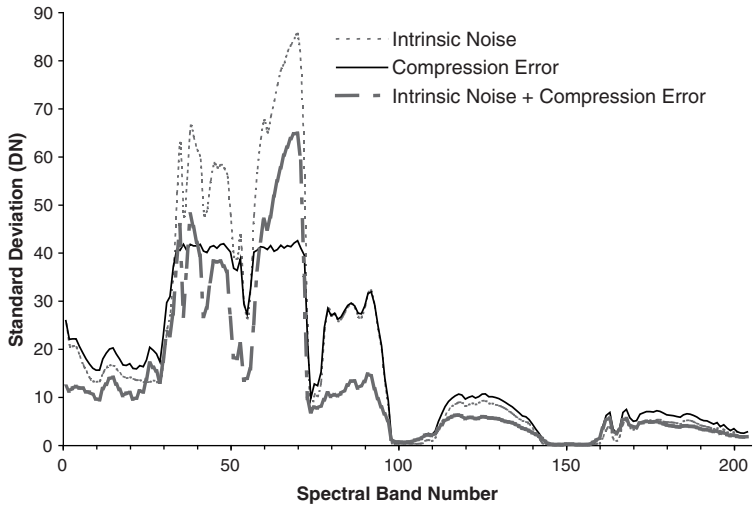


Figure 5.10 Standard deviations of single-band images for intrinsic noise, compression error (using SAMVQ at a compression ratio 20:1), and overall noise (intrinsic noise with compression error). Reprinted from Ref. 1.

the overall noise were also calculated for the purpose of comparison (shown in Fig. 5.10). It is observed that the standard deviations of the compression error images (solid line) are much smaller than those of the intrinsic noise images (dotted line) for the bands with high-magnitude noise [between bands 35 (749.96 nm) and 75 (1135.53 nm)]. The rest of the bands are very close. It is also observed that the standard deviations of the overall noise images (thick broken line) that include both the intrinsic noise and the compression error are smaller than those of the intrinsic noise images (dotted line) for almost all of the bands. This observation indicates that the overall noise level of the compressed datacube is even lower than the noise level of the original datacube, which is probably because SAMVQ acts as a low-pass filter, thus suppressing the high-frequency noise. The random errors introduced by the compression algorithm cancelled some of the intrinsic noise in the original data. The test results of standard deviations are consistent with the results of noise profiles shown in Figs. 5.8 and 5.9.

The noise profiles of the compression error and the overall noise of a single ground pixel compressed using HSOCVQ are similar to those compressed using SAMVQ, as shown in Figs. 5.8 and 5.9. Figure 5.11 shows standard deviations of band images for the intrinsic noise (dotted line), the compression error (solid line), and the overall noise (thick broken line) as a function of spectral band number when the datacube was compressed using HSOCVQ at 10:1. The standard deviations of band images of compression error are close to or smaller than those of intrinsic noise for the bands with high-magnitude noise [between bands 35 (749.96 nm) and 75 (1135.53 nm)].

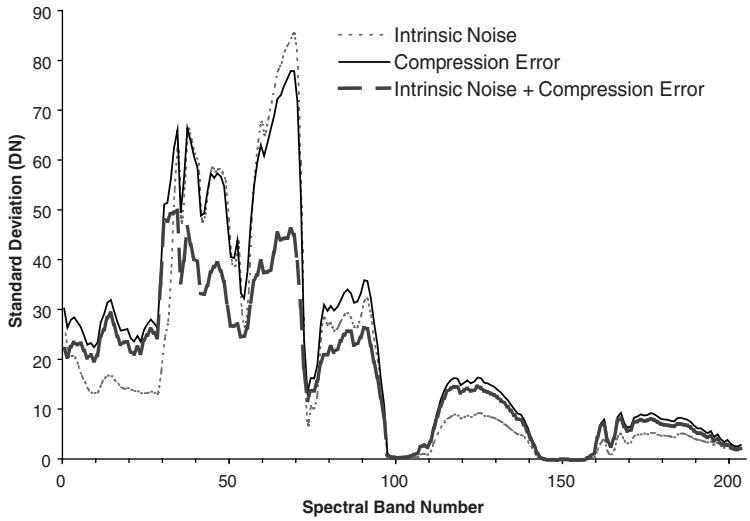


Figure 5.11 Standard deviations of single-band images for intrinsic noise, compression error (using HSOCVQ at a compression ratio 10:1), and overall noise (i.e., intrinsic noise with compression error). Reprinted from Ref. 1.

The standard deviations of the overall noise are smaller than those of the intrinsic noise between bands 35 and 105 (1413.37 nm).

The previous experimental results show that the compression errors introduced by SAMVQ and HSOCVQ are at the same level as the intrinsic noise caused by the instrument noise and other noise sources contained in the original datacube (such as calibration and atmospheric correction). The compression errors are smaller than the intrinsic noise in band images with high-magnitude noise. The experimental results justify the claim that SAMVQ and HSOCVQ are near lossless for remote sensing applications compared to the intrinsic noise of the original datacube. The noise contained in the reconstructed data is the overall noise that includes both the intrinsic noise of the original datacube and the compression error. The overall noise of the reconstructed datacube is even smaller than the intrinsic noise for all of the bands when the data is compressed using SAMVQ and for most of the bands when the data is compressed using HSOCVQ. Statistically, SAMVQ shows better near-lossless performance than HSOCVQ.

5.5 Evaluation of SAMVQ with Regard to the Development of International Standards of Spacecraft Data Compression

The Consultative Committee for Space Data System (CCSDS) is developing new international standards for satellite multispectral and hyperspectral data compression, and the SAMVQ technique has been selected as a candidate.

This section reports the evaluation results of SAMVQ in the participation in the development of the international standards. The evaluation results show that SAMVQ produces competitive rate-distortion performance on the CCSDS test images acquired by the hyperspectral sensors and hyperspectral sounders.

5.5.1 CCSDS test datasets

The CCSDS working group has created a set of test datasets (59) acquired by 12 spaceborne/airborne hyperspectral sensors, multispectral sensors, and hyperspectral sounders. There are four hyperspectral images acquired using different hyperspectral sensors; three of them are tested:

1. AVIRIS Scene 0 dataset: raw data, 12 bits/sample, acquired using AVIRIS, with a datacube size of 512 lines by 614 samples by 224 bands.
2. CASI t0180f07 dataset: raw data, 12 bits/sample, acquired using CASI, with a datacube size of 2852 lines by 405 samples by 72 bands.
3. Geo sample flatfielded dataset: raw data, 12 bits/sample, acquired using EO-1 Hyperion, with a datacube size of 256 lines by 1024 samples by 242 bands. The flat-fielded process attempts to normalize the detector variations, causing some data points to have negative values. However, the dynamic range is still within 12 bits.

There are two sets of CCSDS test hyperspectral-sounder datasets: atmospheric infrared sounder (AIRS) and infrared atmospheric sounding interferometer (IASI). The AIRS (granule 9) dataset, which is in raw data format stored as a 2-byte integer, is tested. The actual sample values have bit depths that range from 12 to 14, depending on the spectral band. The unstable bands have been deleted, reducing the dataset from 2107 to 1501 spectral bands. The datacube size is 135 lines by 90 samples by 1501 bands.

There are six sets of CCSDS test multispectral images; three of them are tested:

1. MODIS datasets: raw data with 12 bits/sample, acquired using Moderate-Resolution Imaging Spectroradiometer (MODIS) in the year 2001. There are 36 bands in this Earth-view (EV) data, grouped into four datasets based on different GSDs:
   - Group 1: 250 m, bands 1 and 2 (2 bands);
   - Group 2: 500 m, bands 3–7 (5 bands);
   - Group 3: 1 km-day, bands 8–19 (1 km-day file contains both high-gain and low-gain readings for bands 13 and 14, for a total of $12 + 2 = 14$ bands); and
   - Group 4: 1 km-night, bands 20–36 (17 bands).

   Groups 4 and 5 are selected as CCSDS test multispectral images. The datacube size of the MODIS-Day and MODIS-Night is 2030 lines by

1354 samples by 14 bands, and 2030 lines by 1354 samples by 17 bands, respectively.

2. Pleiades datasets (there are two datasets: Montpellier and Perpignan sites): raw data with 12 bits/sample, simulation of the Pleiades sensor based on real airborne and spaceborne data. The GSD of datasets is 2.8 m. There are four bands of datasets:
   - B0: 430–550 nm (blue),
   - B1: 490–610 nm (green),
   - B2: 600–720 nm (red), and
   - B3: 750–950 nm (NIR).

   The datacube size of the Montpellier and Perpignan is 224 lines by 2456 samples by 4 bands, and 224 lines by 3928 samples by 4 bands, respectively.
3. VEGETATION dataset: raw data with 10 bits/sample, acquired using the VEGETATION payload aboard SPOT-4. There are four bands of the dataset:
   - B0: 430–470 nm (blue),
   - B2: 610–680 nm (red),
   - B3: 780–890 nm (NIR), and
   - MIR: 1580–1750 nm.

   The datacube size is 10080 lines by 1728 samples by 4 bands.

5.5.2 Test results of hyperspectral datasets

This section describes compression results of the CCSDS test hyperspectral datasets using SAMVQ. For the sake of comparison, the results produced using six other lossy data-compression techniques selected by the CCSDS working group are also reported:

1. JPEG2000 Compressor with bitrate allocation (JPEG2000 BA),[9]
2. JPEG2000 Compressor with spectral decorrelation (JPEG2000 SD),[9]
3. CCSDS Image Data Compressor[10] (CCSDS-IDC) in frame mode with 9/7 wavelet floating in block 2,
4. ICER-3D,[11]
5. Fast-lossless / near-lossless (FL-NLS),[12] and
6. Fast-lossless / near-lossless, updated in 2009 (FLNLS2009).

JPEG2000 was applied to compress the hyperspectral images with bitrate allocation mode (2D) and spectral decorrelation mode (3D).[9]

CCSDS-IDC is a 2D compression algorithm[10] that consists of two processing steps: a discrete wavelet transform (DWT) that performs decorrelation of an image to be compressed, and a bit-plane encoder that encodes the decorrelated data. There are two operating modes: frame and strip. The frame mode supports input formats produced, for example, by

CCD arrays, and the strip mode supports input formats produced by push-broom sensors. CCSDS-IDC can compress images in both lossy and lossless formats if the integer DWT is used. The compression results reported in this section are obtained by using frame mode (Daubechies 9/7 wavelet in a float DWT), as this parameter setting produces the best rate-distortion performance.

ICER is a wavelet-based 2D image data compressor designed to meet the specialized needs of deep-space applications.[11] It features progressive compression and can provide both lossless and lossy compression. ICER compresses a simple binary coefficient of the transformed image by successively encoding groups of bits, starting with groups containing highly significant bits and working toward groups containing less-significant bits. The entropy coder used is interleaved entropy coding. It also incorporates an error-containment scheme to limit the effects of data loss. The compression results reported in this section were obtained by adding a spectral decorrelation step before applying ICER, which is referred to as ICER-3D.

"Fast lossless," a term used by Klimesh,[12] is a 3D, predictive lossless-compression algorithm designed to be suitable for implementation in hardware, such as a field programmable gate array (FPGA), for onboard use. The predictive step of the algorithm makes use of the sign algorithm, which is a relative of the least mean square algorithm from the field of low-complexity adaptive filtering. The compressed datastream consists of prediction residuals encoded using a method similar to that of the JPEG-LS lossless image-compression standard. The essence of this algorithm is the adaptive linear predictor using the sign algorithm for filter adaptation, with local mean estimation and subtraction. This predictor uses six neighboring samples with three samples from the same band as the sample to be predicted, and one sample each from the three preceding bands. This algorithm can provide near-lossless or lossy compression when the prediction residuals are quantized and encoded. The compression results reported in this section were obtained by using the near-lossless mode of the algorithm (referred to as FL-NLS). Klimesh has slightly tuned and updated the algorithm in 2009 after the publication of his early work.[12] This updated version was also used in the test (referred to as FLNLS2009).

The compression results are reported in terms of rate-distortion performance. The CCSDS working group recommended six bitrates for the compression tests: 0.1, 0.25, 0.5, 1.0, 2.0, and 4.0. The distortion is measured using PSNR and RMSE. The *Peak signal* used in the equation is the real maximum value in the datacubes instead of the maximum value defined by the word-length of the datacubes.

Figures 5.12–5.15 show the rate-distortion curves of the three testing hyperspectral datasets and one sounder dataset produced using SAMVQ and

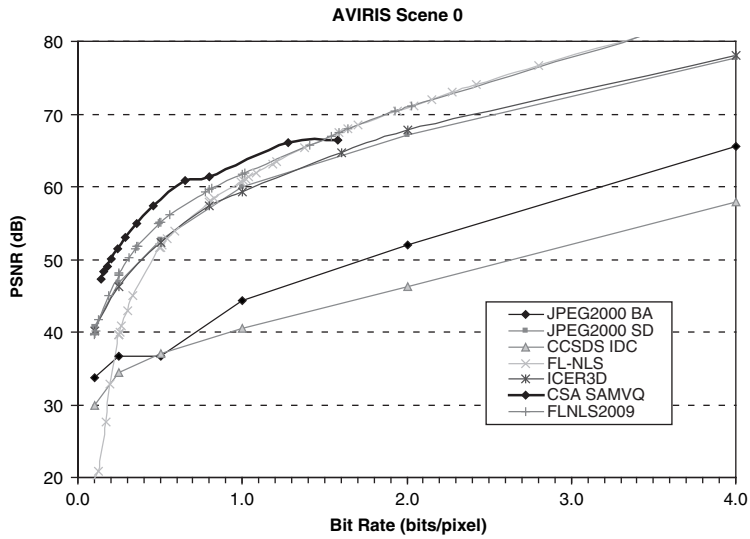


Figure 5.12 Rate-distortion curves of the AVIRIS Scene 0 datacube compressed using SAMVQ and the six compression techniques selected by the CCSDS.

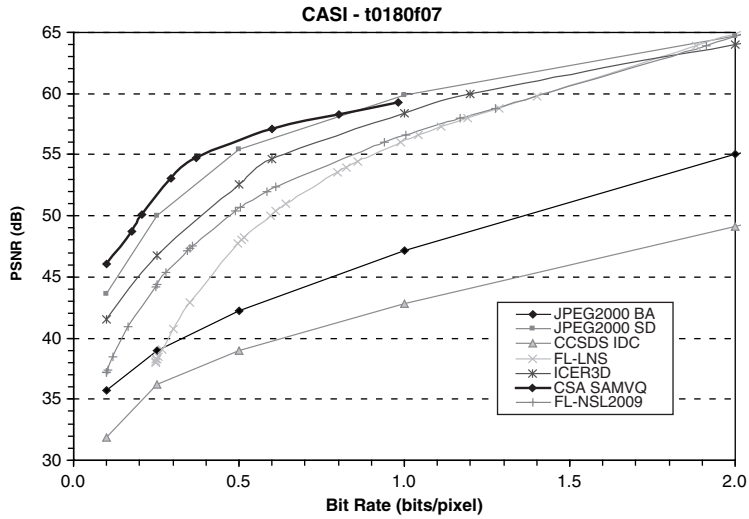


Figure 5.13 Rate-distortion curves of the CASI datacube compressed using SAMVQ and the six compression techniques selected by the CCSDS.

the six CCSDS-selected compression techniques. The figures show that SAMVQ produces the best rate-distortion performance among all of the test images at the lower bitrates. CCSDS-IDC produces the worst rate-distortion performance, and JPEG2000 BA performs the second-worst for the three test

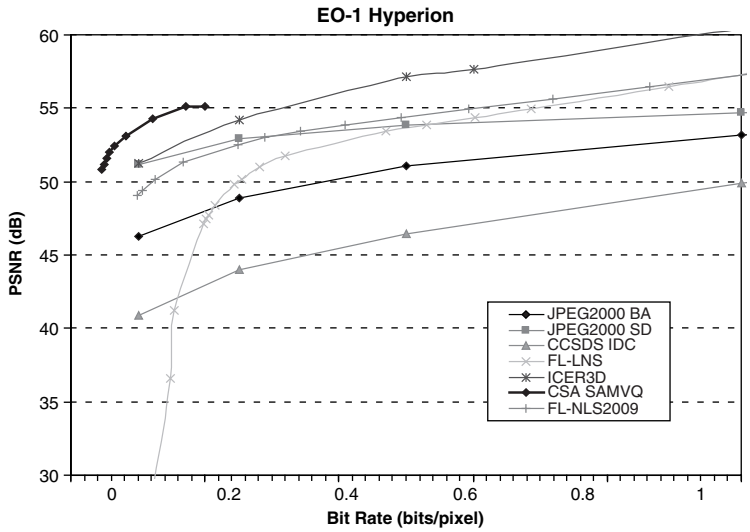


Figure 5.14 Rate-distortion curves of the EO-1 Hyperion datacube compressed using SAMVQ and the six compression techniques selected by the CCSDS.

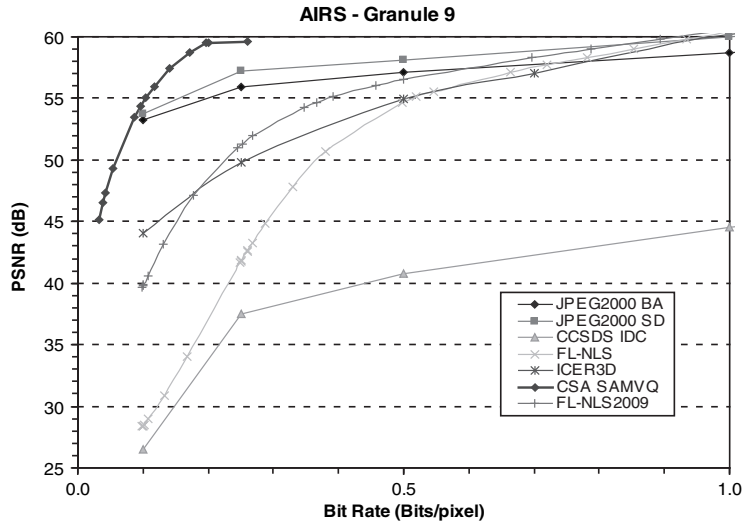


Figure 5.15 Rate-distortion curves of the AIRS Granule 9 datacube compressed using SAMVQ and the six compression techniques selected by the CCSDS.

hyperspectral images. This is expected, as both of them are 2D compression algorithms. The other four compression techniques are 3D algorithms that perform one after another for different test images, with FL-NLS showing the worst performance at lower bitrates.

From the rate-distortion curves of the AIRS sounder image in Fig. 5.15, it can be seen that JPEG2000 SD performs the second-best and JPEG2000 BA performs the third-best at bitrates $\leq$0.5 bits/pixel. JPEG2000 BA is expected to perform rate distortion worse than the 3D compression algorithms because it is a 2D compression algorithm. These compression performances remain to be studied and explained.

It is worth mentioning that SAMVQ does not produce compression results at high bitrates (e.g., $>$1 bit/pixel) because it is a successive, multistage, approximation VQ algorithm that compresses a datacube by checking the convergence of compression error stage by stage. It ceases to compress on the compression error at a stage reaching the asymptote of the curve of fidelity vs. compression ratio in order to achieve an efficient compression. For the AVIRIS datacube, the upper-bound bitrate that SAMVQ can produce is 1.58 bits/pixel. For the CASI datacube, the upper-bound bitrate is 1.0 bits/pixel. For the Hyperion datacube, the upper-bound bitrate is 0.2 bits/pixel. Generally speaking, the noisier the datacube to be compressed is, the smaller the upper-bound bitrate.

5.5.3 Compression of multispectral datasets using SAMVQ

Figure 5.16 shows the compression results of a MODIS-Day image using SAMVQ and the six CCSDS-selected compression techniques. SAMVQ still outperforms the other compression techniques, and the CCSDS-IDC remains the worst. JPEG2000 SD and BA produce the second- and fourth-best rate-distortion

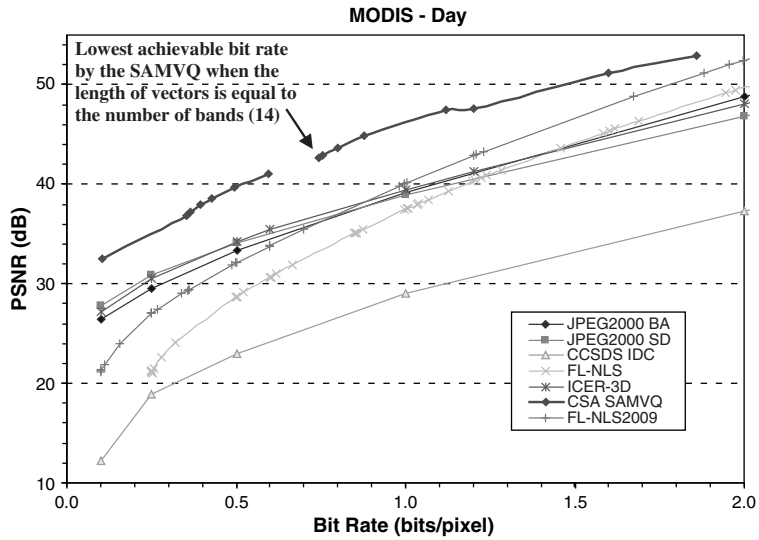


Figure 5.16 Rate-distortion curves of the MODIS-Day image compressed using SAMVQ and the six compression techniques selected by the CCSDS.

performance when the bitrate is $\leq$0.5 bits/pixel. These compression results make more sense than those of the AIRS sounder dataset in Section 5.5.2, as the band images of a multispectral sensor are less correlated than those of hyperspectral and sounder sensors.

When SAMVQ is applied to compress a hyperspectral or multispectral datacube, it assigns all of the elements of the spectrum of a spatial sample pixel as a vector. Due to the small number of bands in a multispectral dataset, the length of the spectral vectors is short; thus the lowest bitrate (i.e., highest compression ratio) that can be achieved by SAMVQ is normally limited because the compression ratio attained by SAMVQ is proportional to the number of spectral bands N_b, as defined by Eq. (4.25). The longer the vector is, the higher the compression ratio. For example, the MODIS-Day image has $N_b = 14$ bands, and the rate-distortion curve produced by SAMVQ shown in Fig. 5.16 indicates that the lowest achievable bitrate is 0.74 bits/pixel (corresponding to the highest compression ratio of 16.3:1). There are two segments of the rate-distortion curve produced by SAMVQ in Fig. 5.16: the right segment is the rate-distortion curve when the length of vectors is equal to the number of spectral bands $N_b = 14$, and the left segment is the rate-distortion curve produced after increasing the length of vectors to 28 by reorganizing the vectors (shown in Fig. 5.17).

In order to attain lower bitrates (i.e., higher compression ratios) for the multispectral datasets, this section reorganizes the multispectral data to form

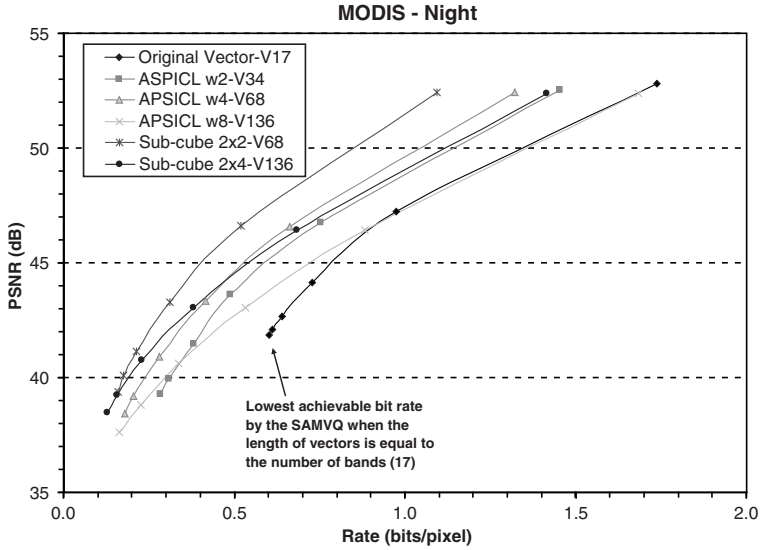


Figure 5.17 Rate-distortion curves of the MODIS-Night image compressed using SAMVQ with vectors longer than the number of bands.

longer vectors (i.e., higher vector dimension). Three schemes are proposed to form longer vectors:

1. In band-interleaved-by-pixel (BIP) format of a multispectral datacube file, every w (e.g., 2 or 4) adjacent pixel spectra in a cross-track line are organized into a longer vector (e.g., the second spectral vector is appended to the end of the first spectral vector, the third spectral vector is appended to the end of the second spectral vector, and so on). The length of such a reorganized vector is equal to $w \times N_b$. The vectors formed in this way still retain spatial correlation of the pixels for compression. This vector-reorganized scheme is referred to as adjacent pixel spectra in a cross-track line (APSICL).
2. In BIP format of a multispectral datacube file, pixel spectra within each subcube of spatial size $h \times w$ (e.g., 2×2 or 2×4) pixels are organized into a longer vector. The spectral vectors in a subcube are linked one after another in the order of raster scanning to form a longer vector. The length of such a reorganized vector is equal to $h \times w \times N_b$. When the height of the subcubes is reduced to $h = 1$, this scheme returns to scheme 1. In band-sequential (BSQ) format of a multispectral datacube file, pixel values within a block of size $h \times w$ of a band image are organized into a subvector. Then this subvector is appended by the subvector formed in the same way as the next band image, and so on. The longer vectors formed in this way are the same as those formed from the datacube file in BIP format except that the order of the vector elements are different, which will not affect the compression results of a VQ-based compression algorithm. This vector-reorganized scheme is referred to as Subcube $h \times w$.
3. In BSQ format of a multispectral datacube file, a partial line of length w pixels of a band image is assigned as a vector. The length of such a formed vector is equal to w. It is preferable to select a value of w so that it can divide a line of the datacube in integer. In this way, such a formed vector will not cover two lines. This vector-reorganized scheme is referred to as Segment Line w.

Figure 5.17 shows SAMVQ compression results of the MODIS-Night datacube produced after using longer vectors with vector-reorganizing schemes APSICL and Subcube. The MODIS-Night datacube has $N_b = 17$ bands. The original vector length is equal to $N_b = 17$. For this datacube, after longer vectors are used, lower bitrates are achieved, and the rate-distortion performance is improved (up to 4 dB). The vector-reorganizing scheme "Subcube 2×2" (i.e., vector length $= 2 \times 2 \times 17 = 68$) produces the best rate-distortion performance. The vector-reorganizing scheme "APSICL w4" that forms the same vector length as "Subcube 2×2" performs worse than that of the "Subcube 2×2."

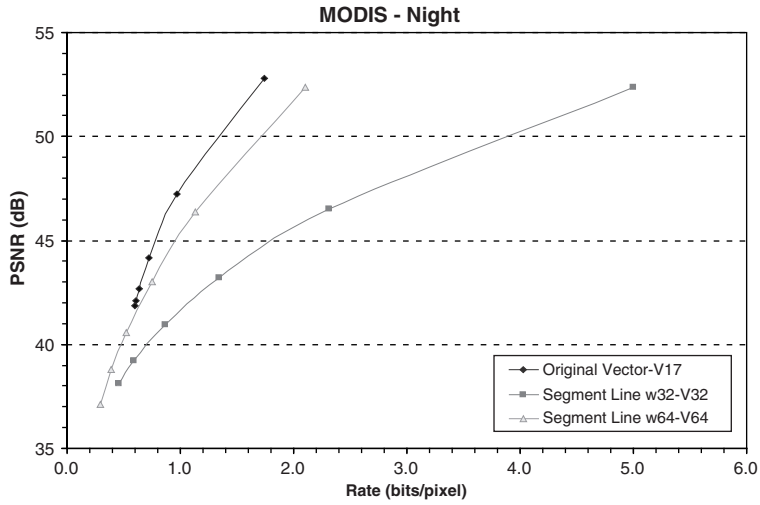


Figure 5.18 Rate-distortion curves of the MODIS-Night image using the Segment Line scheme to form longer vectors.

The lowest bitrate that can be achieved by SAMVQ is 0.6 bits/pixel when the vector length is equal to $N_b = 17$. The compression results of a MODIS-Night datacube using the Segment Line scheme to form longer vectors are shown in Fig. 5.18. It can be seen from this figure that the rate-distortion performance produced by using the Segment Line scheme to form longer vectors is worse than that of the original vector length, which is probably due to the fact that the vectors formed in this way do not take advantage of the spectral correlation of the datacube.

Figure 5.19 shows SAMVQ rate-distortion curves of a MODIS-Day datacube produced after using longer vectors with vector-reorganizing schemes APSICL and subcube. MODIS-Day datacube has $N_b = 14$ bands. The original vector length is equal to $N_b = 14$. The lowest bitrate that can be achieved by SAMVQ is 0.74 bits/pixel when the original vector length is used. For this datacube, with the original vector length, SAMVQ produces the best rate-distortion performance. When longer vectors are used, only the lower bitrates are achieved in exchange for poorer rate-distortion performance. The shorter the reorganized vector length is, the better the rate-distortion performance.

Figures 5.20–5.22 report the compression results of SAMVQ with the vector length equal to the number of bands, and the longer vectors for the CCSDS test multispectral datacubes whose number of bands is extremely small ($N_b = 4$). When the vector length is equal to $N_b = 4$, the highest compression ratio that can be achieved by SAMVQ is too small to be attractive. A VQ-based compression algorithm is usually not recommended

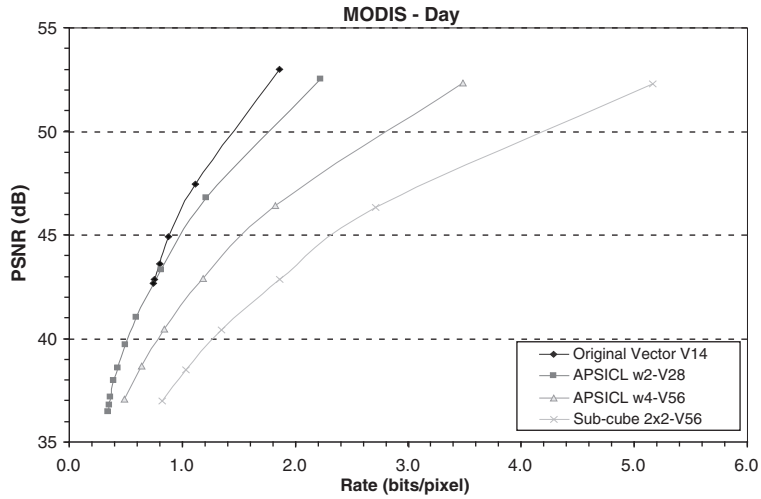


Figure 5.19 Rate-distortion curves of the MODIS-Day image using the vectors longer than the number of bands.

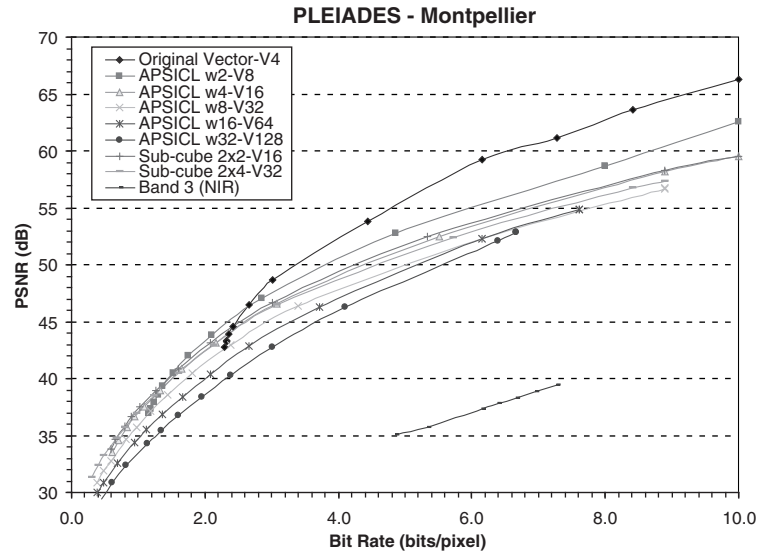


Figure 5.20 Rate-distortion curves of the PLEIADES-Montpellier image using the vectors longer than the number of bands.

for use if the number of dataset bands is too small. This section investigates the lowest bitrate that can be achieved by SAMVQ when the length of the vectors is increased by reorganizing the image data at the cost of compression fidelity.

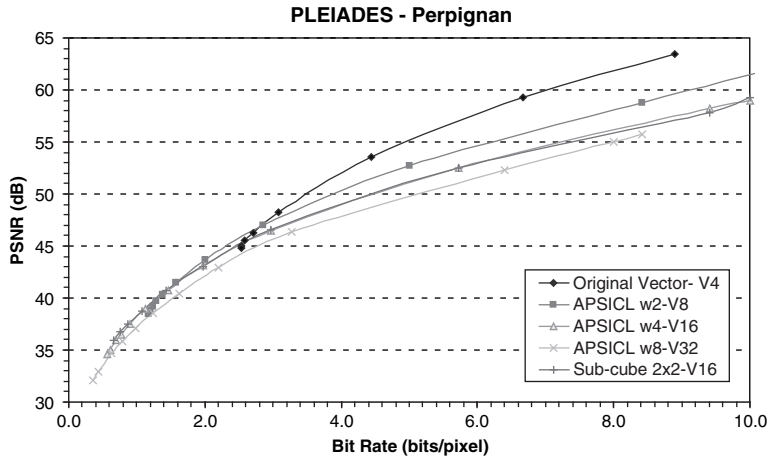


Figure 5.21 Rate-distortion curves of the PLEIADES-Perpignan image using the vectors longer than the number of bands.

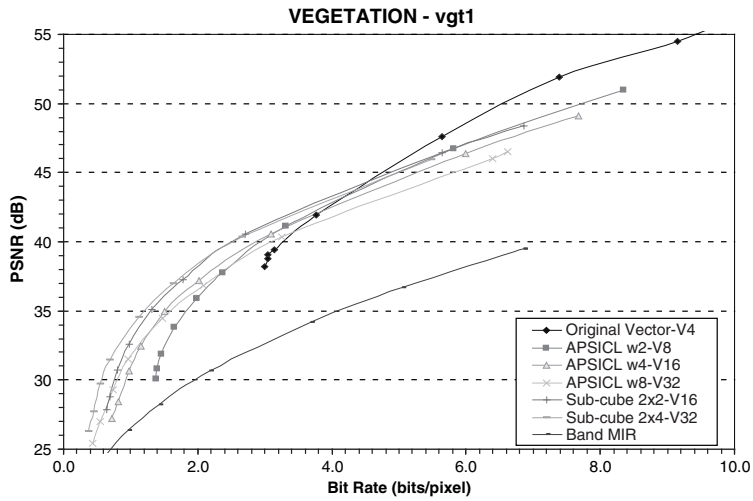


Figure 5.22 Rate-distortion curves of the VEGETATION-vgt1 image using vectors longer than the number of bands.

The rate-distortion curves of the PLEIADES-Montpellier datacube, with the vector length equal to the number of bands and the longer vectors, are shown in Fig. 5.20. When vector length is equal to $N_b = 4$, SAMVQ produces the best rate-distortion performance from the high bitrates to a bitrate of 2.6 bits/pixel. Beyond this point, when the longer vectors are used, SAMVQ produces better rate-distortion performance than that achieved when the original vector length is used. With lower bitrates, SAMVQ produces the best

rate-distortion performance with a vector length of 8 using "APSICL w2-V8," a vector length of 16 using "Subcube 2 × 2-V16", and a vector length of 32 using "Subcube 2 × 4-V32." Extremely extended, longer vectors (e.g., V64 or V128) do not show good rate-distortion performance for this datacube. Figure 5.20 also shows the rate-distortion curve of a single band image (Band 3–NIR) compressed by SAMVQ. The rate-distortion performance of this compression case is significantly poor compared to compression of the multispectral bands with the original vector length and the longer vectors.

Figure 5.21 depicts the rate-distortion curves of the PLEIADES-Perpignan datacube with the vector length equal to the number of bands and the longer vectors. The compression results of this datacube demonstrate the similar observation as the PLEIADES-Montpellier datacube. The rate-distortion curves of the extremely longer vectors (e.g., V64 or V128) are not shown because they are similar to those in Fig. 5.20.

The rate-distortion curves of the VEGETATION-vgt1 datacube, with the vector length equal to the number of bands and the longer vectors, are shown in Fig. 5.22. The compression results of this datacube appear similar to the two PLEIADES datacubes.

References

1. Qian, S.-E., M. Bergeron, I. Cunningham, L. Gagnon, and A. Hollinger, "Near Lossless Data Compression On-board a Hyperspectral Satellite," *IEEE Trans. Aerospace and Electron.Systems* **42**(3), 851–866 (2006).
2. Chen, K. and T. V. Ramabadran, "Near-lossless compression of medical images through entropy-coded DPCM," *IEEE Trans. Med. Images* **13**, 538–548 (1994).
3. Wu, X. and P. Bao, "L_∞ constrained high-fidelity image compression via adaptive context modeling," *IEEE Trans. Image Process.* **9**, 536–542 (2000).
4. Aiazzi, B., L. Alparone, and S. Baronti, "Near-lossless compression of 3-D optical data," *IEEE Trans. Geosci. Remote Sens.* **39**(11), 2547–2557 (2001).
5. Magli, E., G. Olmo, and E. Quacchio, "Optimized onboard lossless and near-lossless compression of hyperspectral data using CALIC," *IEEE Geosci. Remote Sens. Lett.* **1**, 21–25 (2004).
6. Qian, S.-E. and A. Hollinger, "Method and System for Compressing a Continuous Data Flow in Real-Time Using Cluster Successive Approximation Multistage Vector Quantization," U.S. Patent No. 7,551,785, issued on June 23, 2009.
7. Qian, S.-E. and A. Hollinger, "Method and System for Compressing a Continuous Data Flow in Real-Time Using Recursive Hierarchical

Self-Organizing Cluster Vector Quantization," U.S. Patent No. 6,798,360 B1, issued on September 28, 2004.

8. Natarajan, B. K., "Filtering Random Noise from Deterministic Signals via Data Compression," *IEEE Trans. Signal Process*. **43**, 2595–2605 (1995).
9. International Standard Organization, *Information Technology - JPEG 2000 Image Coding System: Core Coding System*, 2nd Ed., ISO/IEC 15444-1, Geneva, Switzerland (2004).
10. "Recommendation for Space Data System Standards," *Lossless Image Compression*, CCSDS 122.0-B-1, Blue Book, Issue 1, available at http://public.ccsds.org/publications/archive/122x0b1c3.pdf, CCSDS, Washington, D.C. (Nov. 2005).
11. Kiely, A. and M. Klimesh, "The ICER progressive wavelet image compressor," *The Interplanetary Network Progress Report* **42**(155), 1–46, Jet Propulsion Laboratory, Pasadena, CA (Nov. 2003).
12. Klimesh, M., "Low-Complexity Adaptive Lossless Compression of Hyperspectral Imagery," *Proc. SPIE* **6300**, 63000N (2006).

Chapter 6
Optimizing the Performance of Onboard Data Compression

6.1 Introduction

This chapter addresses the optimization and implementation aspects of the onboard near-lossless data compression using SAMVQ and HSOCVQ in the image-acquisition and data-handling chain of a satellite system. The raw data acquired by a satellite sensor is often not perfect; there are anomalies in the raw data caused by detector and instrument defects. It is necessary to assess how these anomalies will affect the compression performance onboard. The outcome of the assessment will help determine whether the anomalies should be removed onboard before compression.[1]

Another implementation aspect related to optimization of the onboard near-data compression is preprocessing and radiometric normalization to convert raw sensor data to radiance. In other words, SAMVQ or HSOCVQ should be applied to either the raw sensor data or the radiance data. The evaluation of the effect of onboard preprocessing and radiometric conversion needs to be performed in order to examine whether they should be carried out onboard before compression. Radiance data obtained after radiometric calibration often contains random noise and some artifacts induced during this process. How do the random noise and artifacts in radiance affect the compression performance if the onboard compression is applied to the radiance data?

For hyperspectral imagery, there are two kinds of distortions: spatial distortion (often referred to as "keystone") and spectral distortion (often referred to as "smile"). How do these two distortions compromise the SAMVQ or HSOCVQ performance onboard? Should these distortions be corrected onboard before compression?

Finally, the resilience of the two compression techniques to bit errors caused by single-event upsets (SEUs) is evaluated. This feature helps add proper error-correction measures to enhance the robustness of the compressed files and to decide the appropriate system requirement that prevents error propagation and data loss.

6.2 The Effect of Raw Data Anomalies on Compression Performance

This section evaluates the impact of anomalies in raw hyperspectral data on the performance of data compression using SAMVQ and HSOCVQ algorithms for the purpose of determining whether the anomalies need to be removed onboard before compression.

6.2.1 Anomalies in the raw hyperspectral data

Anomalies in raw data can be due to a variety of causes, including detector defects, saturating targets, etc. Detector defects can appear as zero-pixel values (dead detector elements), fixed values (frozen detector elements), spikes (isolated over-responsive detector elements), saturation, etc. Some detector anomalies occur regularly: their locations in the detector array (i.e., spectral band numbers and cross-track columns) do not change. Of these, dead detector elements (zeroes) in the same bands of a spectrum of the raw data are expected to have no impact on data compression because they do not contribute to the codevector training or to the calculation of the compression fidelity. Fixed values are expected to have a minor effect on data compression because their values do not change in the bands of spectra where frozen detector elements occur. Thus, the values of codevectors in these bands remain the same during the codevector training. This section focuses on the effect of spikes and saturation on raw hyperspectral data.

Two hyperspectral datacubes are used to test the anomaly impact on data compression performance. Datacube 1 was acquired using an airborne Short-Wave Infrared Full Spectrum Imager II (SFSI-II)[2] on June 7, 2002 at an altitude of 2900 m with a swath width of 1.8 km, for a ground pixel size of 3.5 m $\times$ 3.5 m, and 240 bands between 1200–2450 nm with a band interval of 5 nm. It is a raw datacube and recorded in 12-bit DN. The size of the datacube is 140 lines by 496 pixels per line by 240 bands. This SFSI-II datacube was originally acquired for use in target detection of hyperspectral imagery. Synthetic targets with sizes ranging from 12 m $\times$ 12 m to 0.2 m $\times$ 0.2 m of five different materials (awnings, polythene, plastic tarp, cotton, and vinyl mat) were deployed in the scene.

Datacube 2 was acquired using an airborne CASI over the cornfields at the L'Acadie experimental farm (Agriculture and Agri-Food Canada) during the summer of 2000. CASI has 72 spectral bands with a spectral resolution of 7.6 nm in the range of 408–947 nm. Figure 6.1 shows the color-composite image of the raw datacube with band 53 (775 nm) printed as red, band 33 (631 nm) as green, and band 20 (538 nm) as blue. The whole image covers an area of 2450 m $\times$ 810 m, with the image size of 1225 lines by 405 pixels per line, and the GSD of 2 m $\times$ 2 m.

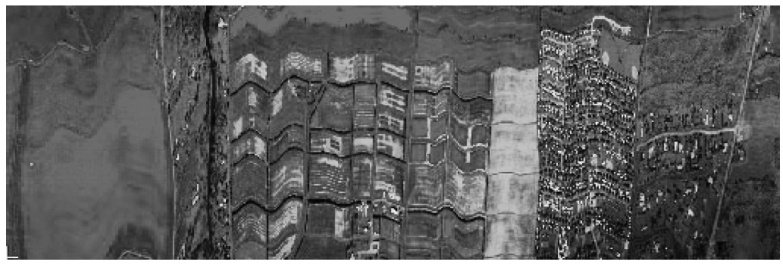


Figure 6.1 RGB image of the raw CASI test datacube (displayed in black and white here).

6.2.2 Effect of spikes on compression performance

The test SFSI-II datacube was evaluated to study the effect of anomalies. It was found that there are five detector elements that produce over-responsive values in the SFSI-II 2D detector array; Figure 6.2 shows the locations of these elements in the detector array. The size of the detector array is 240 rows by 496 columns. The column dimension of the detector array corresponds to the ground samples in a cross-track line, whereas the row dimension corresponds to the wavelength expansion (or spectral bands) of each of the ground samples. With this detector configuration, there are 496 ground samples in a cross-track line, and each ground sample has 240 spectral bands.

Figure 6.3 shows examples of spikes in the spectra of the raw SFSI-II datacube due to the over-responsive elements. It can be seen that the spikes occur at spectral bands 11, 34, 93, and 173, and at cross-track column numbers 32, 125, 286, 326, and 327. These spikes occur in the spectra of the

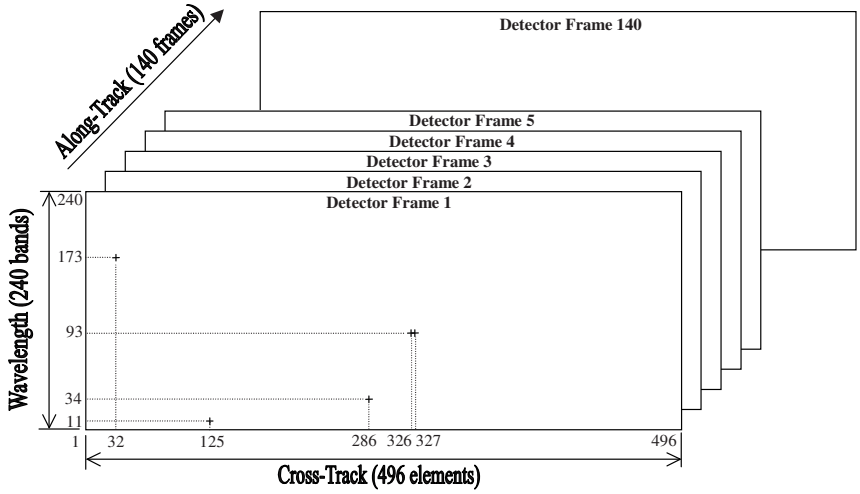


Figure 6.2 Locations of over-responsive elements (spikes) in the detector array (reprinted from Ref. 1).

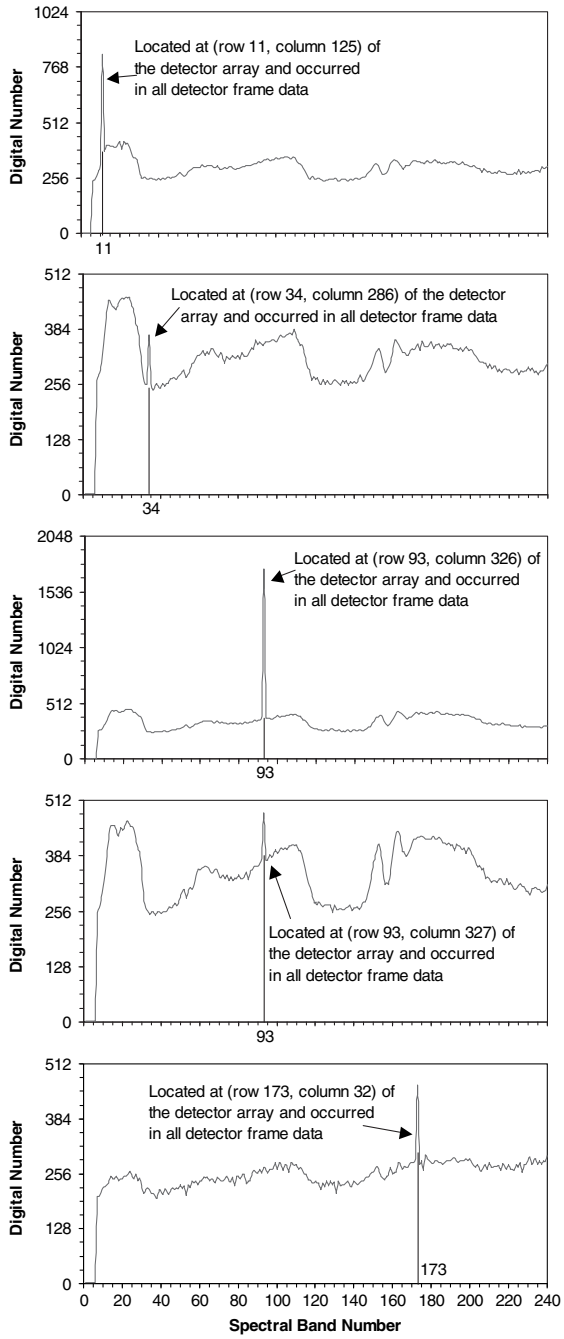


Figure 6.3 Spikes in spectra of raw DN data from SFSI-II (reprinted from Ref. 1).

affected cross-track columns for all of the 140 frames when SFSI-II flies along-track to generate the 140 detector frames (covering 140 along-track lines) of the datacube, as these over-responsive detector elements repeat for all of the detector frames.

In order to examine the impact of the spikes on the compression performance, the spikes were removed by replacing them with interpolation values derived from their spectral neighbors. This spike removal process is also referred to as "despike" here. Both the raw SFSI-II datacube and its despiked datacube were compressed using HSOCVQ and SAMVQ at compression ratios of 10:1 and 20:1. With this range of compression ratios, both SAMVQ and HSOCVQ produce near-lossless compression as defined in Section 5.1. This range of compression ratios meets the needs of the data-reduction requirement for transmission of hyperspectral data to the ground.[3]

Three statistical measures—RMSE, SNR, and percentage error (%E)—were used to evaluate the compression performance. The %E is defined as follows:

$$\%\mathrm{E} = \left(\frac{1}{N_r \times N_c \times N_b}\right) \sum_{i\ 1}^{N_r} \sum_{j\ 1}^{N_c} \sum_{\lambda\ 0}^{N_b} \left| \frac{x_{i,j}(\lambda) \quad \hat{x}_{i,j}(\lambda)}{x_{i,j}(\lambda)} \right| \times 100, \qquad (6.1)$$

where $x_{i,j}(\lambda)$ and $\hat{x}_{i,j}(\lambda)$ are digital numbers of the raw data and the reconstructed data, respectively, at spatial location (i, j) of band λ. N_r, N_c, and N_b are the total number of lines, the total number of pixels per line, and the total number of bands of the datacube, respectively.

The upper part of Table 6.1 lists the compression results. The maximum value of the datacube is reduced to 518 DN from 1754 after

Table 6.1 Compression statistics and evaluation results using target detection as a remote sensing application for the raw SFSI-II datacube with and without removing spikes before compression.

Compression Statistics	**HSOCVQ**				**SAMVQ**			
	10:1	**10:1**	**20:1**	**20:1**	**11.7:1**	**11.7:1**	**20:1**	**20:1**
Spikes removed?	**No**	**Yes**	**No**	**Yes**	**No**	**Yes**	**No**	**Yes**
Maximum value (DN)	1754	518	1754	518	1754	518	1754	518
RMSE (DN)	5.21	5.21	5.83	5.79	3.98	4.04	4.32	4.42
SNR (dB)	35.35	35.35	34.37	34.43	37.68	37.56	36.97	36.77
%E	1.20	1.20	1.36	1.35	0.83	0.86	0.97	0.97
Evaluation using R/S application								
Blind file name	CSA1	CSA3	N/A	CSA5	CSA11	CSA10	N/A	CSA8
Total score	10	11	N/A	5	14	15	N/A	10
Acceptability	A	A	N/A	N	A	A	N/A	A

A: Acceptable M: Marginally Acceptable N: Not Acceptable N/A: Not Applicable

despiking. For HSOCVQ, identical compression fidelity (SNR, %E, etc.) was attained for the raw SFSI-II datacube with and without spike removal when the compression ratio (CR) is 10:1. It produced slightly better compression fidelity (0.06-dB higher SNR, and 0.01% lower %E) for the raw SFSI-II datacube with despiking when the CR is 20:1; however, this improved compression performance due to despiking is insignificant. These results show that the HSOCVQ algorithm is insensitive to spike anomalies in the raw data.

SAMVQ produced slightly different compression results than HSOCVQ. When the CR is 20:1, SAMVQ produced identical percentage error (0.97%) for the raw SFSI-II datacube with and without removing the spikes. It yielded 0.2-dB lower SNR for the raw SFSI-II datacube with despiking, which is a bit surprising because removing spikes is expected to improve the compression fidelity. This effect is probably caused by the nature of successive approximation of the SAMVQ algorithm and the reduction of dynamic range (maximum value) of the data after despiking. The dynamic range is reduced to $0 \sim 518$ after removing the spikes. The influence (weight) of the intrinsic noise of the datacube may become more explicit for the convergence of the successive approximation because the intrinsic noise is at the same level as the error induced by the compression, as described in Section 5.4. SAMVQ produces slightly poorer fidelity after the maximum value reduced.

The lowest CR achieved for the raw SFSI-II datacube with despiking is 11.7:1, which is caused by one of the features of the SAMVQ algorithm. In SAMVQ, a graph of fidelity versus compression ratio would statistically reach an asymptote. As the number of approximation stages increases beyond a certain stage, the reconstruction fidelity ceases to increase, and the compression ratio decreases; this point is referred to as an inflection point. As described in Section 4.9, SAMVQ compression proceeds automatically stage by stage until the compression fidelity reaches a given threshold or the inflection point is detected, whichever comes first. In this way, efficient compression is achieved. The inflection point was detected at a CR of 11.7:1, when the despiked raw datacube was compressed (the CR for the raw SFSI-II datacube without despiking was set to this ratio for a consistent comparison). It can be seen from Table 6.1 that when the CR is 11.7:1, SAMVQ produced slightly better fidelity (0.12-dB higher SNR, and 0.03% lower %E) for the raw SFSI-II datacube compared to despiking. This also applies to a CR of 20:1. At both ratios, the difference of the compression fidelity attained for the raw SFSI-II datacube with and without despiking is insignificant. These results indicate that the SAMVQ algorithm is also insensitive to spike anomalies in the raw data.

In order to examine how the spikes affect the compression fidelity on individual spectra where the spikes occur, these spectra were compared with their uncompressed versions. Figure 6.4 shows the difference spectra calculated

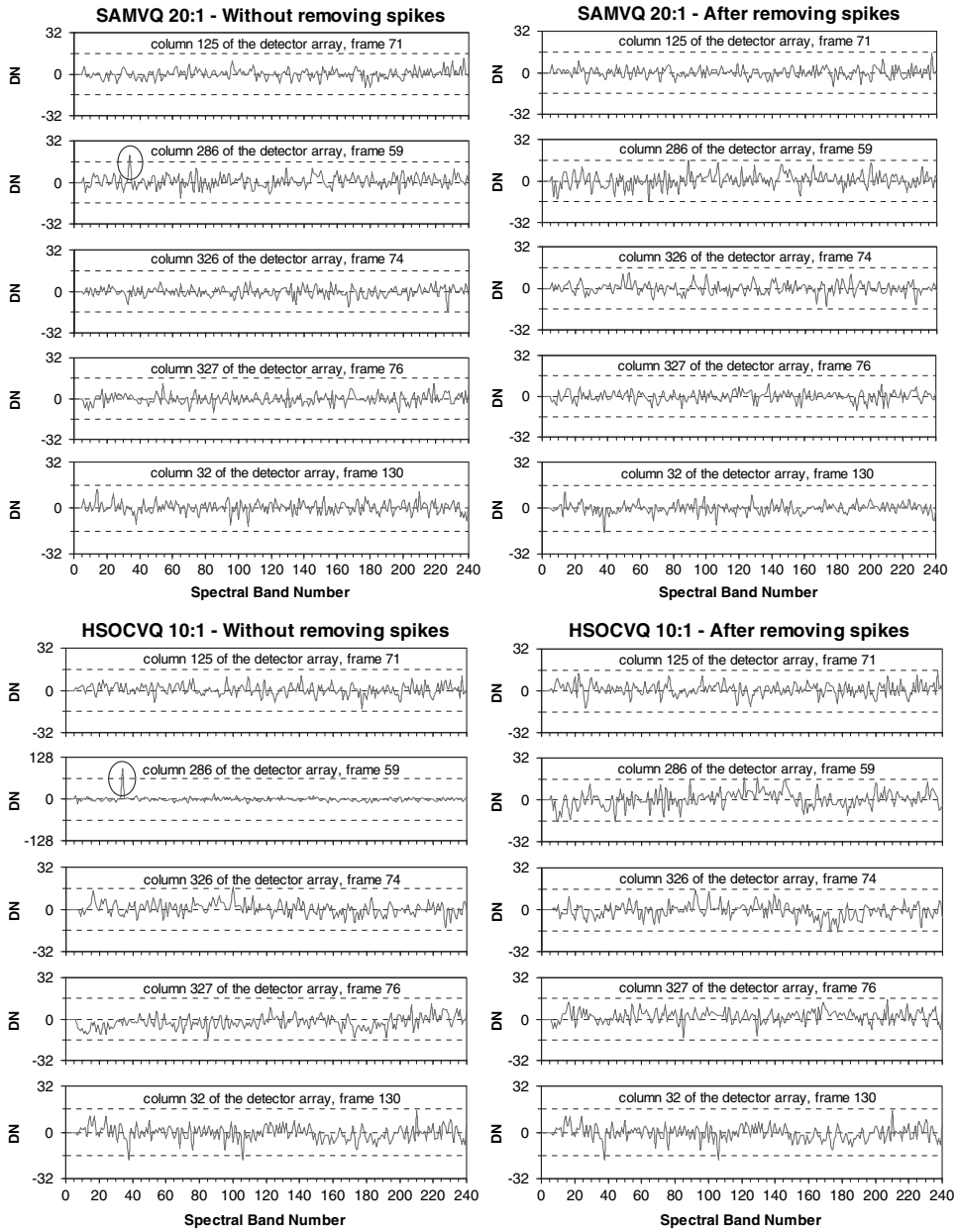


Figure 6.4 Difference spectra between the original and the reconstructed datacubes for the columns of the detector array where the over-responsive elements (spikes) are located (reprinted from Ref. 1).

between the original datacube (uncompressed) and the reconstructed datacube compressed using SAMVQ at a ratio of 20:1 and using HSOCVQ at a ratio of 10:1 on the raw SFSI-II datacube with and without despiking. The frame (i.e., cross-track line) numbers of the spectra were selected arbitrarily. It can be seen that both SAMVQ and HSOCVQ produced excellent reconstruction fidelity for these individual spectra from the compressed datacubes no matter if the raw datacube had or had not been despiked before compression. The difference values are randomly distributed and well bounded within the range of −16 to 16 DN. The dynamic range of the raw datacube is 0 ~ 1754 DN and becomes 0 ~ 518 DN after despiking. For the raw SFSI-II datacube without despiking, the difference spectra do not show large difference values at the bands (11, 34, 93, and 173) where the spikes occur (even for the largest spike at band 93 and column 326 of the detector array), except for the spectrum of pixels at cross-track column 286, where a large difference occurred in band 34 (marked with dotted-line circles).

It is worth the effort to evaluate the impact of spikes on the data compression for remote sensing applications. The raw SFSI-II datacube was originally acquired for investigation of target detection of hyperspectral imagery. Five synthetic targets—awnings, polythene, plastic tarp, cotton, and vinyl mat—with sizes ranging from 12 m × 12 m to 0.2 m × 0.2 m were deployed in the scene of the datacube. This datacube has been used to evaluate impact of data compression on remote sensing application of target detection.[4] This section uses this application to evaluate the impact of spikes on the data compression performance.

The raw SFSI-II datacubes with or without despiking were compressed/decompressed using SAMVQ and HSOCVQ and then sent to the user for evaluation. A double-blind test was used to reduce self-deception and bias in the evaluation. The decompressed datacubes were named using an arbitrary number before being sent to the user. Because the compressed datacubes were in raw DNs, they were first preprocessed to remove periodic noise, dark current, slit curvature (i.e., smile), and keystone. A vicarious calibration was then performed using calibration coefficients derived from a previous SFSI-II survey to convert the raw datacubes into radiance. Nine endmembers were selected that correspond to the five materials of the synthetic targets and the four ground features (forest, gravel road, sand, and grass). Nine regions of interest (ROIs) were selected and used to extract the endmember spectra from each blind-compressed datacube.

The following four qualitative and quantitative criteria were used to assess the impact, and one point was scored for each criterion met:

1. All targets that are present in the fraction images derived from the original datacube are present in the fraction images derived from the compressed datacubes.

2. No targets other than the ones present in the fraction images derived from the original are present in the fraction images derived from the compressed datacubes.
3. The T-test of the distribution of the fraction images derived from the compressed datacubes is not significantly different than that of the fraction images derived from the original datacube at a significance level of 0.01.
4. The percentage standard error (%SE) of a target ROI is less than 5%. It is used to measure the relative average deviation of the fraction images derived from the compressed datacubes and is defined as follows:

$$SE\% = \frac{\frac{1}{n}\sqrt{\sum_{i\ 1}^{n}(f_i \quad \hat{f}_i)^2}}{\overline{f}} \times 100, \tag{6.1}$$

where f_i is the fraction of an endmember for a pixel in an ROI derived from the original datacube, $\hat{f}_i$ is the fraction of the endmember for the same pixel derived from a compressed datacube, $\overline{f}$ is the mean fraction of the endmember for the ROI of the original datacube, and n is the number of pixels in the ROI.

The evaluation was performed on a ROI-by-ROI basis. The full score for a target ROI is 4, and thus the full score for a compressed datacube is 20, as there are five target ROIs. The total score for the evaluation and the acceptance of the six blind-compressed datacubes provided by the user are listed on the lower part of Table 6.1.

For HSOCVQ, the compressed datacubes at a CR of 10:1 without and with despiking before compression received a total score of 10 and 11, respectively, out of the full score of 20. The user accepted both of them. However, the compressed datacube at a CR of 20:1 with despiking before compression got a total score of 5, which the user rejected.

For SAMVQ, the compressed datacubes at a CR of 11.7:1 without and with despiking before compression got a total score of 14 and 15, respectively. The compressed datacube at a CR of 20:1 with despiking got a total score of 10. The user accepted all three of the SAMVQ-compressed datacubes. The evaluation results indicate that removal of spikes before compression has no significant impact on compression performance, as the reconstructed datacubes without and with spikes removed before compression contain the same information for the application.

6.2.3 Effect of saturation on compression performance

It has been found that there are saturations in the raw CASI datacube, examples of which are illustrated in Figure 6.5. It can be seen from the

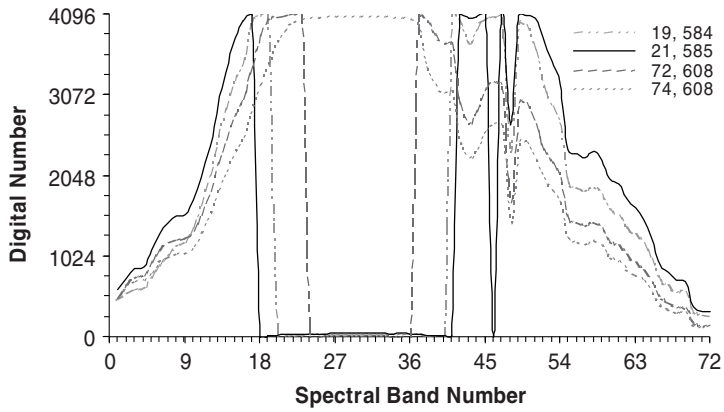


Figure 6.5 Saturations in spectra of four ground sample pixels of the raw CASI datacube (reprinted from Ref. 1).

spectra in the figure that bands that reach the saturation level sharply drop to a value close to zero. This treatment of saturation is unusual; normally, a saturated signal is clipped to the saturation level rather than switched to a value close to zero.

In order to examine the impact of the saturations on the compression performance, the spectra showing signs of saturation were entirely replaced by a typical spectrum without saturation extracted from the datacube. This spectrum was used to replace all the saturated spectra of the datacube for the sake of simplifying the saturation removal.

Table 6.2 lists the compression results of the raw CASI datacube with and without removing the saturated spectra using HSOCVQ and SAMVQ at CRs of 10:1 and 20:1. For HSOCVQ, identical compression results were attained for both the raw datacube and the datacube after removing saturation when the CR is 10:1. Slightly better compression fidelity (0.01-dB higher SNR and 0.01% lower %E) was achieved for the raw CASI datacube after removing the saturations, when the compression ratio is 20:1. These results show the same conclusion as for the impact of spikes on HSOCVQ.

Table 6.2 Compression results of the raw CASI datacube with and without removing saturations

	HSOCVQ				SAMVQ			
	10:1	**10:1**	**20:1**	**20:1**	**10:1**	**10:1**	**20:1**	**20:1**
Saturations removed?	No	Yes	No	Yes	No	Yes	No	Yes
Maximum value (DN)	4095	4095	4095	4095	4095	4095	4095	4095
RMSE (DN)	15.38	15.38	20.92	20.84	3.82	3.78	6.68	6.44
SNR (dB)	35.60	35.60	32.93	32.94	47.69	47.77	42.84	43.14
%E	1.45	1.45	1.99	1.98	0.33	0.32	0.60	0.58

SAMVQ produced slightly better compression fidelity for the raw CASI datacube with removing saturations than without removing saturations. The improvement to the SNR is 0.08 dB and 0.30 dB, respectively, for CRs of 10:1 and 20:1. The reduction of percentage error is 0.01% and 0.02% for CRs of 10:1 and 20:1. The improvement to compression fidelity after removing saturations is expected although insignificant. These compression results are contrary to those for the impact of spikes, which is probably caused by the nature of the anomalies (spikes vs. saturations) and the approaches used to remove them. Removing spikes reduced the maximum value of the datacube significantly (from 1754 to 518) for the raw SFSI-II datacube, whereas removing saturations did not change the maximum value of the raw CASI datacube (4095). The spikes in the spectra were removed by replacing them with the interpolated values from their spectral neighbors. The approach used to remove the saturations in the raw CASI datacube replaced an entire spectrum with a unique typical spectrum extracted from the datacube if the spectrum was found containing saturation in at least a single spectral band. This approach increases the occurrence frequency of the typical spectrum and ultimately increases the compressibility of the datacube after the saturations were removed.

Figure 6.6 shows the difference spectra calculated between the raw CASI and the reconstructed datacubes compressed using SAMVQ at a CR of 20:1 on the raw datacubes with and without removing saturations. It can be seen that the difference spectra of the four ground sample pixels previously shown in Fig. 6.5 are near zero (between 5 and 5 DN) when the saturations in the raw datacube are removed, which implies that the spectra before and after

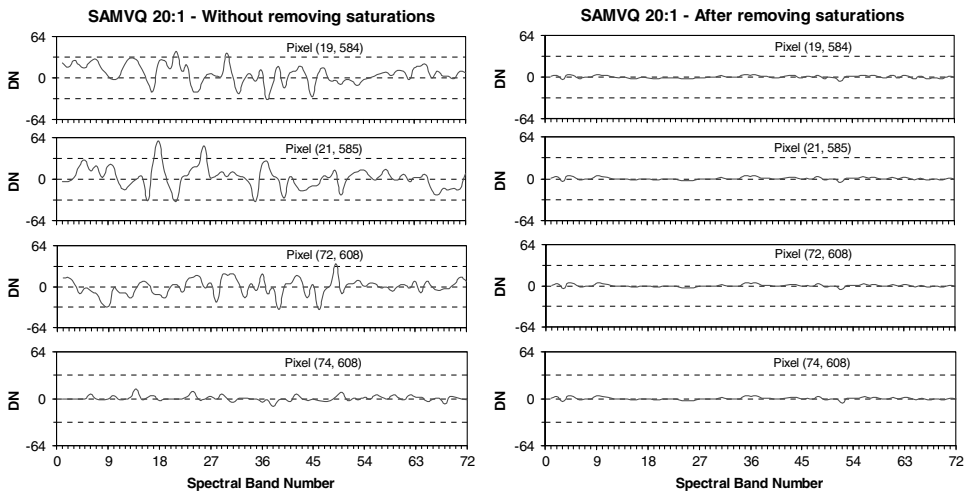


Figure 6.6 Difference spectra between the original and the reconstructed datacubes of the ground sample pixels where the saturations are occurred (reprinted from Ref. 1).

compression are nearly the same. This is expected because the saturated spectra were replaced with a unique typical spectrum and then easily encoded by a unique codevector.

For the raw CASI datacube with saturation effects, the difference spectrum of ground sample pixel (74, 608), whose spectrum values were not switched to a value close to zero (see the dotted-line spectrum in Fig. 6.5), is much smaller than the difference spectra of the other three saturated ground sample pixels, whose spectrum values were switched to a value close to zero when saturation occurred. The values of the difference spectrum of the ground sample pixel (74, 608) are between –10 and 10 DN. Its RMSE is 5.36 DN, which is even smaller than 6.68 DN, the overall RMSE of the reconstructed datacube compressed using SAMVQ at a compression ratio of 20:1 (see Table 6.2). This difference spectrum indicates that SAMVQ can achieve good compression fidelity even if spectra with near- or at-saturation level, provided these spectra are not switched to values close to zero. The difference spectra of ground sample pixels (19, 584), (21, 585), and (72, 608) are relatively large (between 35 and 59 DN) in the spectral band region between 18 and 46, where the spectrum values were switched to close to zero (not to a constant value) when saturation occurred. This is because the close-to-zero values in the spectral bands where the saturations occurred have too little weight to influence the codevector during the training process and are therefore not well accounted for. The approach to saturation whereby the saturated values are switched with a value close to zero is not expected. It is currently unknown why this raw CASI dataset exhibits this treatment of saturation.

6.2.4 Summary of anomaly effects

This section evaluated the impact of spikes and saturations on data compression using SAMVQ and HSOCVQ. Two raw hyperspectral datacubes acquired using airborne hyperspectral sensors SFSI-II and CASI were tested. Statistics-based measures RMSE, SNR, and %E were used to evaluate the compression performance. Difference spectra between the original and reconstructed datacubes at spatial locations where anomalies occur were plotted and verified. The compressed SFSI-II datacubes were also evaluated using a remote sensing application (target detection).

The experimental results show that HSOCVQ is insensitive to both the spikes and saturations when the raw hyperspectral data is compressed. It produced almost the same statistical results, no matter if the spike and saturation anomalies were removed or not before compression. The two reconstructed SFSI-II datacubes compressed using HSOCVQ at a CR of 10:1 with and without removing the spikes before compression were assessed using a target detection application. The reconstructed datacube without removing the spikes before compression received 10 points out of the full score of 20, whereas the reconstructed datacube with the spikes removed before

compression received 11 points. It can be determined that there is no significant impact on the application with respect to removal of spikes because the evaluation scores are close and the user accepted both of them based on a set of predefined criteria.

The experimental results show that the data fidelity of SAMVQ-compressed datacubes is slightly reduced (0.12–0.2 dB SNR) after spikes were removed from the raw datacube before it was compressed. This is probably caused by the reduction of dynamic range (maximum value) of the raw datacube, which was reduced significantly to 0 ~ 518 from 0 ~ 1754 DN after removing the spikes. The influence of the intrinsic noise becomes more explicit for the convergence of the successive approximation of the SAMVQ algorithm. The two reconstructed SFSI-II datacubes compressed using SAMVQ at a CR of 11:7 with and without removing the spikes before compression were evaluated using a target-detection application. The evaluation results showed that the removal of spikes before compression has no significant impact on SAMVQ compression performance because the reconstructed datacubes with and without removing the spikes before compression received a close evaluation score (15 vs. 14). They contain the same information for the application. SAMVQ produced slightly better data fidelity (from 0.08 dB to 0.3 dB SNR) with removing the saturations than without removing the saturations when the raw datacube was compressed at ratios of 10:1 and 20:1. This is because (1) removal of the saturations did not change the dynamic range of the datacube, and (2) an entire spectrum was replaced by a unique typical spectrum if a spectrum were found to contain saturation in a single spectral band. This approach to removing saturations increases the occurrence frequency of the typical spectrum and ultimately increases the compressibility of the datacube.

6.3 The Effect of Preprocessing and Radiometric Conversion on Compression Performance

This section examines the impact of preprocessing and radiometric conversion on data compression using SAMVQ and HSOCVQ aboard a hyperspectral satellite to decide whether they should be applied onboard before compression. In other words, the compression should be applied to either raw data or its radiance version.

6.3.1 Artifacts introduced in preprocessing and radiometric conversion

Preprocessing includes the removal of dark current, offset, and noise, and the correction of nonuniformity. Radiometric conversion refers to the conversion of the raw data to at-sensor radiance. After preprocessing and radiometric conversion, the radiance data is usually encoded with 16-bit signed data. Preprocessing and radiometric conversion processes can introduce spikes in

the spectra of the radiance data, for example, when the raw data values are divided by extremely small coefficients for correcting nonuniformity of the detectors. A spike introduced in these processes is different from that in the raw data caused by an isolated over-responsive detector element. The former occurs at arbitrary spectral bands of an arbitrary radiance spectrum, while the latter occurs at a persistent band of the spectrum of raw data at a fixed cross-track pixel for all of the along-track lines (along-track lines corresponding to a focal plane frame), as shown in Fig. 6.3. Regularly and frequently occurring spikes in a raw datacube can be well trained by SAMVQ and HSOCVQ algorithms at the cost of a few additional codevectors. A spike occurring at an arbitrary band of an arbitrary radiance spectrum in a datacube cannot be easily handled during the compression's training phase and will ultimately reduce the CR and compression fidelity of the datacube.

These processes can also introduce negative values in radiance data when large coefficients are subtracted from small values of the raw data to remove dark current and offsets. A negative value ' 1' in radiance data, for example, will be interpreted as 2^{16} 1 = 65,535 when a compression algorithm is set to operate on unsigned 16-bit data. This ' 1' will be treated as a spike by the algorithm if it is an isolated value. The impact of this kind of false spikes on data compression is even higher than the spikes due to detector defects, as these false spikes occur irregularly, and their amplitudes are extremely large (e.g., 65,535). Experimental results indicate that the impact of spikes and negatives in radiance data introduced in preprocessing and radiometric conversion on compression performance is significant, and they should be removed before compression.

Because both the preprocessing and radiometric conversion alter the raw data, their impact on data compression cannot be evaluated by simply comparing the statistics of the compression results (e.g., RMSE, SNR, and %E) obtained for the raw data with those for the radiance data; remote sensing applications were used for that. This section shows the results from the evaluation of compressed datacubes using remote sensing application products with preprocessing and radiometric conversion performed either after or before compression. In other words, the compression was applied to either a raw datacube or to the corresponding radiance datacube. Three datacubes from CASI and one datacube from SFSI-II were tested. Spikes and negative values in radiance datacubes were removed before compression. The spikes and saturations in raw data were also removed for a consistent evaluation, although they do not show a significant impact on the compression performance (as described in Section 6.2).

6.3.2 Evaluation using crop leaf area index in agriculture applications

Crop leaf area index (LAI) derived from hyperspectral datacubes was used as a remote sensing application to evaluate the impact of preprocessing and

radiometric conversion on data compression. Both the LAI derived from the compressed datacube with the compression applied to raw data and the LAI derived from the compressed datacube with the compression applied to the same data that has undergone preprocessing and radiometric conversion were compared with the ground truth. A double-blind test was adopted to reduce the bias in the evaluation.

The hyperspectral datacubes tested were acquired using CASI over the crop fields at the former Greenbelt farm of Agriculture and Agri-Food Canada, Ottawa in three intensive field campaigns (IFCs) during the summer of 2001 (IFC-1, June 13; IFC-2, June 26; and IFC-3, July 19). The CASI sensor was configured to 72 spectral bands with a spectral resolution of 7.6 nm in the spectral range of 408–947 nm. The spatial resolution of the data is approximately 2 m $\times$ 2 m. The three campaigns were planned to coincide with different phenological development stages of three crops: corn, wheat, and soybean. The LAI values of the three crops were measured in 14 sites of the crop field by Agriculture and Agri-Food Canada during IFC-2 and IFC-3; they were used as ground truth in the evaluation. The details of the CASI data, the crop conditions, and the color composite images of the three campaign datacubes have been reported by Hu et al.[5]

The raw datacubes from the three field campaigns were compressed using SAMVQ at CRs of 20:1, 30:1, and 50:1, and HSOCVQ at CRs of 10:1, 20:1, and 30:1. The compressed data was decompressed to produce the reconstructed datacubes. A reconstructed datacube is the same as the original except it has undergone the compression. For each campaign, the raw datacube and the six reconstructed datacubes were mixed with the original datacube, named using arbitrary number identifiers by an individual who was neither the evaluator nor the user, and then returned to the user for derivation of the LAI. They were referred to as "blind datacubes" because the user has no knowledge of which is the original datacube and which is the compressed datacube. The blind datacubes were preprocessed, radiometrically converted to radiance, and then converted to reflectance datacubes. LAI was derived from each of the reflectance datacubes using the algorithm developed by Haboudance et al.[6]

To evaluate the impact of preprocessing and radiometric conversion on data compression, the raw datacubes from the three campaigns were first converted to radiance using the same preprocessing coefficients and calibration coefficients as those used previously. The three radiance datacubes were compressed using both SAMVQ and HSOCVQ at the same compression ratios as those for the raw datacubes. The reconstructed radiance datacubes were mixed with the original radiance datacube and then sent back to the user with the blind names. The user converted the blind datacubes to reflectance and derived the LAI from each of the reflectance datacubes using the same processing as that for the raw datacubes.

The user utilized visual inspection as a qualitative measure to examine the spatial and temporal patterns of the LAI images derived from the blind datacubes. Based on visual inspection, the user qualitatively accepted all of the 42 blind datacubes (i.e., the 36 mixed compressed and the 6 original datacubes) because their spatial and temporal patterns are similar. Figure 6.7 shows the LAI images derived from the original IFC-2 datacube and its reconstructed datacubes using SAMVQ and HSOCVQ at a CR of 20:1, with preprocessing and radiometric conversion done after and before compression. It can be seen that spatial patterns of all the LAI images are similar. Almost no difference can be seen from the LAI images derived from the original and from the datacubes with preprocessing and radiometric conversion done after (compression applied to raw data) and before (compression applied to radiance data) compression. This indicates that the order of applying preprocessing and radiometric conversion before or after compression is not critical. The LAI images derived from the IFC-2 datacubes with other CRs and those derived from IFC-1 and IFC-3 datacubes show similar results.

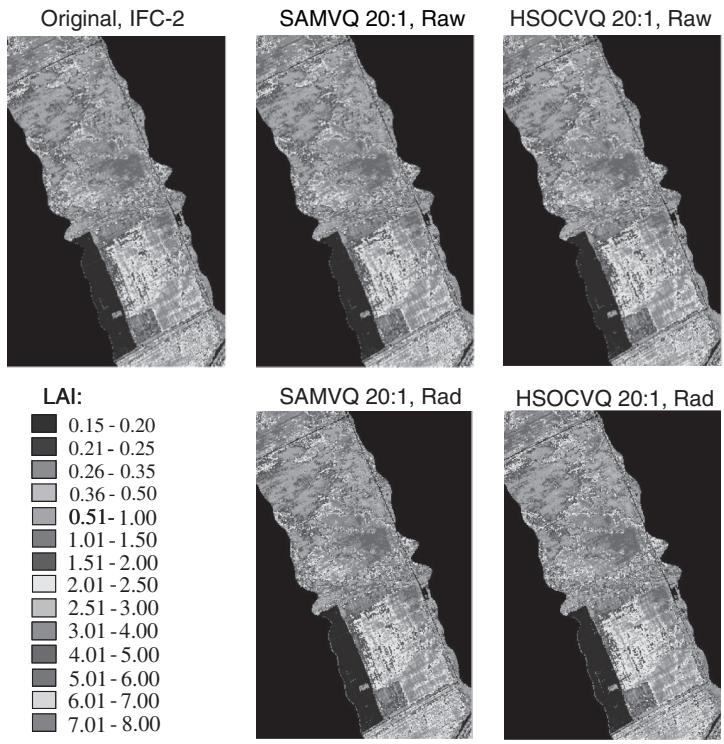


Figure 6.7 LAI images derived from the original IFC-2 datacube and from the compressed datacubes (20:1) with compression applied to the raw and to the radiance (Rad) datacubes (reprinted from Ref. 1). For a color version of this figure, see Plate 6 in the color plate section of this book.

Three quantitative measures were used to evaluate the impact: the correlation coefficient (R^2), the absolute root mean square error (A-RMSE), and the relative RMSE (R-RMSE) between the derived LAI of a blind datacube and the measured LAI (ground truth). From an application perspective, a blind datacube is acceptable if over 70% of the variation ($R^2 \geq 0.7$) in the measured LAI values can be explained by the derived LAI values of the blind datacube. A blind datacube is acceptable if its A-RMSE of the derived LAI is comparable with the standard deviation of the ground truth or if its R-RMSE is less than 10%. The criteria were selected by the user based on the retrieval accuracy of the LAI from the original reflectance data.[6] The measured LAI values range from 0.87–5.30 with a standard deviation of 1.40.

Table 6.3 lists the R^2, A-RMSE, and R-RMSE calculated between the derived LAI and the ground truth for corn, wheat, and soybean, with the compression applied to the raw (Raw) and to the radiance (Rad) datacubes. Both the original raw and the radiance datacubes were sent back to the user with blind names along with their compressed datacubes. The user produced the

Table 6.3 The R^2, the absolute RMSE, and the relative RMSE between the ground truth and the LAI derived from the compressed datacubes.

Compression Algorithm & Ratio	R^2		A-RMSE		R-RMSE (%)	
	Raw	**Rad**	**Raw**	**Rad**	**Raw**	**Rad**
Corn						
1:1 (Original)	0.872	0.872	1.576	1.576	5.422	5.422
SAMVQ 20:1	0.869	0.871	1.579	1.600	5.439	5.533
SAMVQ 30:1	0.868	0.875	1.584	1.583	5.445	5.514
SAMVQ 50:1	0.867	0.857	1.598	1.590	5.460	5.566
HSOCVQ 10:1	0.868	0.869	1.557	1.618	5.446	5.653
HSOCVQ 20:1	0.868	0.868	1.559	1.621	5.467	5.675
HSOCVQ 30:1	0.868	0.880	1.559	1.576	5.471	5.675
Wheat						
1:1 (Original)	0.955	0.955	0.559	0.559	8.089	8.089
SAMVQ 20:1	0.955	0.955	0.557	0.565	8.124	8.310
SAMVQ 30:1	0.956	0.955	0.558	0.569	8.134	8.235
SAMVQ 50:1	0.956	0.954	0.556	0.558	8.146	8.413
HSOCVQ 10:1	0.953	0.951	0.569	0.558	8.065	8.675
HSOCVQ 20:1	0.954	0.949	0.565	0.557	8.097	8.720
HSOCVQ 30:1	0.953	0.957	0.566	0.565	8.105	8.593
Soybean						
1:1 (Original)	0.998	0.998	1.387	1.387	5.996	5.996
SAMVQ 20:1	0.998	0.998	1.390	1.398	6.010	6.086
SAMVQ 30:1	0.998	0.998	1.384	1.405	6.010	6.135
SAMVQ 50:1	0.997	0.998	1.412	1.395	6.065	6.038
HSOCVQ 10:1	0.997	0.998	1.340	1.391	6.094	5.871
HSOCVQ 20:1	0.997	0.998	1.401	1.393	6.121	5.873
HSOCVQ 30:1	0.997	0.998	1.401	1.399	6.121	5.918

identical R^2, A-RMSE, and R-RMSE in both Raw and Rad cases (see the first row of each crop in the table), which indicates that the product algorithm is stable and repeatable. The user quantitatively accepted all of the blind datacubes because the R^2 corresponding to all of the datacubes are between 0.87 and 0.99, which is better than 0.7; the A-RMSEs are between 0.6 and 1.6, which is comparable to 1.4, the standard deviation of the ground truth; and the R-RMSEs are between 5.4% and 8.7%, which is smaller than 10%.

Table 6.3 shows that there is no significant difference between the R^2, A-RMSEs, and R-RMSEs for the raw cases and those for the radiance cases, indicating that preprocessing and radiometric conversion after or before compression has no impact on the LAI product.

6.3.3 Evaluation using target detection

Ground target detection using spectral unmixing was selected as a remote sensing application to evaluate the impact of preprocessing and radiometric conversion on data compression. The compressed datacubes, using SAMVQ and HSOCVQ at CRs of 10:1–30:1 with the compression applied to the raw and to the radiance data, were used to derive the fraction images for detecting targets. The targets derived from the compressed datacubes were compared with those derived from the original datacube. A double-blind test was adopted.

The hyperspectral datacube used in this evaluation was acquired with SFSI-II, which has been described in Section 6.2.1. The following eight processing steps were performed to generate the fraction images when a blind datacube to be evaluated was raw data:

1. Removal of periodic noise of raw data: The raw data is transformed into Fourier frequencies, and a notch filter is applied to remove the peaks that exceed 20 units.
2. Subtraction of dark current: The average of the dark current recorded at the beginning and the end of each flight line is subtracted from each pixel.
3. Correction of spectral distortion (smile):[7] The effect of the slit curvature of the instrument is corrected.
4. Correction of spatial distortion (keystone):[8] The geometric distortion caused by misalignment of the lens in the focal plane is corrected.
5. Radiometric conversion: A vicarious calibration is performed using calibration coefficients derived from a previous SFSI-II survey to convert the raw data into at-sensor radiance.
6. Removing random noise: Random noise in radiance is removed using an algorithm developed by Khurshid et al.[9] This noise removal significantly improves the data quality for remote sensing applications. (Section 6.4 describes this step and its effect in detail.)
7. Destriping: The intensity of the "bad" stripes is readjusted by applying a gain to them.

8. Spectral unmixing: Nine endmembers and nine ROIs are selected manually from the original datacube. They consist of five synthetic targets and four ground features (forest, gravel road, sand, and grass). Each ROI is used to extract the corresponding endmember spectrum from a blind datacube using constrained linear spectral unmixing. A fraction image corresponding to each endmember is produced for each blind datacube and used to identify the targets.

Processing steps 1–4 are the preprocessing; step 5 is the radiometric conversion that converts raw data to radiance data; and steps 6 and 7 are additional processing steps to clean up the artifacts in the radiance data before applying the application product algorithm.

When the compressed datacubes were already radiance data, only processing steps 7 and 8 were performed to generate the fraction images because steps 1–6 had already been performed and converted raw data to radiance before the compression.

The same four qualitative and quantitative criteria described in Section 6.2.2 were used to assess the impact of compression on target detection. Table 6.4 lists the evaluation score per target, total score, ranking, and user acceptability of the compressed datacubes with compression applied to the raw datacube (shaded columns) and on the corresponding Rad datacube.

When SAMVQ or HSOCVQ compression was applied to the Rad datacube (i.e., the compression applied after preprocessing and radiometric conversion), the SAMVQ-compressed datacube 10:1 got a total score of 18 out of the full score of 20 and was ranked #1. For targets polythene, cotton, and vinyl mat, this datacube got a full score of 4. The HSOCVQ-compressed datacubes 10:1 and 20:1 both got a total score of 15 and were ranked #2. There is a fault of detection for the HSOCVQ-compressed datacube 10:1—it got only 1 point out of 4 for target cotton. SAMVQ-compressed datacubes 20:1 and 30:1 got total scores of 14 and 13, respectively, and were ranked #3 and #4. The user accepted all five of the compressed Rad datacubes based on the criteria.

When SAMVQ or HSOCVQ compression was applied to the Raw datacube (i.e., applied before preprocessing and radiometric conversion), the compressed datacube at a CR of 10:1 using SAMVQ got a total score of 15 out of the full score of 20, was ranked #1, and was accepted. The HSOCVQ-compressed datacube 10:1 got a total score of 11 and was ranked #2. The SAMVQ-compressed datacubes 20:1 and 30:1 got total scores of 10 and 9, respectively, and were ranked #3 and #4. The user marginally accepted them. The HSOCVQ-compressed datacube 20:1 got a total score of 5 and was rejected.

The total scores for the compressed datacubes with compression applied to the raw data are all smaller than those with compression applied to the

Table 6.4 Score per target, tota score, rank, and user acceptab ty of the compressed SFSI-II datacube w th compress on app ed to the raw datacube and to the rad ance (Rad) datacube.

Compression Algorithm & Ratio	Score per Target										Total Score per Datacube		Rank		Acceptance	
	Awning (ROI 1 size = 71)		Polythene (ROI 2 size = 29)		Plastic tarp (ROI 3 size = 29)		Cotton (ROI 4 size = 28)		Vinyl mat (ROI 5 size = 80)							
	Raw	Rad	Raw	Rad	Raw	Rad	Raw	Rad	Raw	Rad	Raw	Rad	Raw	Rad	Raw	Rad
SAMVQ 10:1	3	3	4	4	3	3	2	4	3	4	15	18	1	1	A	A
SAMVQ 20:1	2	3	1	3	2	2	2	3	3	3	10	14	3	3	M	A
SAMVQ 30:1	2	3	1	2	2	2	2	3	2	3	9	13	4	4	M	A
HSOCVQ 10:1	2	3	2	4	2	3	2	1	3	4	11	15	2	2	M	A
HSOCVQ 20:1	1	3	2	4	1	2	0	3	1	3	5	15	5	2	N	A

(A Acceptable, M Marginally Acceptable, N Not Acceptable)

radiance data. The evaluation results of this application indicate that preprocessing and radiometric conversion applied before or after compression affects compression performance. Compression of the radiance data produces higher evaluation scores and better user acceptability.

6.4 The Effect of Radiance-Data Random Noise on Compression Performance

The evaluation results in Section 6.3.3 indicate that it might be necessary to apply preprocessing and radiometric conversion onboard to convert raw data to radiance before compression because it preserves more information and produces better user acceptability. On the other hand, as described in Section 6.3.1, additional errors or artifacts are induced in the process of the preprocessing and radiometric conversion. This section assesses the impact of removing random noise of radiance data on data compression onboard a hyperspectral satellite using the same target detection application. The same SFSI-II datacube used in Section 6.3 was tested. The impact on data compression was evaluated using both the statistical measures and a remote sensing application. The evaluation results show that compression of radiance data after removal of random noise produces better reconstruction fidelity and much higher evaluation scores for the remote sensing application than compression of radiance data without removal of random noise. The evaluation results indicate that random noise of radiance data should be removed before compression if it is applied to radiance data onboard.

6.4.1 Data processing procedure

Figure 6.8 shows the block diagram of the data processing procedure for a target detection application using a spectral unmixing approach. There are

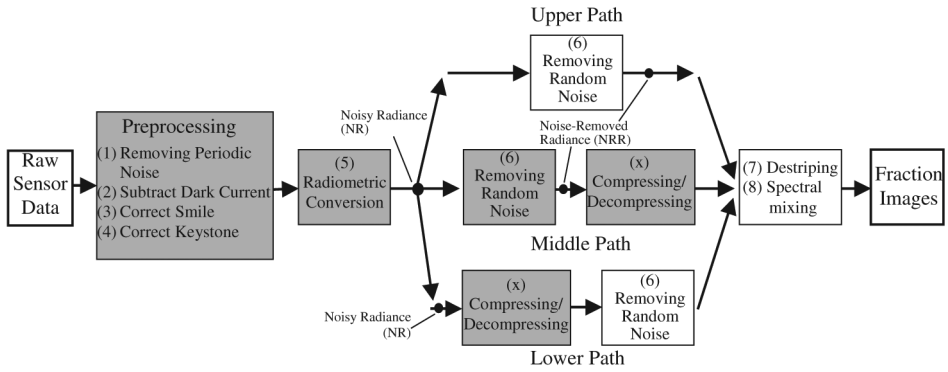


Figure 6.8 Block diagram of the data processing procedures (shaded boxes denote onboard processing if onboard data compression is deployed).

three paths of processing procedure in the figure. In the upper path, which does not contain the onboard data compression, there are eight processing steps to generate the fraction images from the acquired raw SFSI-II hyperspectral data, as described in Section 6.3.3.

Processing steps 1–4 are the preprocessing steps, and the raw data after preprocessing is converted to radiance in step 5. The middle and lower paths shown in Fig. 6.8 are the subject of this section. In the middle-path processing procedure, the noisy radiance data underwent removal of random noise before being sent to compression. The input of the compression is the noise-removed radiance (NRR) data. In the lower-path processing procedure, the noisy radiance data was sent to compression directly, and the random noise was removed after compression. The input of the compression is the noisy radiance (NR) data. In fact, the evaluation of the impact of preprocessing and radiometric conversion on compression described in Section 6.3.3 includes the upper- and middle-path processing procedures.

Random noise of the radiance data is removed by applying a spectral–spatial smoothing operation based on the known noise characteristics of the SFSI-II sensor. This is done by averaging together spectra from a number of different pixels that are equivalent in light of the known noise characteristics of the SFSI-II sensor. This noise removal can significantly improve the data quality for remote sensing applications. Figure 6.9 shows the typical radiance spectra of the test SFSI-II datacube over the site before and after noise removal. The H_2O band at 1470 nm and the water vapor value cannot be properly estimated before the random noise is removed, but they can be well estimated in the NRR spectra.

6.4.2 Evaluation results using statistical measures

This section begins by evaluating the effect of removing random noise on data compression performance using statistical measures. The NRR and NR datacubes were compressed using SAMVQ at ratios of 10:1–30:1, and HSOCVQ at ratios of 10:1–20:1. The same evaluation metrics as in Section 6.2.2—RMSE, SNR, and %E—are used.

Table 6.5 lists the metrics of the reconstructed datacubes for the NRR (shaded columns) and NR datacubes. Compression of the NRR datacube produces better reconstruction fidelity than that of the NR datacube for the same compression algorithm and same compression ratio. The compression of the NRR datacube attains a SNR gain between 7.4–9.7 dB and a %E reduction between 1.8–2.3%, which are due to the removal of random noise of the radiance data. The NRR datacube is more compressible than the NR datacube. The NRR compressed datacubes with higher reconstruction fidelity are expected to produce more-accurate fraction images for ground-target detection applications.

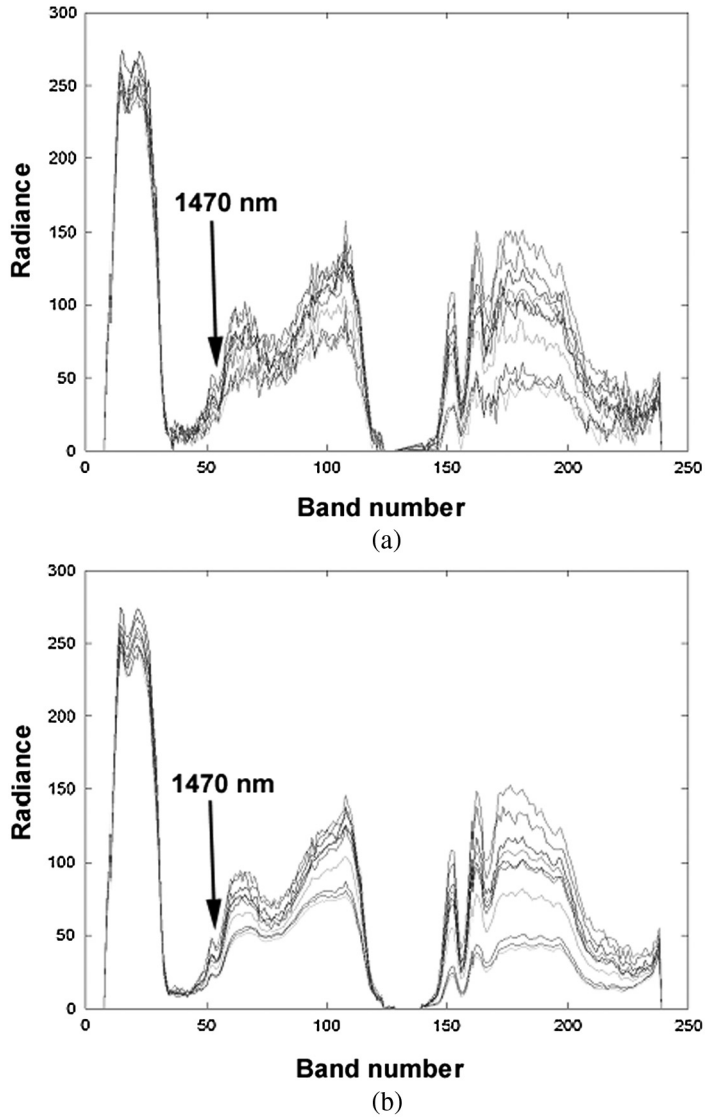


Figure 6.9 Spectra of the SFSI-II radiance over the target site (a) before and (b) after removal of random noise. Band numbers start at 1100 nm (band 0) and end at 2500 nm (band 240).

6.4.3 Evaluation results using target detection

This section uses ground target detection of hyperspectral data as a remote sensing application to evaluate the impact of removing radiance-data random noise on data compression. The NRR and NR compressed datacubes using SAMVQ at ratios of 10:1–30:1 and HSOCVQ at ratios of 10:1–20:1 were decompressed to produce the reconstructed datacubes for evaluation. The

Table 6.5 Statistical measures of reconstructed data with compression applied to noise-removed radiance (NRR) and noisy radiance (NR) data.

Compression Algorithm & Ratio	RMSE			SNR (dB)			Percentage Error (%)		
	NRR	NR	NRR-NR	NRR	NR	NRR-NR	NRR	NR	NRR-NR
SAMVQ 10:1	8.93	21.92	12.99	41.26	31.59	9.67	0.68	2.46	1.78
SAMVQ 20:1	12.92	28.00	15.08	38.05	29.46	9.67	0.95	2.96	1.87
SAMVQ 30:1	15.22	33.35	18.13	36.62	27.94	8.59	1.09	3.28	2.19
HSOCVQ 10:1	17.43	32.63	15.20	35.45	28.08	8.68	1.33	3.64	2.31
HSOCVQ 20:1	20.85	36.05	15.20	33.89	27.21	6.37	1.59	4.05	2.46

reconstructed datacubes are of the same size and format as the original but have undergone compression. A double-blind test was used to reduce self-deception and bias in the evaluation of the impact, and the compressed datacubes were named with an arbitrary number when returned to the user, who then derived the fraction images for detecting the targets and evaluated the impact based on predefined criteria by comparing the products derived from the compressed datacubes with those derived from the original datacube. The user had no knowledge of the compression status of the datacubes evaluated. The same four predefined criteria proposed in Section 6.2.2 were used to assess the impact; 1 point was scored for each criterion met.

The evaluation was performed on a ROI-by-ROI basis. The full score for a target ROI is 4. The full score for a compressed datacube is 20, as there are 5 targets. Table 6.6 lists the evaluation score per target and the total score per datacube of the blind-compressed datacubes with compression applied to the NRR datacube (shaded columns) and to the NR datacube.

When the compression was applied to the NRR datacubes (i.e., the middle path in Fig. 6.8, wherein the compression is applied to radiance data

Table 6.6 Application evaluation score per target and total score per compressed datacube with compression applied to a noise-removed radiance (NRR) datacube and to a noisy radiance (NR) datacube.

Compression Algorithm & Ratio	Score per Target											
	Awning (ROI 1 size 71)		Polythene (ROI 2 size 29)		Plastic tarp (ROI 3 size 29)		Cotton (ROI 4 size 28)		Vinyl mat (ROI 5 size 80)		Total Score per Datacube	
	NRR	NR	NRR	NR	NRR	NR	NRR	NR	NRR	NR	NRR	NR
SAMVQ - 10:1	3	2	4	0	3	1	4	1	4	1	18	5
SAMVQ - 20:1	3	2	3	0	2	1	3	1	3	1	14	5
SAMVQ - 30:1	3	1	2	0	2	0	3	1	3	1	13	3
HSOCVQ - 10:1	3	1	4	0	3	1	1	1	4	3	15	6
HSOCVQ - 20:1	3	1	4	0	2	1	3	2	3	3	15	7

after removing random noise), the SAMVQ-compressed datacube 10:1 got a total score of 18 out of the full score of 20 and was ranked #1. For targets polythene, cotton, and vinyl mat, this datacube got a full score of 4. The HSOCVQ-compressed datacubes 10:1 and 20:1 got a total score of 15 out of 20 and were ranked #2. SAMVQ-compressed datacubes 20:1 and 30:1 got total scores of 14 and 13, respectively, and were ranked #3 and #4. The user accepted all five of the NRR-compressed datacubes based on the predefined criteria.

When the compression was applied to the NR datacubes (i.e., the lower path in Fig. 6.8, wherein the compression was applied before removal of random noise), the compressed datacubes at compression ratios of 10:1 and 20:1 using SAMVQ both got a total score of 5 out of 20; the SAMVQ-compressed datacube 30:1 got a total score of 3; and the HSOCVQ-compressed datacubes 10:1 and 20:1 got total scores of 6 and 7, respectively.

The total evaluation score of a NR-compressed datacube is much smaller than that of a NRR-compressed datacube with the same CR and the same compression algorithm. These evaluation scores are consistent with the statistical measures of the compressed datacubes shown in Section 6.4.2. The evaluation results using this application show that removing radiance-data random noise has a significant impact on SAMVQ or HSOCVQ compression performance. Compression of the NRR datacubes by applying the removal of random noise in the radiance datacube before compression produces much-higher evaluation scores and better user acceptability than that of the NR datacubes.

The reason for the poor evaluation scores of the compressed datacubes whose random noise were removed after SAMVQ or HSOCVQ compression is that the noise removal is not effective. This is probably because the noise-removal algorithm was designed to remove random noise in the original SFSI-II radiance data. The SAMVQ and HSOCVQ algorithms are lossy data-compression algorithms. They act like a low-pass filter, suppressing the high-frequency noise during the compression.[10] The random noise in the reconstructed datacubes is not the same as that in the original radiance data. The noise-removal algorithm might not well remove the random noise in the reconstructed SFSI-II radiance data after compression using SAMVQ and HSOCVQ. The noise-removal algorithm probably needs to be redesigned in order to effectively remove random noise in the compressed radiance data.

6.5 Effect of Keystone and Smile on Compression Performance

This section evaluates the effect of spatial distortion (keystone) and spectral distortion (smile) of hyperspectral sensors to examine whether these distortions have any impact on compression performance, and thus whether these distortions should be corrected onboard before compression.

In an imaging spectrometer, keystone refers to the across-track spatial misregistration of the ground sample pixels of the various spectral bands of the spectrograph. It is caused by geometric distortion, as can be seen in camera lenses, by chromatic aberration, or a combination of the two. Smile, also known as spectral line curvature, refers to the spatial nonlinearity of a monochromatic image of a straight entrance slit as it appears in the focal plane of a spectrograph. It is caused by a dispersion element, prism, or grating, or by aberrations in the collimator and imaging optics. These distortions have the potential to affect compression performance.

A datacube acquired using CASI in an application of boreal forest environment was used as a test datacube. It has 72 spectral bands with a spectral sampling distance of approximately 7.2 nm. The half-bandwidth used is 4.2 nm. Keystone and smile were simulated and then incorporated into the test datacube, as shown in Fig. 6.10.

The generated keystone was to simulate the shift of nominal data due to a linear keystone whose maximum amplitude can be specified by a user. The amplitude of the keystone is defined as the maximum angular shift in the pixel center position from the nominal value in the full detector array. Because the keystone is linear and symmetric around the array center, the maximum keystone is located at the array edges (i.e., the first and last pixel in the array, and for the first and last bands in the array). The generated keystone is the same from one focal plane frame to the next.

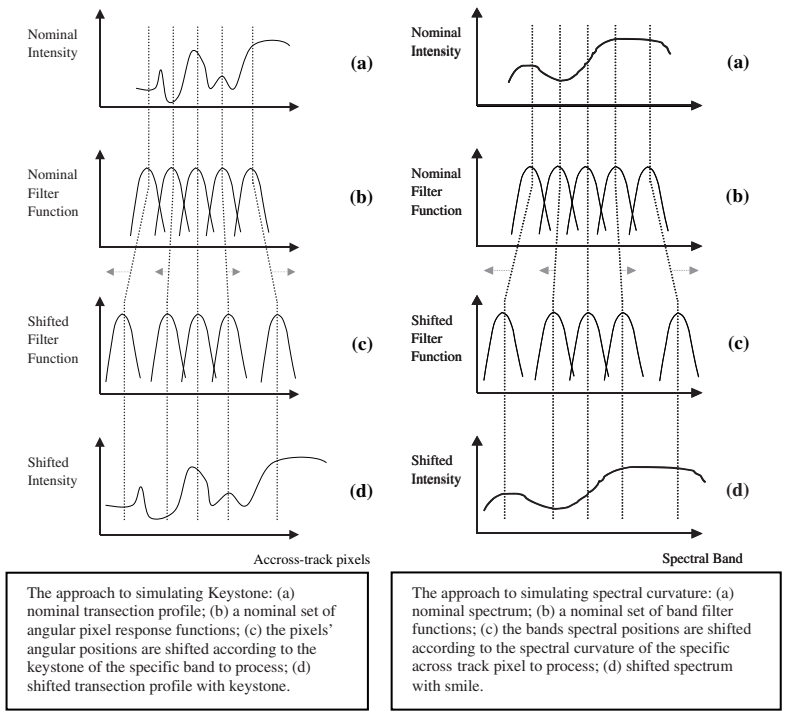
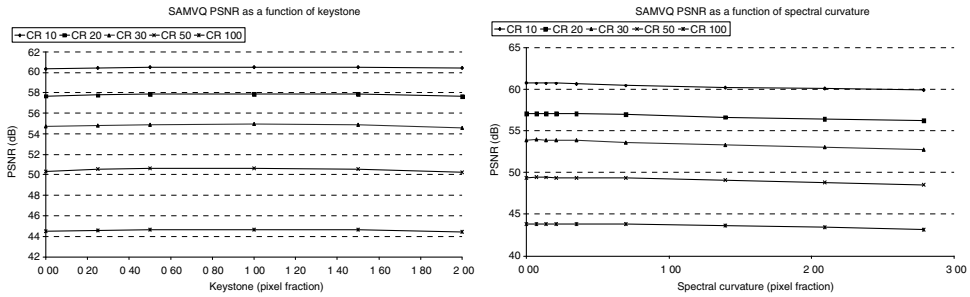


Figure 6.10 Simulation of keystone (left) and smile (right) of hyperspectral datacubes.



(a) (b)

Figure 6.11 Compression fidelity (PSNR) produced using SAMVQ as a function of magnitude of (a) keystone and (b) smile.

The generated smile was to simulate the shift of nominal data due to a quadratic spectral line curvature whose maximum amplitude can also be specified by a user. The simulation approach assumes that the diffraction slit is curved so that the smile is minimal in the middle of the array. The amplitude for the smile is defined as the maximum spectral shift in the band center wavelength from the nominal value in the full detector array. Because the smile is quadratic and symmetric around the array center, the maximum smile is located at the array edges.

Compression was applied to the original test datacube and the datacubes with simulated keystone and smile. Figure 6.11 shows the curves of compression fidelity (PSNR) produced using SAMVQ as a function of magnitude of keystone and smile. Experimental results showed that keystone has little or no impact on the compression fidelity produced by both SAMVQ and HSOCVQ. The PSNR fidelity loss is <1 dB. Smile has little to some impact on the compression fidelity. The PSNR fidelity loss is typically 2 dB with HSOCVQ and <1 dB with SAMVQ.

6.6 Enhancing the Resilience of Compressed Data to Bit Errors in the Downlink Channel

In SAMVQ and HSCOVQ, compressed data includes index maps and codebooks. If one bit or multiple bits of an index in the index map are flipped, caused by the single-event upsets in the downlink channel, the decompression of this index on the ground will be in error. A wrong codevector will be picked up in the codebook to reconstruct the spectrum for the ground sample at the location of that index. These bit errors will not propagate to other indexes of the index map. If one bit or multiple bits of an element (i.e., spectral value) of a codevector in the codebook are flipped due to single-event upsets, this element of the codevector is damaged and is in error. In decompression on the ground, all of the ground samples with the

same index associated to this codevector will be reconstructed using this corrupted codevector; these ground samples will all have a wrong value in the spectral band corresponding to that corrupted element. The single-bit error or multiple-bit burst errors within a codevector element are propagated to an entire datacube, but they corrupt only in a specific band. In both cases presented earlier, a single-bit error and multiple-bit burst errors will not result in data loss; instead, they will cause inaccurate reconstruction of the compressed data. This is the advantage of VQ-based compression techniques compared to other compression techniques.

Although both SAMVQ and HSCOVQ are more bit-error resistant than the traditional compression algorithms, experimental results show that when the bit-error rate (BER) exceeds 10^{-6}, the compression fidelity starts to drop. This section explores the benefits of employing triple-module redundancy and forward error correction on top of data compression (SAMVQ or HSOCVQ) to deal with higher BERs. In particular, it is shown that proper use of triple-module redundancy and convolutional codes can improve the resilience of compressed hyperspectral data against bit errors by close to two orders of magnitude.

6.6.1 Triple-module redundancy used in the header of the codebook and index map

As described in Chapter 5, SAMVQ generates a codebook and an index map in each approximation stage of the multistage compression. If S approximation stages have undergone compression, S codebooks and S index maps will have been generated. HSOCVQ generates only one codebook and one index map. In order to reconstruct the compressed data on the decoder side, the key parameters of the compression (such as codebook size, number of spectral bands, spatial size of the datacube, etc.) need to be stored together with the codebooks and index maps. In SAMVQ and HSOCVQ, these key parameters are stored in the header of a codebook file and an index-map file. The header of a codebook file and an index-map file is critical because it is more sensitive to bit errors than the body of the codebooks and index maps. For instance, if the bits of a codebook size N are flipped due to single-event upsets, the reconstruction will be in error.

In order to prevent the key parameters from corrupting, triple-module redundancy has been used in the header in both SAMVQ and HSOCVQ. Tables 6.7–6.9 show the layout of the header of the index map and codebook of the SAMVQ algorithm. There are four parameters stored in the header of an index-map file: the stage number (*Stage#*), the number of bits per index (*mapbits*), the number of rows of the datacube scene (*Nr*), and the number of columns of the datacube scene (*Nc*). The *Stage#* is assigned for one byte (char) and can record up to 256 stages. The *mapbits* is equal to $\log_2 N$ (N is the size of the codebook of the stage). One byte is

Table 6.7 Header of the index-map file of a SAMVQ algorithm.

Data Type	Item	# of Bytes
char	*Stage#*	1
char	*mapbits*	1
short	*Nr*	2
short	*Nc*	2
char	*mapbits*	1
short	*Nr*	2
short	*Nc*	2
char	*mapbits*	1
short	*Nr*	2
short	*Nc*	2

sufficient to record it. Both *Nr* and *Nc* are encoded using 2 bytes (short) that can record from 0 to 65535.

Among the four stored parameters, *Stage#* is informative and often carried in the file naming. If it is corrupted, the reconstruction will not be affected; however, the other three parameters are critical and will affect the reconstruction if they are corrupted. These three parameters are recorded in triplicate in the header. On the decoder side, the values of these three

Table 6.8 Header of a codebook file of a SAMVQ algorithm with the shortest fitting word-length per codevector. Note that INDEXSAM = char in this example.

Data Type	Item	# of Bytes
char	*Stage#*	1
INDEXSAM	*N*	1
short	*Nb*	2
INDEXSAM	*N*	1
short	*Nb*	2
INDEXSAM	*N*	1
short	*Nb*	2
char	*vectorbits*[*0*]	1
char	*vectorbits*[*1*]	1
char	*vectorbits*[*2*]	1
...		
char	*vectorbits*[*N − 1*]	1
char	*vectorbits*[*0*]	1
char	*vectorbits*[*1*]	1
char	*vectorbits*[*2*]	1
...		
char	*vectorbits*[*N − 1*]	1
char	*vectorbits*[*0*]	1
char	*vectorbits*[*1*]	1
char	*vectorbits*[*2*]	1
...		
char	*vectorbits*[*N − 1*]	1

Table 6.9 Header of the codebook file of a SAMVQ algorithm with the shortest fitting word-length per codebook.

Data Type	Item	# of bytes
char	*Stage#*	1
INDEXSAM	*N*	1
short	*Nb*	2
char	*vectorbits*	1
INDEXSAM	*N*	1
short	*Nb*	2
char	*vectorbits*	1
INDEXSAM	*N*	1
short	*Nb*	2
char	*vectorbits*	1

parameters will be decided based on the majority voting of the triple-module redundancy to prevent bit errors caused by single-event upsets. The overhead of this redundancy is 10 additional bytes whose effect on the compression ratio can be negligible compared to the total number of bytes of the index-map file.

Section 5.2.4 notes that the word-length of a codebook generated in an approximation stage can vary. Two schemes have been adopted to take this feature into account when encoding the codebook generated in an approximation stage: one uses the shortest fitting word-length for each single codevector of the stage codebook, and the second uses the shortest fitting word-length for all of the codevectors of the entire stage codebook. Table 6.8 illustrates the header for the first scheme. As in the index-map file, *Stage#* is informative and assigned for one byte without triple-module redundancy. The other three parameters are critical and encoded with triple-module redundancy. These parameters are codebook size N (i.e., the total number of codevectors within a codebook), number of spectral bands Nb (i.e., the total number of elements of a codevector), and the word-length of each codevector *vectorbits*[k], $k = 0, 1, 2,$ N 1. The data type of the codebook size N is a macro-variable in the c-code of the SAMVQ algorithm, which is determined based on the maximum size of the codebooks. If it is defined as INDEXSAM = char, the maximum codebook size allowed is $N = 256$. If it is defined as INDEXSAM = short, the maximum codebook size allowed is $N = 65535$. The number of spectral bands Nb is assigned to 2 bytes (short), which is sufficient to cover any value of number of spectral bands up to 65535. The number of bits of the *vectorbits*[k] is between 1 and 16 for short (2-byte) data or between 1 and 32 for long (4-byte) data. One byte is assigned to record the *vectorbits*[k] in the header.

Table 6.9 depicts the header for the second scheme of the codebook file, where a single word-length *vectorbits* is used to encode all of the codevectors within the codebook. This *vectorbits* is equal to the largest one in the N *vectorbits*[k], $k = 0, 1, 2, N$ 1 in order to cover all of the codevectors.

There is a trade-off between the two schemes in encoding the word-length of the codevectors in the codebook file. The first scheme needs to record N *vectorbits* in the header. The total cost is $3N$ bytes in the header with the triple-module redundancy; however, this scheme saves on the total number of bits in encoding the codevectors, as some of codevectors are encoded using very short word-lengths. The second scheme records only one *vectorbits* in the header. The total cost is 3 bytes in the header with the triple module redundancy. However, this scheme uses more bits in encoding the codevectors, as some codevectors whose amplitude is very small are encoded using the maximum word-length. Experimental results showed that the first scheme yields slightly better compression ratios than the second scheme. The first scheme is selected as the default algorithm for SAMVQ.

6.6.2 Convolutional codes

There exist many efficient channel codes that permit reliable communication of information without sacrificing the transmission rate, wasting the frequency bandwidth, or wasting energy. In particular, convolutional codes[11] have often been used to improve the performance of deep-space and satellite communications. A convolutional code is partly specified by a triplet (k, n, L_c), where k and n denote, at any given time unit, the number of information bits and the number of coded bits, respectively, and L_c, the so-called constraint length, denotes the number of successive encoder input blocks that affect the encoder output block at any given time. In other words, the encoder output block $\mathbf{v}_l$ (of n-size bits) at time l depends not only on the encoder input block $\mathbf{u}_l$ (of k-size bits) at time l but also on $L_c - 1$ encoder input blocks $\mathbf{u}_{l-1}, \mathbf{u}_{l-2}, \ldots, \mathbf{u}_{l-(Lc-1)}$ (each of k-size bits) at times $l - 1, l - 2, \ldots, l - (L_c - 1)$, respectively. The nominal code rate R of a convolutional code is k/n. The encoding process of a convolutional code is very simple, requiring only delays and adders, which is a great advantage for onboard use. The decoding process of a convolutional code, on the other hand, is more sophisticated and requires implementing the Viterbi algorithm,[12] a dynamic programming algorithm used to find the most-likely sequence of states. Nevertheless, the decoding process is perfectly capable on the ground.

This section employs the convolutional codes recommended by the CCSDS[13] for protecting compressed hyperspectral data against the errors induced by the noise in the downlink channel. The basic recommended convolutional code is partly specified by (1, 2, 7), has a rate $R = 1/2$ (i.e., 50% overhead), and is well suited for channels with predominantly additive white Gaussian noise (AWGN). The encoder of this code is shown in Fig. 6.12. For this code, the encoder output block $\mathbf{v}_l$ (of 2-size bits) at time l depends on

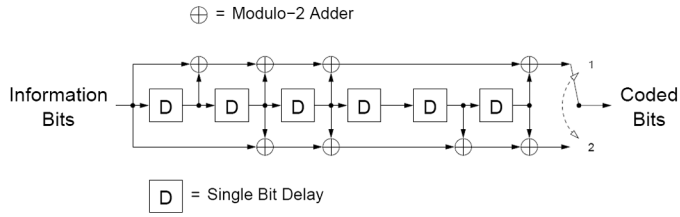


Figure 6.12 Encoder of a 64-state binary linear convolutional code of rate $R = 1/2$.

the encoder input blocks $\mathbf{u}_l, \mathbf{u}_{l-1}, \ldots, \mathbf{u}_{l-6}$ (each of 1-size bit) at times $l, l-1, \ldots, l-6$, respectively, as follows:

$$\mathbf{v}_{1,l} = \mathbf{u}_l \oplus \mathbf{u}_{l-1} \oplus \mathbf{u}_{l-2} \oplus \mathbf{u}_{l-3} \oplus \mathbf{u}_{l-6}, \tag{6.2}$$

$$\mathbf{v}_{2l} = \mathbf{u}_l \oplus \mathbf{u}_{l-2} \oplus \mathbf{u}_{l-2} \oplus \mathbf{u}_{l-5} \oplus \mathbf{u}_{l-6}. \tag{6.3}$$

When higher code rates (i.e., lower overhead) at the expense of lower error-correcting capability is desired, the basic recommended convolutional code may simply be punctured.[13] For instance, if rate $R = 2/3$ (i.e., 33.33% overhead) is desired instead of $R = 1/2$, $\mathbf{v} = (v_{1,0}, v_{2,0}, v_{1,1}, v_{2,1}, v_{1,2}, v_{2,2}, v_{1,3}, v_{2,3}, \ldots)$; the sequence of the coded bits has to be punctured according to the puncturing pattern of SAMVQ and HSOCVQ algorithms into $\mathbf{v}_{\text{punctured}} = (v_{1,0}, v_{2,0}, v_{2,1}, v_{1,2}, v_{2,2}, v_{2,3}, \ldots)$ before transmission over the channel. Clearly, although four coded bits are generated by the encoder for every two information bits, only three out of four are actually transmitted, hence, the rate $R = 2/3$.

The encoder for the convolutional codes recommended by the CCSDS is, in fact, a state machine with $2^6 = 64$ different states. State machines are usually characterized by state-transition diagrams, or alternatively, by trellis diagrams (note that a trellis diagram is a state-transition diagram spread in time). Due to the large number of states in the state machine associated with the recommended convolutional code, it is not easy to properly depict either of the previously mentioned diagrams. However, if the states are numbered from 0 to 63 based on the contents of the delay units in the delay line, it is not difficult to see that S_l, the state number at time l, is $[S_{l-1}/2]$ if $\mathbf{u}_l = 0$ and is $32 + [S_{l-1}/2]$ if $\mathbf{u}_l = 1$, where S_{l-1} is the state number at time $l-1$. For instance, if $S_{l-1} = 21$ and $\mathbf{u}_l = 1$, then $S_l = 32 + [21/2] = 42$. Figure 6.13 shows the building block of both the state-transition diagram and the trellis diagram characterizing this state machine. Note that emanating from each state are two arcs entering into two different states, one corresponding to the information bit that is 0, and the other corresponding to the information bit that is 1. Similarly, but not shown in the figure, entering into each state are two arcs emanating from

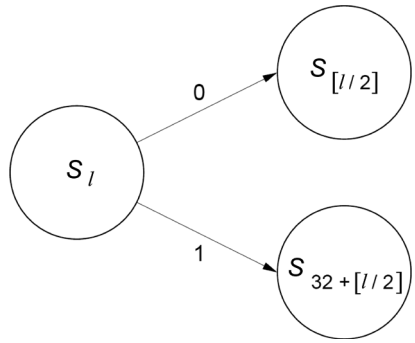


Figure 6.13 Building block of the characterizing diagrams associated with the recommended convolutional code.

two different states. This subtle observation greatly simplifies the implementation of the Viterbi algorithm for this code.

6.6.3 Viterbi algorithm

As mentioned previously, after the encoding process, the sequence of coded bits $\mathbf{v}$ is transmitted over a communication channel. The communication channel discussed in this section is the downlink channel between a hyperspectral satellite and a ground receiving station. This downlink channel can be modeled by a binary-input power-limited Gaussian channel (BIPLGC) for which the (channel) output $\mathbf{Y}$ is given by $\mathbf{Y} = \mathbf{X} + \mathbf{Z}$, where $\mathbf{X}$ is the binary (channel) input chosen from $\{-1, +1\}$, and $\mathbf{Z}$ is a zero-mean Gaussian random variable specified by its variance σ^2. Note that before transmitting over the downlink channel, coded bits 0 and 1 are mapped into $+1$ and -1, respectively. Let $\mathbf{v}^{\pm}$ denote the resulting sequence after the mapping. At the output of the downlink channel, the received sequence is

$$\hat{\mathbf{v}} = \mathbf{v}^{\pm} + \mathbf{n}, \tag{6.4}$$

where $\mathbf{n}$ is the noise sequence. The channel decoder has to recover $\mathbf{v}$ based on $\hat{\mathbf{v}}$. It is not difficult to see that the decoding problem is a minimum distance detection (MDD) problem, i.e., the channel decoder has to find a valid sequence of coded bits $\mathbf{v}$ such that $\|\hat{\mathbf{v}} - \mathbf{v}^{\pm}\|^2$, or equivalently $\sum_l \left|\left|\hat{v}_l - v_l^{\pm}\right|\right|^2$, is minimum. The Viterbi algorithm, which is a dynamic programming algorithm applied to a trellis diagram, can efficiently solve this problem. The following observations must be made in order to determine how the Viterbi algorithm works:

1. Each path in the trellis diagram of the code corresponds to a unique sequence of coded bits. As such, if each branch in the trellis diagram is assigned a metric equal to $\|\hat{\mathbf{v}}_l - \mathbf{b}_l\|^2$, where $\mathbf{b}_l$ is the output n-tuple associated with that branch, then the MDD problem becomes equivalent

to finding π, the path with the smallest overall metric, in the trellis diagram.

2. Consider a state through which π passes at time l. The initial segment of π from time 0 to time l has the smallest accumulated metric amongst the initial segments of all other paths passing through this state at time l. Therefore, it suffices at time l to determine and retain for each state only the path with the smallest accumulated metric, called the survivor.
3. The survivors at time l may be determined from the survivors at time $l - 1$ by a recursive add–compare–select operation as follows:
   a. For each branch from a state at time $l - 1$ to a state at time l, calculate the metric of that branch and add it to the metric of the survivor at time $l - 1$ to get a candidate path metric at time l.
   b. For each state at time l, compare the metrics of the candidate paths arriving at that state.
   c. For each state at time l, select the path corresponding to the smallest metric and store it as the survivor.

It should be clear now that the Viterbi algorithm can solve the MDD problem for a trellis diagram by finding π through recursively performing the add–compare–select operation mentioned in point 3. Equally important is that the regular recursive structure of the Viterbi algorithm makes it very attractive for software and hardware implementations. Note that the complexity of the algorithm is proportional to the number of the branches per unit of time in the trellis diagram, which is determined by the number of states in the state-transition diagram.

6.6.4 Simulation results

Two different convolutional codes have been used to protect three different compressed hyperspectral datacubes before transmitting the data over simulated noisy channels. At the receiver, after decoding and decompressing, the hyperspectral data is reconstructed to measure the SNR fidelity loss.

The two convolutional codes used are of rates $R = 4/5$ (i.e., 20% overhead) and $R = 7/8$ (i.e., 12.5% overhead), which are derived from the basic recommended convolutional code introduced in Section 6.6.1. More specifically, $R = 4/5$ and $R = 7/8$ are obtained by puncturing the output of the channel encoder according to the puncturing patterns [1000; 1111] and [1000101; 1111010], respectively. Note that before running the Viterbi algorithm, the receiver has to insert a '0' in all of the punctured positions.

The two hyperspectral datacubes tested in the simulations were acquired by the CASI and the AVIRIS hyperspectral sensors. The CASI datacube has been compressed by SAMVQ with a compression ratio of 10:1. The AVIRIS

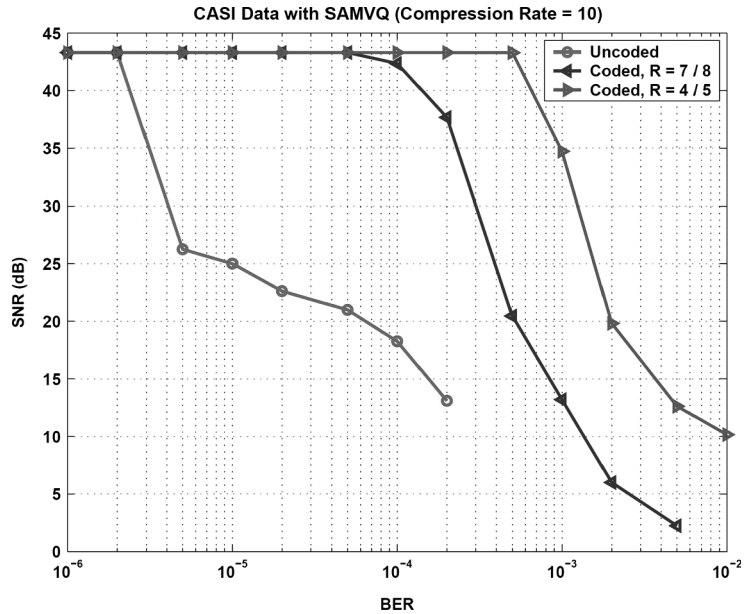


Figure 6.14 Average SNR loss of SAMVQ-compressed data with and without channel coding as a function of bit-error rate (BER) for the CASI datacube.

datacube, captured at Cuprite, NV, USA, has been compressed using SAMVQ and HSOCVQ at CRs of 14:1 and 10:1, respectively.

Figure 6.14 depicts the SNR versus the BER for the CASI datacube compressed by SAMVQ at a compression ratio of 10:1 with and without channel coding. The SNR of the reconstructed datacube is 43.30 dB without any single-event upsets. As can be seen in the figure, while the uncoded compressed data cannot tolerate single-event upsets resulting in BERs greater than 2×10^{-6}, the coded compressed data exhibits a far greater resilience to such errors: by channel coding with an overhead of 12.5% or 20%, BERs as high as 5×10^{-5} or 5×10^{-4}, respectively, are perfectly endured. Note that at BER $= 5 \times 10^{-5}$, the SNR fidelity loss for the uncoded case is 22.39 dB.

Figure 6.15 depicts the average SNR versus the BER for the AVIRIS Cuprite datacube compressed by SAMVQ at a CR of 14:1. The SNR of the reconstructed datacube is 52.99 dB without any single-event upsets. As can be seen in the figure, although the uncoded compressed data cannot endure single-event upsets resulting in BERs greater than 2×10^{-6}, the channel-coded compressed data can easily tolerate BERs as high as 1×10^{-4} or 5×10^{-4} with overheads of 12.5% or 20%, respectively. Note that at BER $= 1 \times 10^{-4}$, the SNR fidelity loss for the uncoded case is 32.69 dB.

Figure 6.16 depicts the average SNR versus the BER for the AVIRIS Cuprite datacube compressed using HSOCVQ. The SNR for this datacube is

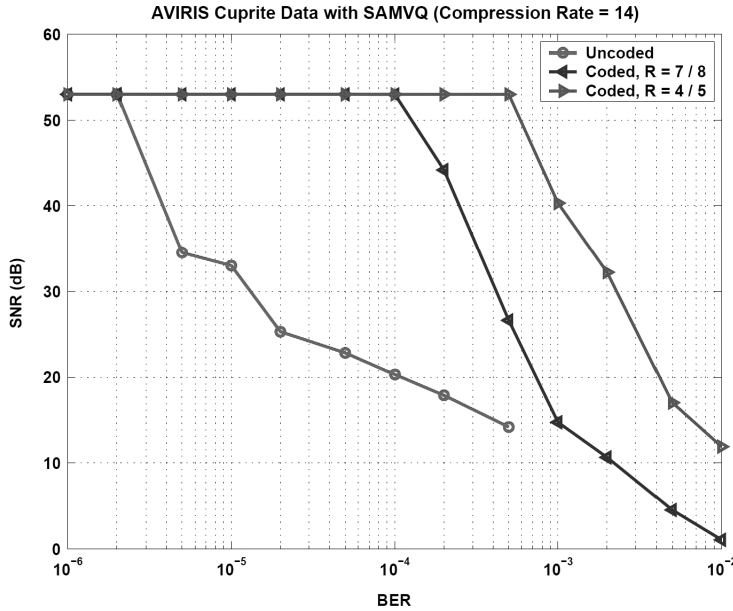


Figure 6.15 Average SNR loss of SAMVQ-compressed data with and without channel coding as a function of bit-error rate (BER) for the AVIRIS datacube.

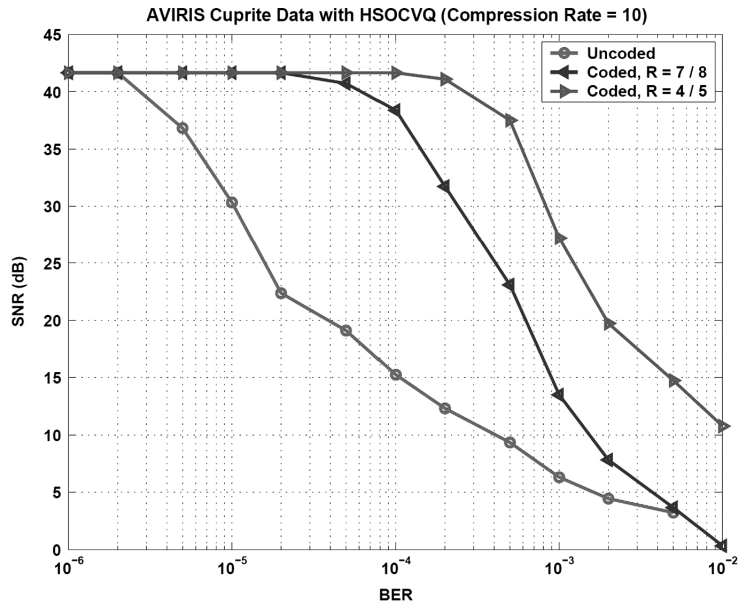


Figure 6.16 Average SNR loss of HSOCVQ-compressed data with and without channel coding as a function of bit-error rate (BER) for the AVIRIS datacube.

41.66 dB without any single-event upsets. As can be seen in the figure, although the uncoded compressed data cannot tolerate single-event upsets resulting in BERs greater than 2×10^{-6}, the coded compressed data can endure BERs as high as 2×10^{-5} or 1×10^{-4} with overheads of 12.5% or 20%, respectively. Note that at BER $= 2 \times 10^{-5}$, the SNR fidelity loss for the uncoded case is 18.49 dB. Also noteworthy is the fact that at BER $= 5 \times 10^{-3}$, the curve representing the uncoded case is about to intersect the curve representing the coded case for $R = 7/8$. In other words, the uncoded case outperforms the coded case at large BERs. This is not surprising, however, as the coding gain is counterbalanced at large BERs by the reduction in the amount of energy per coded bit, which is required for a fair comparison.

References

1. Qian, S.-E., M. Bergeron, J. Lévesque, P. Oswald, C. Black, and A. Hollinger, "Considerations of Data Handling System of a Spaceborne Imaging Spectrometer with Onboard Data Compression," *Canadian J. Remote Sens.* **34**(S1), 12–28 (2008).
2. Neville, R.A., N. Rowlands, R. Marois, and I. Powell, "SFSI: Canada's First Airborne SWIR Imaging Spectrometer," *Canadian J. Remote Sen.* **21**, 328–336 (1995).
3. Hollinger, A., M. Bergeron, M. Maszkiewicz, S.-E. Qian, H. Othman, K. Staenz, R. A. Neville, and D. G. Goodenough, "Recent Developments in the Hyperspectral Environment and Resource Observer (HERO) Mission," *Proc. IGARSS 2006*, 1620–1623 (2006).
4. Qian, S.-E, A. Hollinger, M. Bergeron, I. Cunningham, C. Nadeau, G. Jolly, and H. Zwick, "A Multi-disciplinary User Acceptability Study of Hyperspectral Data Compressed Using Onboard Near Lossless Vector Quantization Algorithm" *Int. J. Remote Sens.* **26**(10), 2163–2195 (2005).
5. Hu, B., S.-E. Qian, D. Haboudane, J. R. Miller, A. B. Hollinger, and N. Tremblay, "Retrieval of crop chlorophyll content and leaf area index from decompressed hyperspectral data: the effects of data compression," *J. Remote Sens. Environ.* **92**(1), 139–152 (2004).
6. Haboudance, D. et al., "Hyperspectral vegetation indices and novel algorithms for predicting green LAI of crop canopies: modeling and validation in the context of precision agriculture," *Remote Sens. Environ.* **90**(3), 337–352 (2004).
7. Neville, R. A., L. Sun, and K. Staenz, "Detection of spectral line curvature in imaging spectrometer data," *Proc. SPIE* **5093**, 144–154 (2003) [doi: 10.1117/12.487342].

8. Neville, R. A., L. Sun, and K. Staenz, "Detection of keystone in imaging spectrometer data," *Proc. SPIE* **5425**, 208–217 (2004) [doi: 10.1117/12.542806].
9. Khurshid, S., K. Staenz, L. Sun, R. A. Neville, H. P. White, A. Bannari, C. M. Champagne, and R. Hitchcock, "Preprocessing of EO-1 Hyperion Data," *Canadian J. Remote Sens.* **32**(2), 84–97 (2006).
10. Qian, S.-E., M. Bergeron, I. Cunningham, L. Gagnon, and A. Hollinger, "Near-Lossless Data Compression On-board a Hyperspectral Satellite," *IEEE Trans. Aerospace and Electron. Systems* **42**(3), 851–866 (2006).
11. Elias, P., "Coding for noisy channels," *IRE Convention Record* **4**, 37–46 (1955).
12. Viterbi, A. J., "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. Inform. Theory* **13**(2), 260–269 (1967).
13. "Recommendation for Space Data System Standards," *TM Synchronization and Channel Coding*, CCSDS 131.0-B-2, Blue Book, Issue 2, available at http://public.ccsds.org/publications/archive/131x0b2ec1.pdf, CCSDS, Washington, D.C. (Aug. 2011).

Color Plates

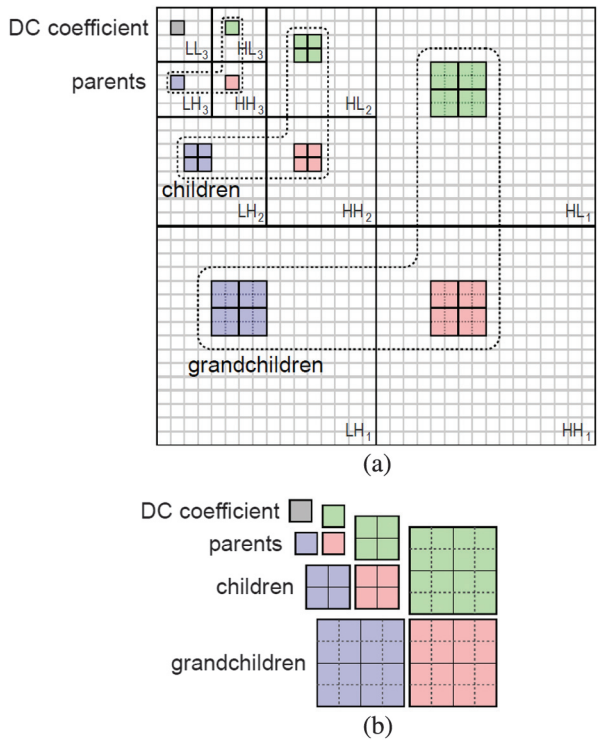


Plate 1 Three-level, 2D WT decomposition of an image and schematic of a transformed image with the 64 shaded pixels that consist of a single block: (a) the WT subband images and (b) a single block with 64 WT coefficients (source: CCSDS).

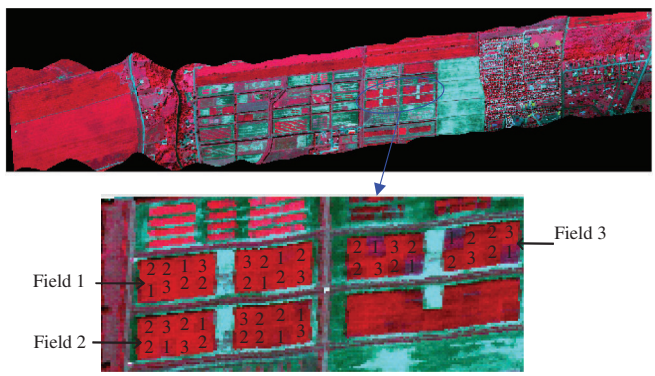


Plate 2 The scene of the CASI Acadie datacube (reprinted from Ref. 25).

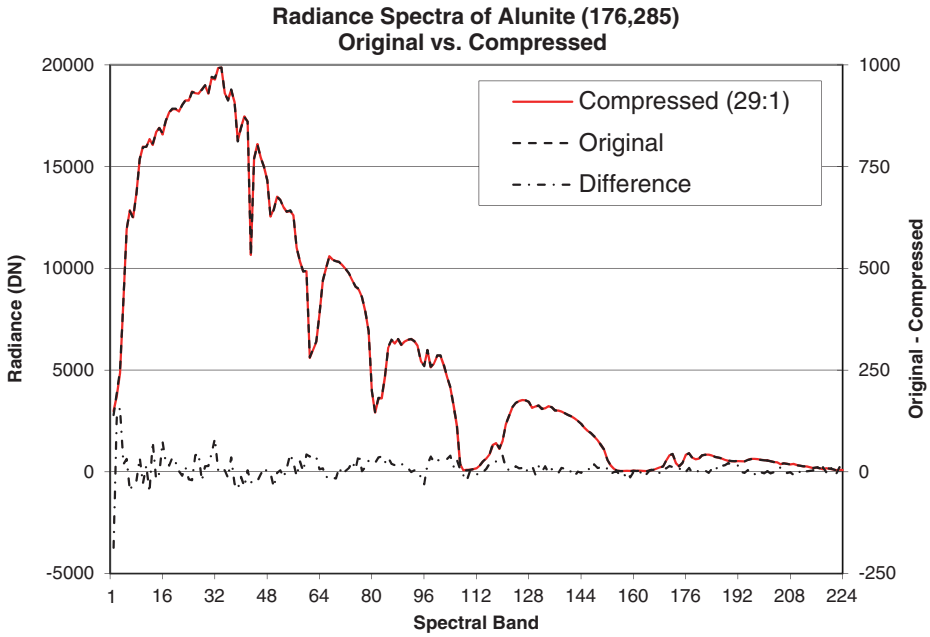


Plate 3 Spectral profile of a spatial sample of the Cuprite datacube before and after compression (29:1), and the difference. The compressed spectrum is overlapping well with the original (reprinted from Ref. 25).

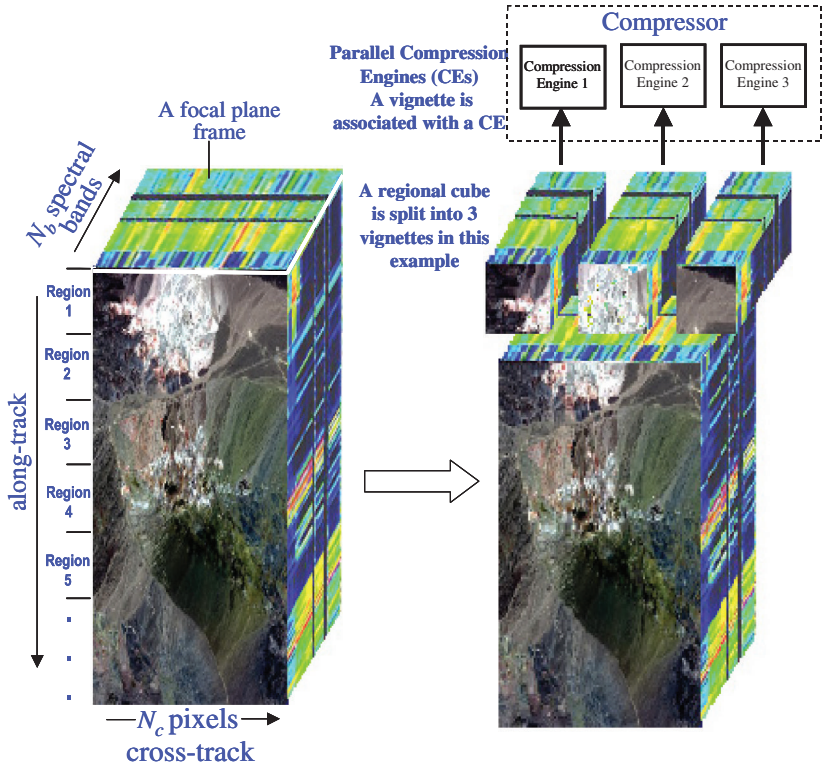


Plate 4 Illustration of (a) organizing continuous 2D focal plane frames into regional datacubes and (b) splitting a regional datacube into vignettes to facilitate parallel hardware implementation.

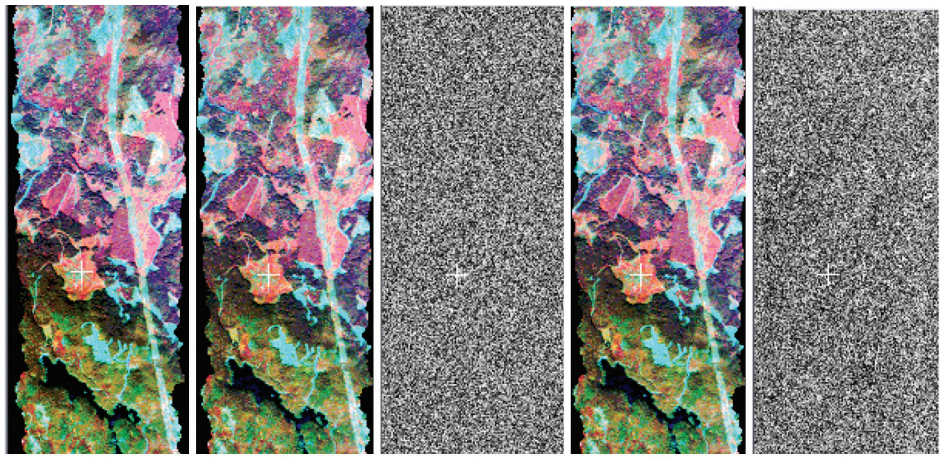


Plate 5 AVIRIS Greater Victoria Watershed District datacube: (a) noise-free datacube, (b) noise-added datacube (uncompressed), (c) intrinsic-noise datacube, (d) compressed datacube, and (e) compression-error datacube (reprinted from Ref. 1).

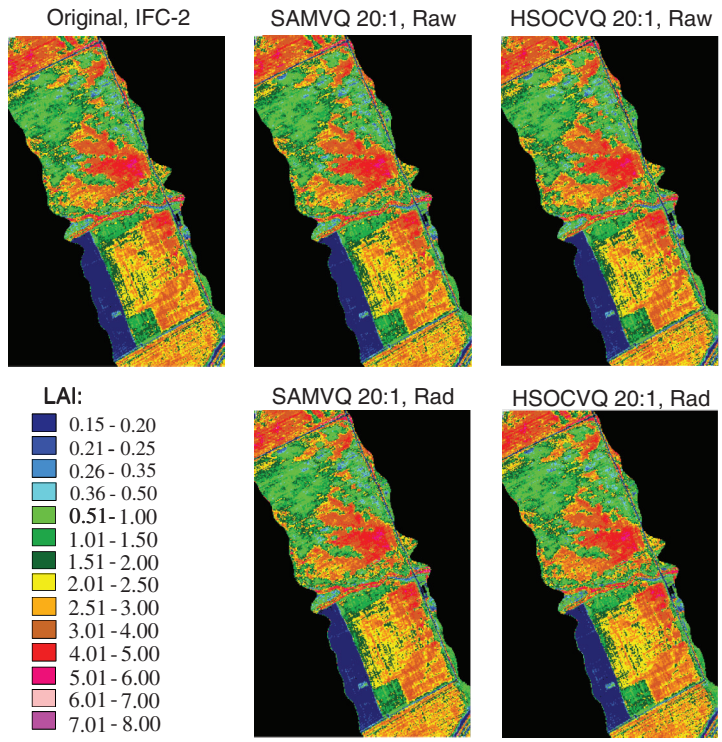


Plate 6 LAI images derived from the original IFC-2 datacube and from the compressed datacubes (20:1) with compression applied to the raw and to the radiance (Rad) datacubes (reprinted from Ref. 1).

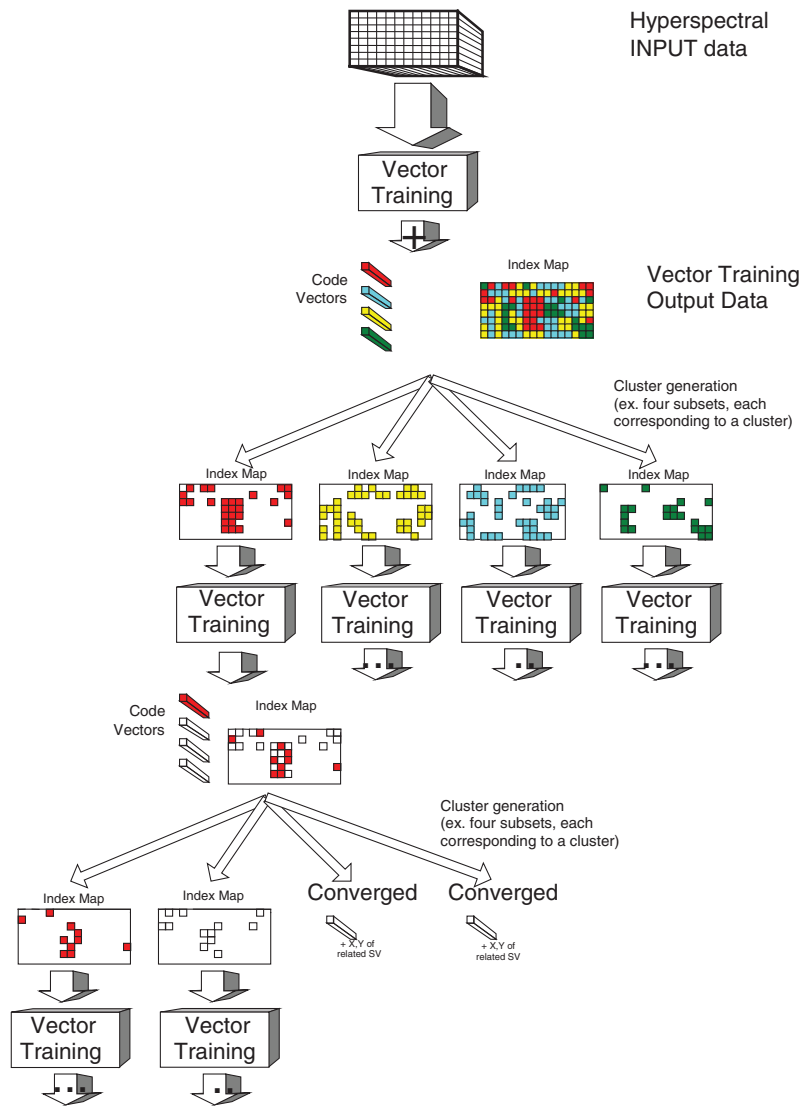


Plate 7 Overall view of the HSOCVQ process.

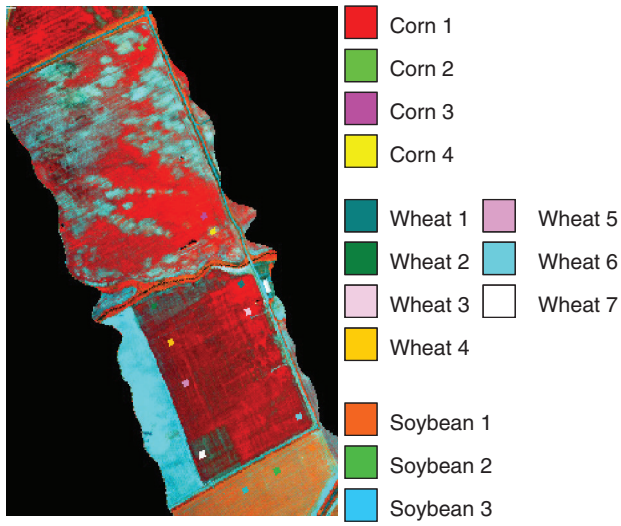


Plate 8 The CASI datacube RGB image [band #41 (708 nm) as red, band #24 (580 nm) as green, and band #8 (460 nm) as blue] with 14 sites where the ground truth was collected. The three bands were used to estimate the crop LAI (reprinted from Ref. 36).

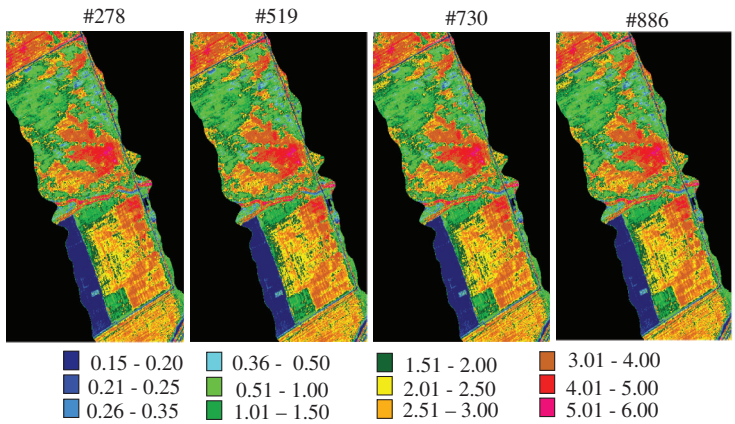


Plate 9 The LAI images derived from the four blind datacubes. The compression ratios associated with blind datacubes are as follows: 10:1 (#519), original (#730), 20:1 (#278), and 30:1 (#886) (reprinted from Ref. 36).

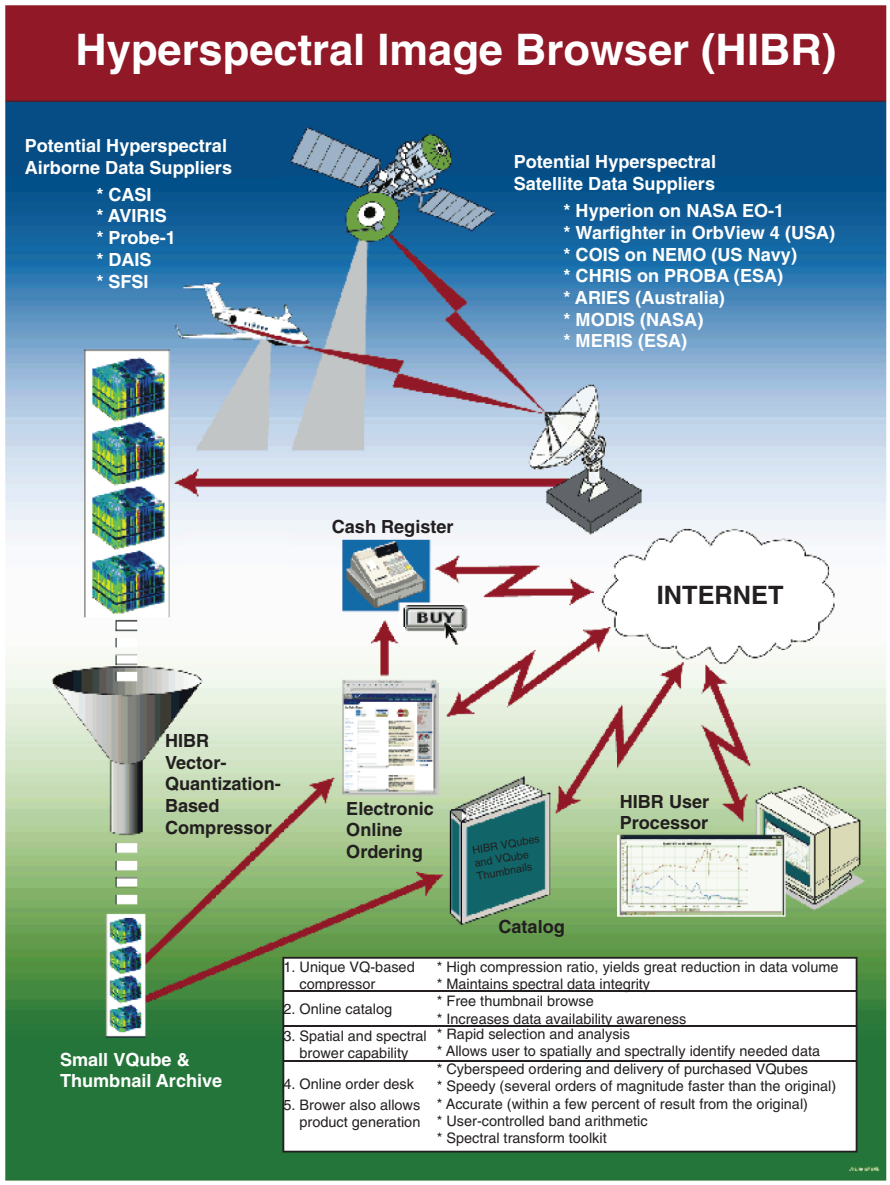


Plate 10 HIBR allows hyperspectral data users to remotely browse a hyperspectral data archive, helping them find datasets of potential interest and evaluate their applicability to the user's application.

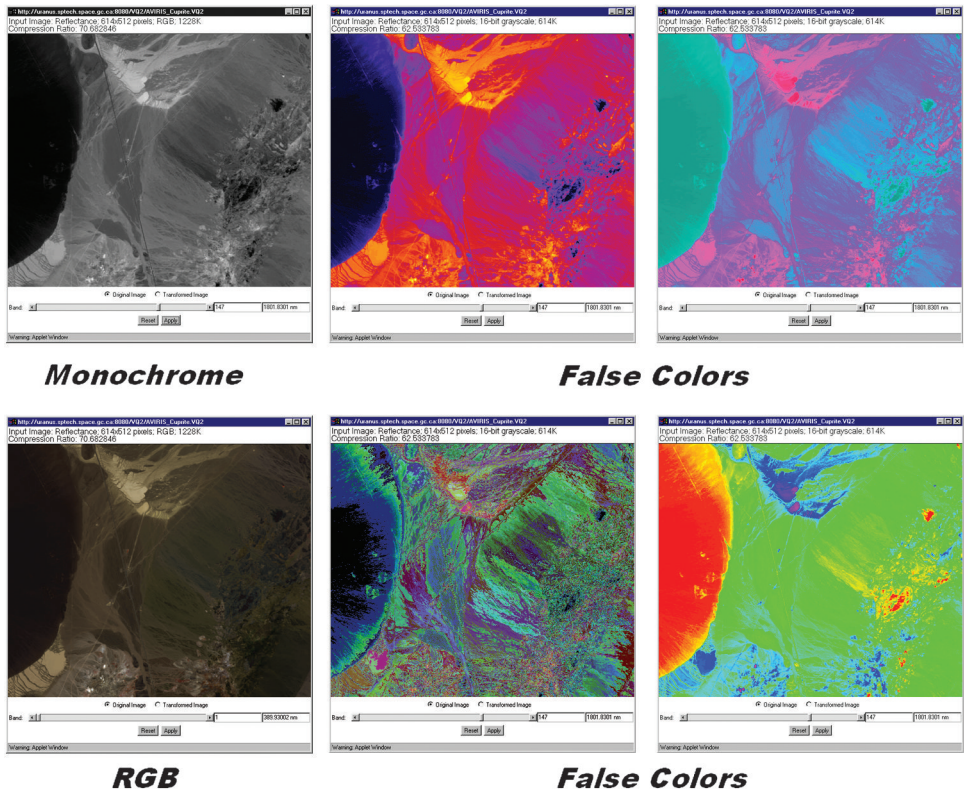


Plate 11 HIBR function for visualizing monochrome, RGB, and false color images.

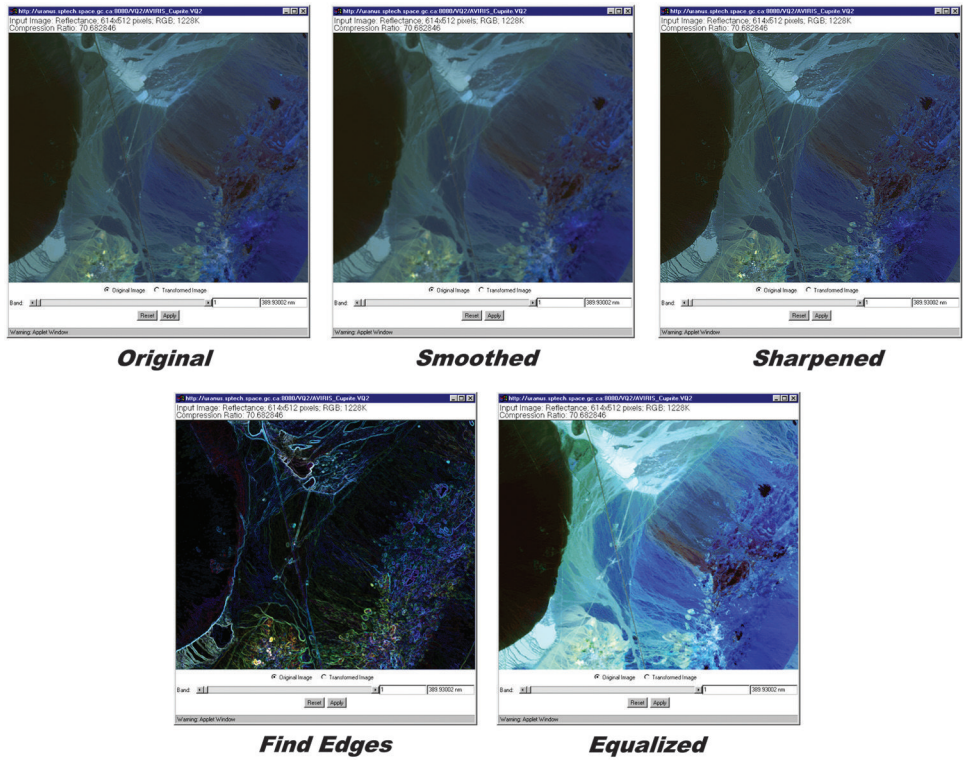


Plate 12 HIBR function for visualizing processed images.

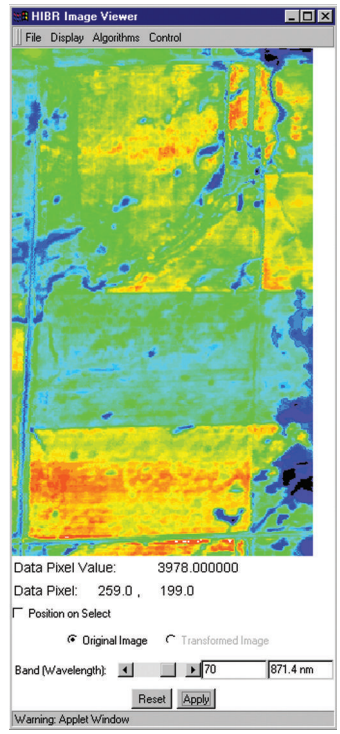


Plate 13 HIBR image viewer.

Chapter 7
Data Compression Engines aboard a Satellite

7.1 Top-Level Topology of Onboard Data Compressors

This chapter describes the hardware implementation of SAMVQ[1] and HSOCVQ[2] algorithms for near-lossless data compression onboard satellites.

Three top-level topologies were considered to meet the initial design objective, including a digital signal processor (DSP)[3] engine-based approach, a high-performance, general-purpose CPU-based approach, and an application-specific integrated circuit (ASIC)[4] or field programmable gate array (FPGA) approach.[5] The resulting onboard data compression engines were evaluated for various configurations. After studying the topologies, the hardware and software architectural options, and candidate components, an architectural preference was placed on a hardware compressor with modularity and scalability. The performance trade-off studies for these architectures showed that the best performance and scalability could be achieved using dedicated compression engines (CEs) based on an ASIC/FPGA topology. The advantages of the ASIC/FPGA approach include the ability to

- Apply parallel processing to increase throughput,
- Provide for successive upgrades of compression algorithms and electronic components over a long term,
- Support high-speed direct memory access (DMA) transfers for data read and write operations,
- Optimize the scale of the design to mission requirements, and
- Provide data integrity features throughout the data handling process.

Performance simulation was carried out for the candidate architectures by coding the FPGA using very high-speed integrated circuit hardware description language (VHDL).[6] This also verified that the proposed architectures support expansion to arrays of CEs. The design of a real-time data compressor, using VHDL tools, benefited from generic functions that

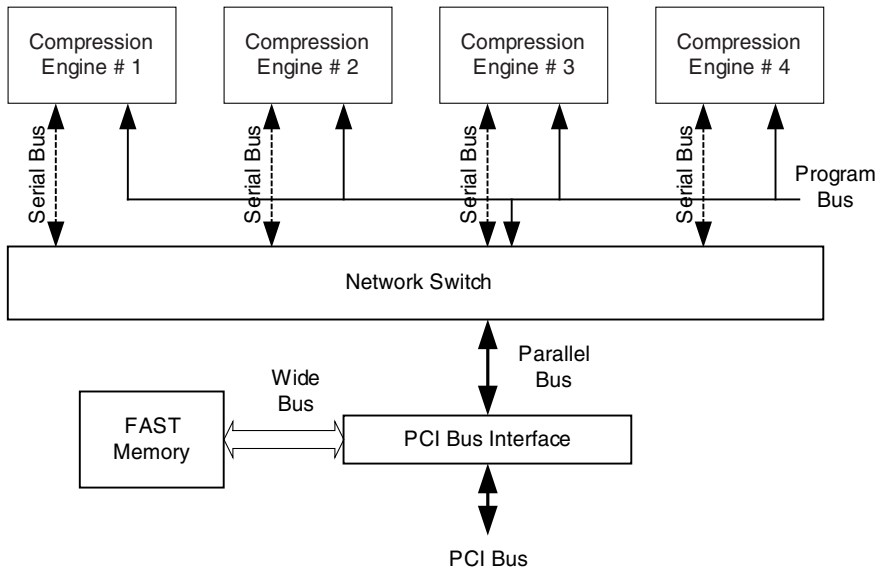


Figure 7.1 Block diagram of the real-time hardware data compressor using the ASIC/FPGA approach.

provide for rapid redesign or resizing. With these infrastructure tools, the CEs can adapt to the scale of different data requirements of a hyperspectral mission. Figure 7.1 shows a block diagram of a real-time compressor. A proof-of-concept prototype compressor has been built based on this block diagram. Figure 7.2 shows the prototype compression engine board; it consists of multiple standalone CEs, each with the ability to compress a subset of spectral vectors in parallel. These are autonomous devices that, once programmed, perform compression in continuous mode, subset by subset. A CE is composed of a FPGA chip. The prototype board also has a network switch, fast memory, and a PCI bus interface. The network switch, which consists of a FPGA chip, is used to serve the data flow transfer in and out of each of the CEs, serving up to eight CEs in parallel using a high-speed serial link.

In the prototype compressor, the fast memory is temporally treated as the continuous data-flow source of focal plane frames from a hyperspectral sensor. The imagery data is fed into CEs via the wide bus and the PCI bus of the controller computer and distributed to each CE by the network switch. The transfer datarate from the fast memory to the network switch may be lower than the real datarate of the focal plane frames produced by a hyperspectral sensor, but the throughput of the compressor from the point where data reaches the network switch to the point of output of the compressor must be greater than or equal to the real datarate. In the real case,

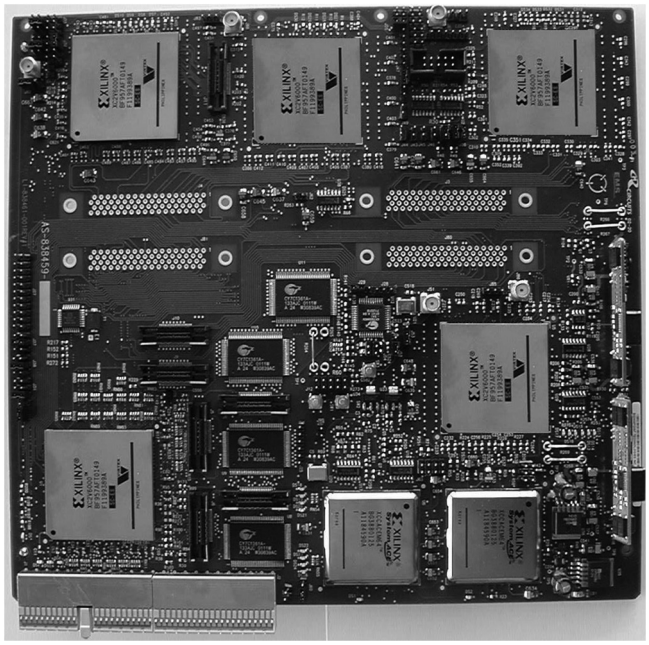


Figure 7.2 The proof-of-concept prototype compressor board (featuring four compression engines and one network switch, each of which uses a FPGA chip).

the fast memory will be replaced by the data buffer after the A/D or preprocessing of a hyperspectral sensor.

The LBG algorithm[7] is a widely used vector-training algorithm in VQ-based data compression techniques. It is utilized in both SAMVQ and HSOCVQ. In the development of a real-time onboard data compressor using SAMVQ or HSOCVQ, the implementation of the LBG algorithm has a significant effect on the performance of the compression system. In the LBG codevector training, the calculation of vector distance between a spectral vector and a codevector is the most-frequent operation. The architecture of computing the vector distance dominates the performance of a compression engine. Two novel and effective vector distance calculators have been developed,[8] referred to as the "along-spectral-bands vector distance calculator" and the "across-spectral-bands vector distance calculator." Based on these two vector distance calculators, two codevector trainers have been developed. Compression engines with four different configurations have also been built.[9]

This chapter describes the hardware implementation of the onboard compressor starting from the elementary level through to the more-complex level, starting with vector distance calculators, then codevector trainers, followed by compression engines, and finally the onboard compressor (compression system).

7.2 Vector Distance Calculators

Vector distance calculation is the most-frequent operation of the codevector training process in a VQ-based data compression technique. Both SAMVQ and HSOCVQ use either absolute distance or squared distance as the vector distance measure. In software implementation, this distance measure is usually designed as a subroutine and is called frequently. Hardware implementation of this vector distance calculation has not yet been reported. This section describes two architectures of the along-spectral-bands and across-spectral-bands vector distance calculators. Their hardware embodiments are called the along-spectral-bands vector distance calculator and the across-spectral-bands vector distance calculator.

7.2.1 Along-spectral-bands vector distance calculator

In the along-spectral-bands vector distance calculator, the distance between a spectral vector and a codevector is obtained immediately (within a few clock cycles) upon the spectral vector and the codevector entering the calculator, as all elements (data bits of each spectral band value) along the spectral band direction of the two vectors are all presented simultaneously and thus can be computed in parallel, as shown in Fig. 7.3.

The figure shows a simplified block diagram of an along-spectral-bands vector distance calculator. A spectral vector (in this example comprising $N_b = 200$ spectral bands) and a codevector are provided to the calculator from a spectral vector memory and a codevector memory, respectively. Each band of the spectral vector and its corresponding component of the codevector are provided to a respective sum of absolute/squared distance (SOAD/SOSD) unit of the calculator. Each SOAD/SOSD unit calculates a scalar distance (absolute or squared distance) between a spectral band of the spectral vector and its corresponding component of the codevector. The scalar distances are then provided to a scalar distance unit comprising cascaded registered adders. To calculate the scalar distance between a spectral vector having spectral bands and a corresponding codevector having components, the calculator comprises eight levels of registered adders starting with a hundred 13-bit adders in the first level, followed by fifty 14-bit adders in the second level, etc., as shown in Fig. 7.3. The last level consists of one 20-bit adder providing the final scalar distance. The scalar distance between the spectral vector and the codevector is obtained within a few clock cycles (eight clock cycles for 200 bands) upon entering the calculator, as all spectral bands of the spectral vector and all corresponding components of the codevector are processed in parallel.

The main features of this architecture are:

1. A best-match codevector is determined as soon as all of the scalar distances between the spectral vector and each of the codevectors have been calculated and compared. The codevector to which the spectral

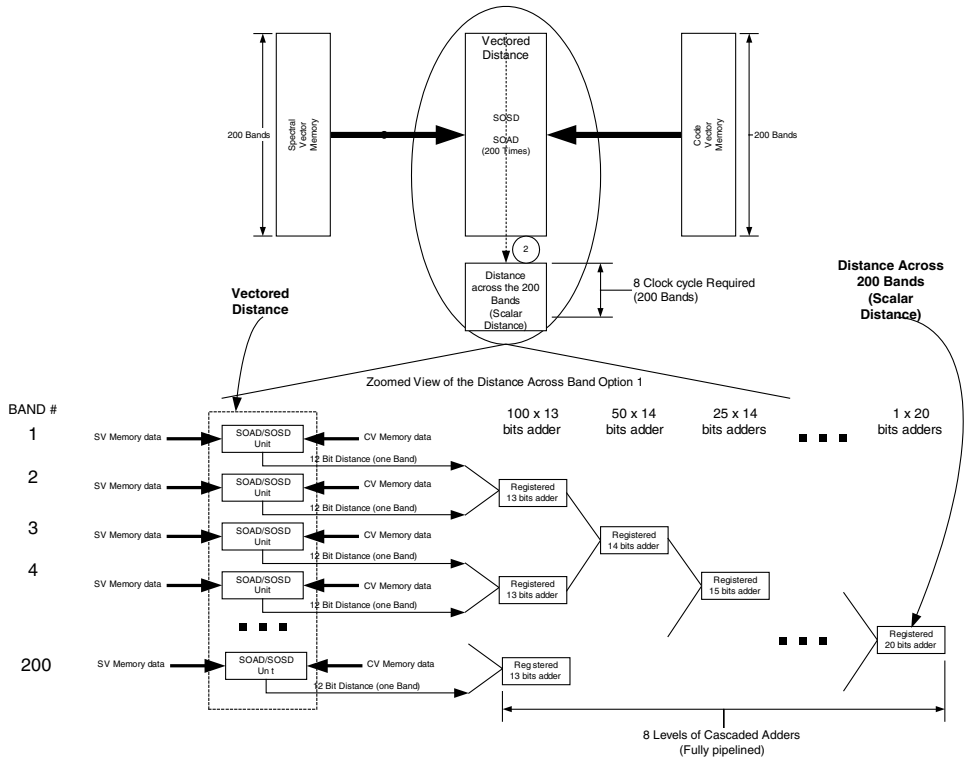


Figure 7.3 Along-spectral-bands vector distance calculator.

vector has a minimum scalar distance is the best-match codevector for the spectral vector. The partition (referred to herein as a set of spectral vectors with the same best-match codevector) to which the spectral vector belongs is known as soon as the best-match codevector is determined. The codevector is the centroid of the partition described in Chapter 4.

2. The index (i.e., address) of the best-match codevector in the codebook is assigned to the spectral vector immediately. The index is the compression output of the spectral vector when the codevector training operation is the last iteration.
3. Furthermore, the along-spectral-bands vector distance calculator supports pipeline processing operations to calculate a vector distance per clock cycle, when desired, with a latency as indicated previously. When pipeline processing, it is advantageous to provide a shift register for cascading an index associated with the codevector such that the output scalar distance and a corresponding codevector index are provided at output ports simultaneously. Thus, to determine a best match, each scalar distance is compared to a previously calculated lowest scalar distance; when the

scalar distance is lower, it replaces the previously calculated lower scalar distance, and an associated index value is stored. Once all codevectors have been provided to the along-spectral-bands, vector distance calculator, the index stored is that of the best-match codevector.

4. Because the partition is known, the spectral vector can be immediately added to the vector accumulator associated with the partition to allow the codevectors to be updated at the end of the current iteration for the next iteration of the codevector training process. (This is rendered possible because the temporary storage for the spectral vector is used for a short period of time before the best-match codevector is found). This feature greatly reduces the effort required for the codevector update operation, which would otherwise be as time-consuming as the codevector training process. The codevector update function is substantially completed when all of the spectral vectors have undergone the codevector matching process. The only operation that remains is normalization, which divides the vector in each vector accumulator by the number of members in the partition (entry counts) to get the updated codevector for that partition; this operation is performed in a few clock cycles.

7.2.2 Across-spectral-bands vector distance calculator

In the across-spectral-bands vector distance calculator, a 2D matrix architecture is proposed to implement the vector distance calculation, as shown in Fig. 7.4. One dimension (vertical in the graphics) is for the spectral vectors, and another dimension (horizontal in the graphics) is for the codevectors. The nodes of the matrix are the parallel operators of SOAD/SOSD, which computes the distance between a spectral vector and a codevector. Each SOAD/SOSD operator computes the distance between the two vectors for a spectral band during a given clock cycle.

In Fig. 7.4, a simplified block diagram of an across-spectral-bands vector distance calculator is shown. Each SOAD/SOSD unit comprises a magnitude comparator, a differentiator, an adder, and a register. In operation, a given spectral band of spectral vectors 0 to $i - 1$ are provided to the SOAD/SOSD unit such that the spectral band of one spectral vector is provided to one row of the matrix. The components of codevectors 0 to $j - 1$ corresponding to the given spectral band are provided to the SOAD/SOSD unit such that the corresponding component of one codevector is provided to one column of the matrix. Each SOAD/SOSD unit determines a distance between the given spectral band of a spectral vector and the corresponding component of a codevector during a given clock cycle. The overall scalar distance (absolute/squared distance) between the two vectors is then obtained by repeating the process N_b times (i.e., along the spectral bands).

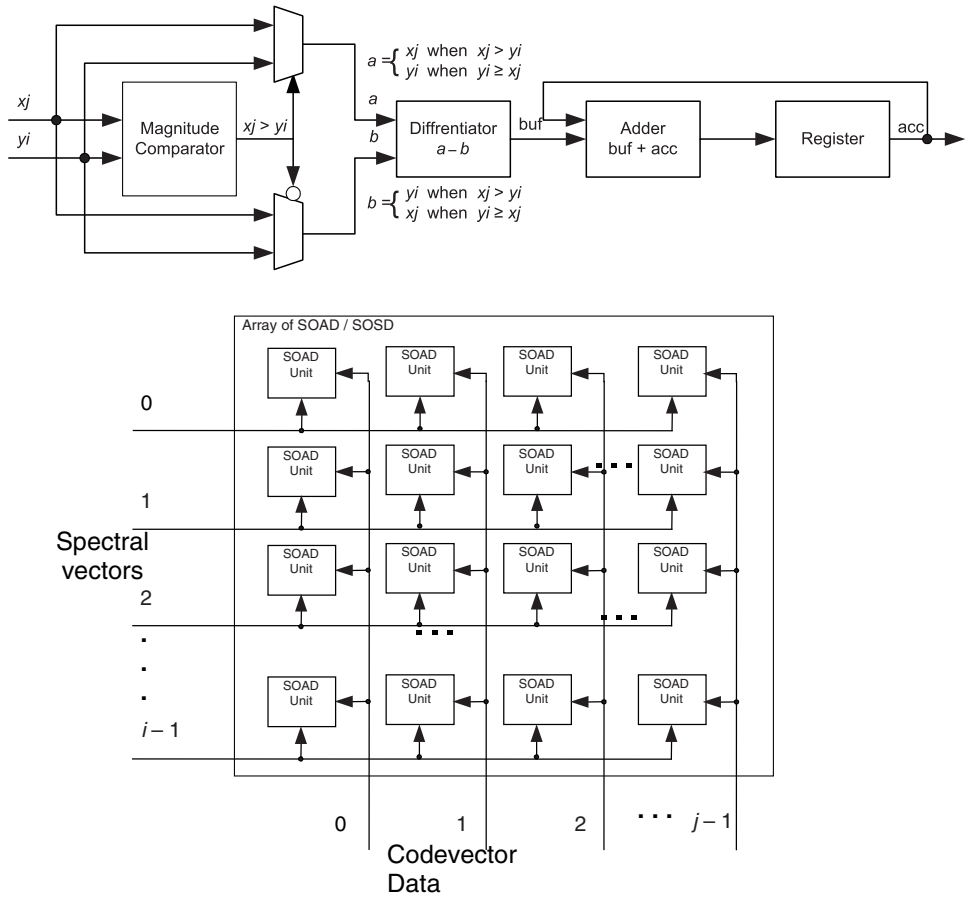


Figure 7.4 Across-spectral-bands vector distance calculator. The top image represents the block diagram of a SOAD unit.

The main features of this architecture are as follows:

1. The across-spectral-bands vector distance calculator provides for parallel processing for calculating vector distances. As shown in Fig. 7.4, a total of $i \times j$ vector component distances between i spectral vectors and j codevectors are determined per clock cycle. Each of the i spectral vectors has its best-match codevector determined and is assigned an index after all elements of the spectral vectors in the along-spectral-bands direction are accumulated.
2. This architecture uses less memory and requires a smaller bus width than the vector calculator described in Section 7.2.1. Spectral vector data corresponding to only one spectral band is fed to the SOAD/SOSD unit at a given time. There is no need to save values relating to the spectral vectors after the vector distance calculation unless the scalar distance is the lowest.

3. Only one adder and one register is used to determine the scalar distance between two vectors (instead of 200 adders and registers in the along-spectral-bands vector distance calculator) because a vector distance is determined on a spectral-band-by-spectral-band basis. The vector distance is obtained by accumulating the component distances associated with the spectral bands.
4. The across-spectral-bands vector distance calculator is highly advantageous for hardware implementation because it substantially reduces hardware size and complexity. It ultimately increases the number of spectral vectors and codevectors being processed using a given FPGA/ASCI chipset within a compression engine.

7.3 Codevector Trainers

There are two types of codevector trainers: along-spectral-bands and across-spectral-bands. Each codevector trainer is associated with a specific vector distance calculator.

7.3.1 Along-spectral-bands codevector trainer

Figure 7.5 shows a simplified block diagram of an along-spectral-bands codevector trainer. The trainer autonomously trains codevectors from the spectral vectors using the LBG iteration. The basic elements of the codevector trainer comprise an along-spectral-bands vector distance calculator, as shown in Fig. 7.3, vector accumulators for codebook update, spectral vector memory, and codevector memory. The along-spectral-bands vector distance calculator is the core of the trainer.

Due to the unique features of the along-spectral-bands vector distance calculator (see Section 7.2.1), the vector distance between an input spectral vector and a codevector is determined quickly. For each spectral vector in the spectral vector memory, the best-match codevector in the codebook is found after computing and comparing its distances to each of the N codevectors (typically $N = 8$). Because the best-match codevector of an input vector is known, the codevector update operation, which is time consuming, can be conducted at the same time as the training process; this saves a significant amount of time for codevector update operation. Figure 7.5 shows the codevector update units in the lower portion.

All hardware units of the along-spectral-bands codevector trainer are accommodated within a single integrated circuit, such as a FPGA. There is no need for external communication once all the data is fed to the trainer at the beginning of the training process. This self-contained implementation of the training process significantly reduces the time required for communication and data transfer.

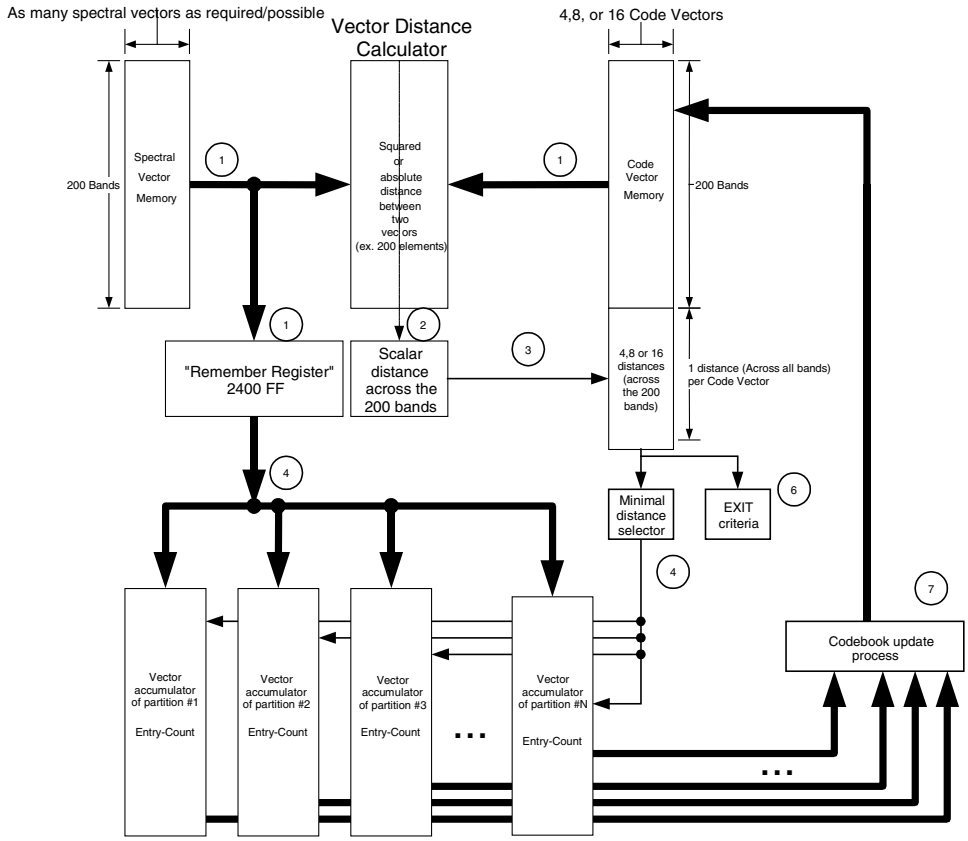


Figure 7.5 Along-spectral-bands codevector trainer.

In Fig. 7.5, the numbers in the circles show the time order of the operations of the codevector training process. The spectral vector memory contains the input data in the form of spectral vectors, with each spectral vector comprising, for example, $N_b = 200$ spectral bands. A spectral vector is read completely during a single clock cycle, i.e., when a given spectral vector of the spectral vector memory is read, 200 spectral band values of 12 bits, each resulting in 2400 bits, are obtained.

The codevector memory contains all N codevectors used in the compression, for example, $N = 4$, 8, or 16. Again, a complete codevector (comprising, for example, 200 components of 12 bits each) is read during a single clock cycle.

The input data corresponding to one spectral vector and one codevector are fed to the along-spectral-bands vector distance calculator to determine a vectored form of the squared distance or absolute distance between the two vectors, i.e., a vector distance comprising 200 scalar distances between the 200 spectral bands of the spectral vector and 200 corresponding components of the

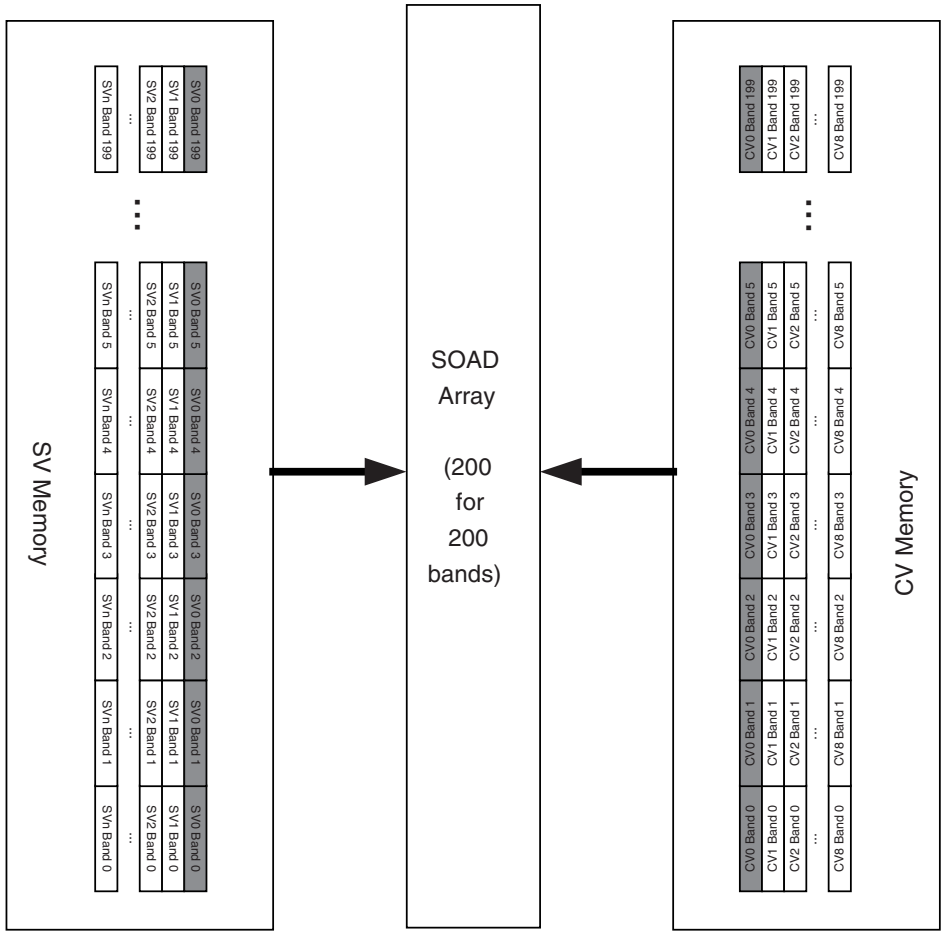


Figure 7.6 Simplified block diagram illustrating memory arrangement of the along-spectral-bands codevector trainer.

codevector. The vectored form of the squared distance or absolute distance is then provided to a scalar distance unit.

In Fig. 7.6, a memory architecture accommodating the data provision to the along-spectral-bands vector distance calculator is shown. As shown in the figure, all spectral bands of a spectral vector (SV) and all components of a codevector (CV) are readily available at output ports of the SV memory and the CV memory, respectively. Due to this specific architecture, no other SV or CV is accessible while a given SV or CV is being accessed. Each memory output is dimensioned large enough to accommodate the provision of a complete vector (SV or CV) typically comprising 200 bands of 12 bits each, i.e., each output data being 2400 bits wide.

A remember register is interposed between the SV memory, the vector distance calculator, and the vector accumulators. The remember register consists of a memory array equal to the spectral vector in size and serves as temporary storage for the spectral vector while the scalar distance between the spectral vector and all of the codevectors is being determined.

The scalar distance unit receives the vectored form of the squared or absolute distance as an input value and sums, for example, its 200 vector components, in order to provide a scalar distance between a given spectral vector and a codevector. The distance sorting unit (the box attached to the bottom of the codevector memory box in Fig. 7.5) is used to store the scalar distances between a given spectral vector and all N codevectors successively provided by the scalar distance unit. The N sorted scalar distances are then sent to a minimal distance selector (MDS) unit to determine the closest codevector to the given spectral vector. The codevector that corresponds to the smallest scalar distance is the best-match codevector.

Based on the determined codevector, the MDS unit determines the associated vector accumulator (VA) and accumulates the given spectral vector to the associated VA. The $N = 4$, 8, or 16 VAs comprise specialized adders for summing all of the spectral vectors associated with a given codevector. Each VA has as many adders as the number of spectral bands in a spectral vector. This process repeats until all of the spectral vectors in the SV memory are trained.

The exit criteria unit receives the overall minimal scalar distances determined for all spectral vectors of the input datacube and determines the end of the codevector training process based on predetermined criteria. When the overall minimal scalar distances are smaller than the predetermined criteria, the exit criteria unit ceases the training iteration; otherwise, it moves to the next iteration. However, before moving to the next iteration, the MDS unit finalizes the codebook update process. The codevectors in the VAs are almost the updated codevectors; they just need to be normalized by dividing them by the number of the counted spectral vectors in each of the partitions and then sent to the CV memory for the next iteration step.

The architecture of the along-spectral-bands vector trainer permits the codevector update operation to be carried out at the same time as the codevector matching process. The time-consuming codevector update process has been substantially completed when all of the spectral vectors in the SV memory have been processed. This unique feature of the along-spectral-bands vector trainer saves a significant amount of time for codevector updating.

7.3.2 Across-spectral-bands codevector trainer

Figure 7.7 shows a simplified block diagram of an across-spectral-bands codevector trainer. It autonomously trains codevectors from the spectral vectors using the LBG iteration. It comprises the along-spectral-bands

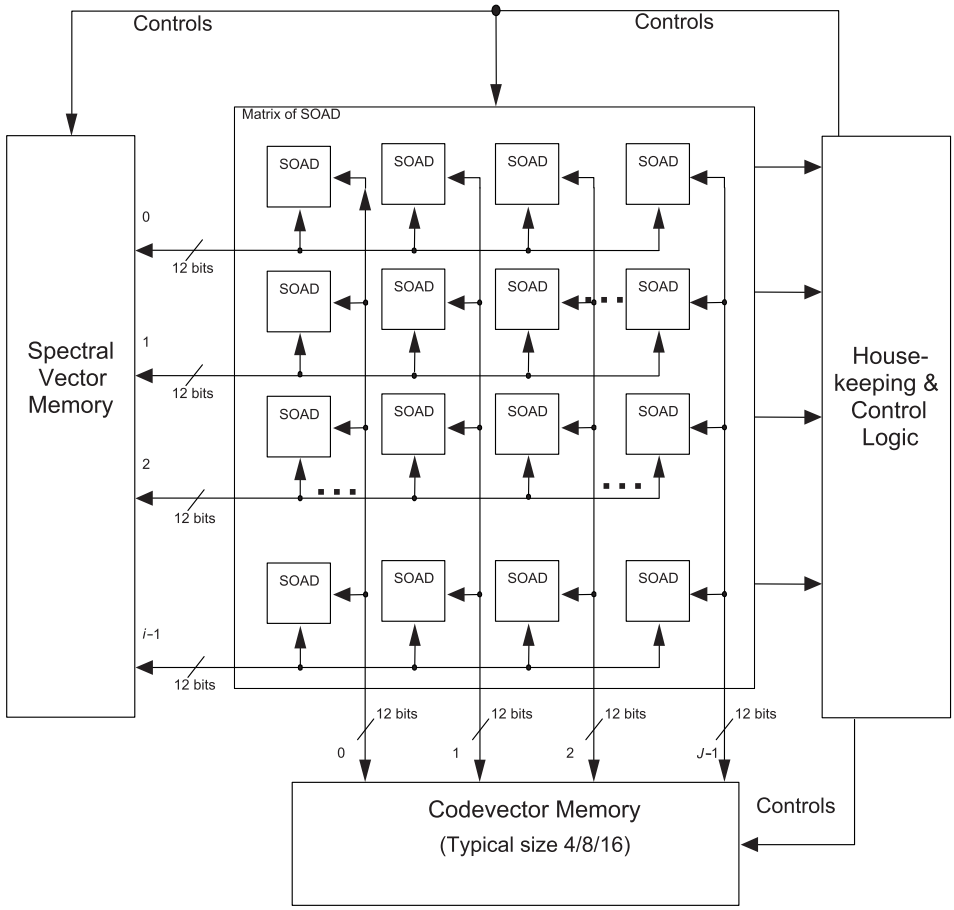


Figure 7.7 Across-spectral-bands codevector trainer.

vector distance calculator, the spectral vector memory, the codevector memory, and the housekeeping control logic. All hardware units are accommodated within a single integrated circuit, such as a FPGA. External communication is not required once all of the data is fed to the trainer at the beginning of the process. The across-spectral-bands vector distance calculator is the core of the trainer.

The spectral vector memory contains the input data in the form of spectral vectors, with each spectral vector comprising, for example, 200 spectral bands. Due to the matrix architecture of the across-spectral-bands vector distance calculator, the codevector trainer calculates multiple vector distances simultaneously on a spectral-band-by-spectral-band basis. The number of codevectors to be trained is usually in a range of 4–16 codevectors, which are accommodated in the columns of the across-spectral-bands vector distance calculator array, as shown in Fig. 7.7. The number of spectral vectors residing

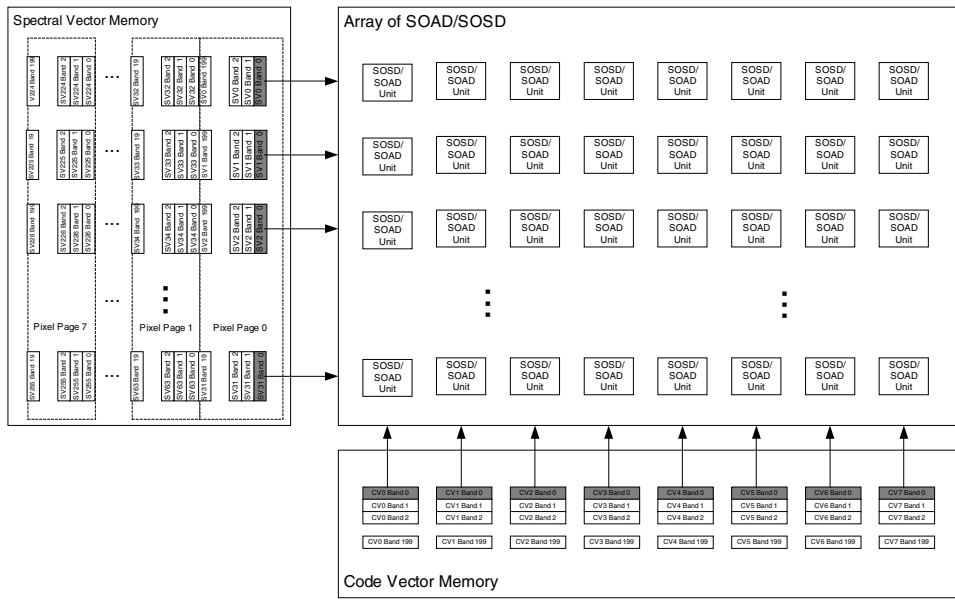


Figure 7.8 Across-bands memory arrangement.

in the SV memory is usually larger than the number of rows of the across-spectral-bands vector distance calculator. Therefore, the total number of spectral vectors stored in the SV memory is grouped into pages of *i*-size spectral vectors. The pages are then processed successively in the across-spectral-bands vector trainer.

As shown in Fig. 7.8, the SV memory provides a spectral band corresponding to the same dimension within the spectral vectors for the *i* spectral vectors, and the CV memory provides the corresponding component of the *j* codevectors. Therefore, the SV memory has a memory output width equal to the number of spectral vectors being read simultaneously multiplied by the sample width, i.e., 384 bits for 32 rows (spectral vectors), and 12 bits per band. Accordingly, the CV memory has a memory output width equal to the number of codevectors multiplied by the sample width, i.e., 48, 96, or 192 bits for 4, 8, or 16 codevectors, respectively.

With each clock cycle $i \times j$, vector distances for a given spectral band are determined and added to the sum of previously determined vector component distances. This process is repeated along spectral bands until the last spectral band of the spectral vectors is processed and $i \times j$ final scalar vector distances for the combination of *i* spectral vectors and *j* codevectors are obtained. With the $i \times j$ scalar vector distances determined, a best-match codevector is determined for each spectral vector of the current page. Using a control unit, indices of the best-match codevectors are then assigned to the spectral vectors. The above process is then repeated for all pages.

In an across-spectral-bands codevector trainer, the codevector update is carried out after the codevector matching process. After all of the spectral vectors have been processed, the SOAD/SOSD matrix of the vector distance calculator is used to accumulate the spectral vectors for each partition. Each partition corresponds to a codevector. The indices assigned to the spectral vectors are used in order to identify the partition to which a spectral vector belongs. Thus, spectral vectors of a given page with the same index are accumulated on a spectral-band-by-spectral-band basis in a same column of the matrix. The accumulation of the spectral vectors is repeated until all of the pages are processed.

After summing all of the vectors of each partition, the remaining operation of the codevector update is to normalize the intermediate vector of each update partition by dividing it by the number of members in each of the partitions to obtain the updated codevector for the associated partition for the next iteration step. The assignment of a spectral vector to a respective partition, the entry count, and the normalization are performed using the control unit. Furthermore, the end of the codevector training process is determined based on predetermined exit criteria using the control unit.

7.4 Vector Quantization Data Compression Engines

A compression engine (CE) is a standalone autonomous machine that compresses a subset of spectral vectors using either SAMVQ or HSOCVQ techniques. Figure 7.9 shows the block diagram of the CE. It is built in a single integrated circuit, such as an FPGA. A CE is composed of a codevector trainer, a state machine controller, an internal RAM, two direct memory access (DMA) interfaces, and a programming bus interface. Dashed lines indicate a programming bus, dotted lines indicate request or handshake lines, and thick continuous lines indicate data buses. The state machine controller may be designed such that the CE can run either SAMVQ or HSOCVQ. A DMA interface transfers raw spectral vectors to the internal RAM at high speed. Another DMA transfers compressed data out of the CE. The programming bus interface enables the programming of the codevector trainer and the state machine controller of the CE. In this way, a codevector trainer can be selectively implemented as either an along-spectral-bands or across-spectral-bands vector trainer. A CE can operate in any of the four configurations listed in Table 7.1 without any change to the hardware.

In a preferred configuration of the CE, the codevector trainer is implemented as an along-spectral-bands codevector trainer (shown in Fig. 7.5) as well as an across-spectral-bands codevector trainer (shown in Fig. 7.7). The state machine controller is designed such that the CE performs either the SAMVQ or the HSOCVQ technique. The programming bus interface

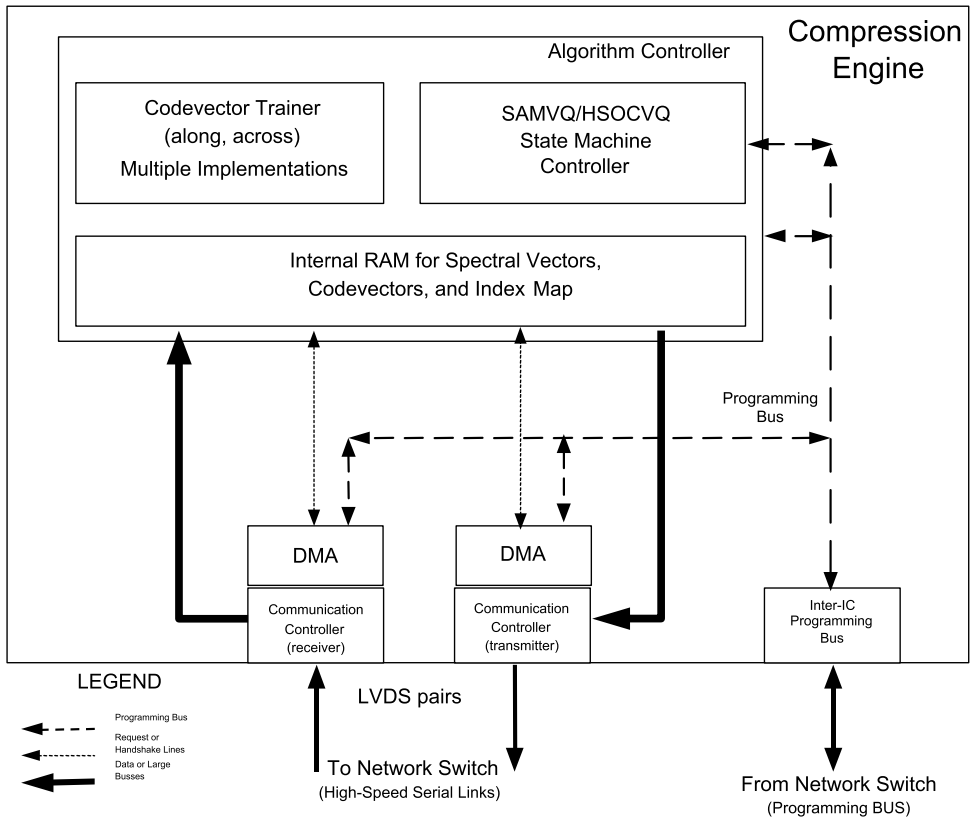


Figure 7.9 Block diagram of a compression engine.

receives a programming signal from a network switch to program the codevector trainer and the internal RAM to support along-spectral-bands codevector training or across-spectral-bands codevector training as well as programming the state machine controller to perform either the SAMVQ technique or the HSOCVQ technique.

The DMA interface receives input spectral vectors from the network switch and transfers the same spectral vectors to the internal RAM at high speed. The DMA interface transfers the compressed data to the network switch. The DMA interfaces are connected via a high-speed serial link pair

Table 7.1 Four configurations of a compression engine: two configurations for implementing SAMVQ, and two configurations for implementing HSCOVQ.

Algorithm	SAMVQ	HSOCVQ
Codevector trainer 1	Along spectral bands	Along spectral bands
Codevector trainer 2	Across spectral bands	Across spectral bands

interface (LVDS) to the network switch for fast transfer of the input spectral vectors and the compressed data. The input spectral vectors are stored in the internal RAM prior to compression. The internal RAM comprises the memory architectures shown in Figs. 7.6 and 7.8 supporting along-spectral-bands codevector training as well as across-spectral-bands codevector training. The compressed data (codevectors and index map) is also stored in the internal RAM prior to transmission via DMA interface.

This hardware design enables the CE to autonomously perform the compression process without external communication, substantially increasing processing speed. Only a few commands such as "feeding vectors" and "start compression" are needed to initiate the compression process. The completion of the compression process is signaled by a prompt. Furthermore, the CE is independent of a system's clock. Once the input data is received, the compression process is performed using an internal clock. The serial links act to decouple the clock domains of the CE and its environment. The programming bus interface is asynchronous and does not constrain the clock of the CE. These cited features are highly advantageous because they provide a CE that is fast, flexible without hardware changes, capable of autonomous operation, and easy to control. Therefore, the CE substantially facilitates formation of a parallel compression system.

7.5 Real-time Onboard Compressor

7.5.1 Configuration

A real-time compressor is composed of a plurality of parallel compression engines and a high-speed network switch, as shown in Fig. 7.10. The network switch (Fig. 7.11) distributes the input spectral vector data into each of the plurality of CEs and then receives and transmits the resulting compressed data. The cluster SAMVQ and recursive HSOCVQ techniques support splitting the raw hyperspectral imagery data into subsets and allow them to be compressed in parallel within a single CE. The real-time compressor is built on a print-circuit board (PCB).

An input buffer memory (RAM) is connected to the network switch via a width data bus (e.g., 128-bit). The input buffer memory allows for multiple subsets of data to be stored and read at very high rates in order to avoid bottlenecks in the serial link communication to the CEs. The PCB enables high-speed serial links between the network switch and the CEs. Reconfigurable integrated circuit (IC) chipsets such as FPGAs connected to the PCB are used, for example, one reconfigurable IC for each of the parallel CEs, and one reconfigurable IC for the network switch. This allows, for example, programming from a ground station of multiple configurations in real-time using the same hardware aboard a spacecraft. Furthermore, the PCB enables

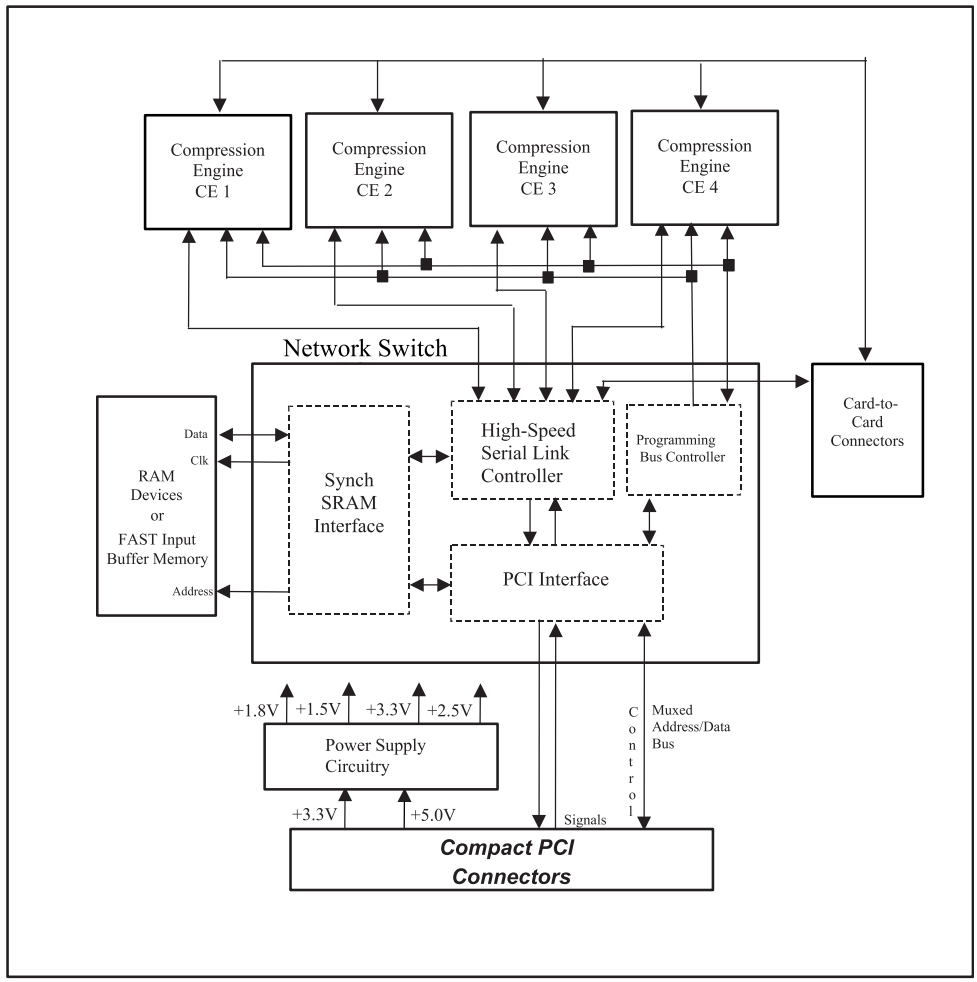


Figure 7.10 Block diagram of the real-time compressor.

a serial link expansion of the network switch to a second PCB via a communication link such as a card-to-card connector, i.e., doubling the number of CEs in the compressor. Using existing technology, four CEs are implemented for the simultaneous processing of four data subsets on one PCB, or eight CEs when two PCBs are linked. The compressor is integrated into an application system environment and linked thereto via a communication link such as the compact PCI connectors for transmission of control signals, data, and power. The received power is transmitted via power supply circuitry to the various components of the compressor. The hardware architecture shown in Fig. 7.10 and the employment of PCBs and FPGAs provides the compressor with modularity that allows adaptation of the compressor to numerous system applications with minimal redesign.

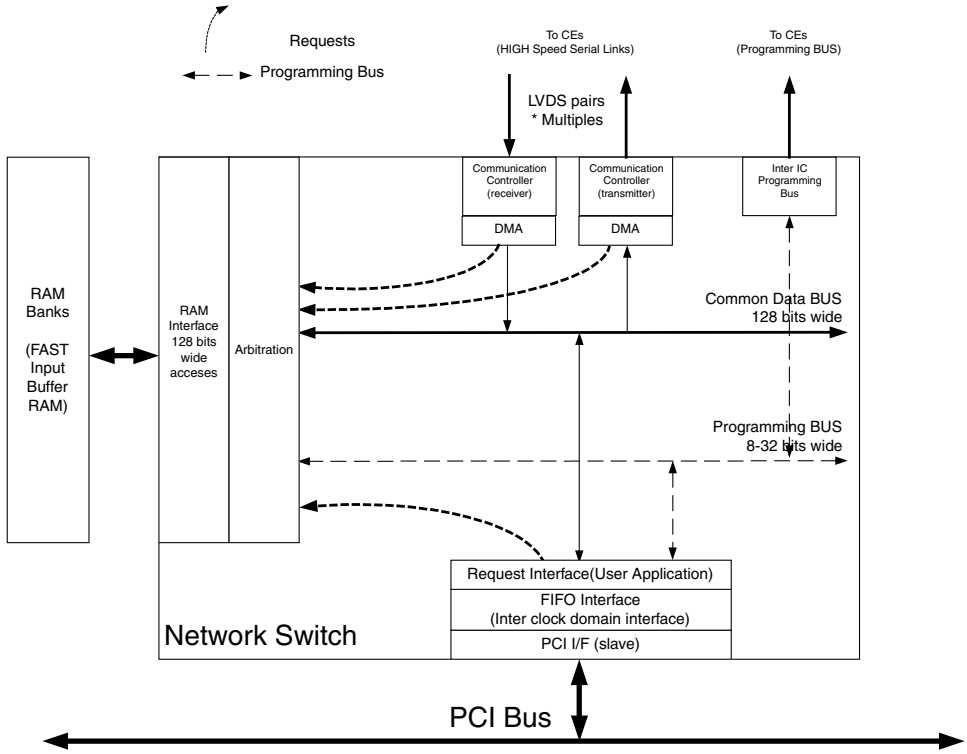


Figure 7.11 Network switch.

The real-time compressor has the following features:

1. The hardware architecture enables the compressor to have modularity and expandability so that it can serve any hyperspectral satellite system requirements with minimal redesign.
2. A reconfigurable IC chipset (such as FPGA) can be used, allowing multiple real-time configurations with the same hardware aboard spacecraft.
3. The PCB allows for high-speed serial links between the network switch and PCB internal and external CEs.
4. The serial link expansion of the network switch is available to a second PCB. A network switch can link to up to eight CEs.
5. It accommodates debugging features in the implementation.

7.5.2 Network switch

The network switch is a unique design in this system (Fig. 7.11). It resides in a single IC (such as FPGA), and its main duties are to

- Provide a PCI interface to the PCB,
- Provide an interface to access the FAST input buffer memory,

- Provide an interface to access the CEs (programming bus),
- Allow interconnection to eight CEs, and
- Allow for expansion to add other functions, such as a summing unit.

Figure 7.11 shows a more-detailed block diagram of the network switch. The switch is linked to the PCB via the PCI interface, which consists of a programming bus link for configuring the CEs via a programming bus (thin dashed lines) of the network switch and inter-IC programming interface, as well as for configuring the fast input buffer RAM via the RAM interface. The PCI interface further comprises a PCI data link for transmission of input spectral vectors data and compressed output data. Data communication within the network switch is enabled using, for example, a 128-bit-wide common data bus (solid lines) connecting the PCI interface, the RAM interface, and two DMA interfaces. The DMA interface transmitter provides a high-speed serial link to the CEs for provision of input data thereto. Accordingly, the DMA interface receiver provides a high-speed serial link to the CEs for receiving the compressed data from the CEs.

In operation, the network switch receives the input spectral vector data from the PCI interface and transmits the received data to the input buffer memory via the RAM interface for storage therein. Depending on the received programming data, the input spectral vector data is then accessed and partitioned for distribution to the plurality of CEs using the RAM interface. The partitioned input spectral vector data is then provided to the plurality of CEs via the common data bus and the DMA interface transmitter. The compressed data is received from the plurality of CEs via the DMA interface receiver and provided via the common data bus and the RAM interface to the fast input buffer RAM for storage therein. Alternatively, the received compressed data is directly transmitted to the PCI interface.

Employment of the RAM interface linked to the fast input buffer RAM is highly advantageous for real-time data compression by allowing receipt of input data during the compression process and provision of the same data to the CEs via a high-speed serial link, substantially increasing the processing speed. Furthermore, partitioning and provision of the partitioned input data for simultaneous processing by a plurality of CEs is substantially facilitated using the fast input buffer RAM. Using the technology available at the development period, the hardware implementation of the network switch is realized using one integrated circuit, such as a FPGA. It allows implementation of eight CEs within one compressor. Furthermore, it is possible to add other functions, such as a summing unit. The hardware architecture shown in Figs. 7.9 and 7.10 allows direct connection of a hyperspectral sensor to the network switch via a high-speed serial link.

Note that the network switch has been instantiated in the design as it would be used in a final flight design; the only difference being that the PCI bus is used to program the devices (CE and NS) on the board, i.e.,

- The network switch has direct access to the fast input buffer RAM,
- It provides a direct connection to eight CE using high-speed serial links, and
- It allows for focal plane frames of a satellite hyperspectral sensor to be connected directly to the network switch using a spare high-speed serial link.

7.6 Hardware Implementation Process of SAMVQ and HSOCVQ

7.6.1 Codevector training

The LBG iteration is used in the vector training process in both SAMVQ and HSOCVQ. It is an iterative, vectored clustering algorithm that groups a large number of spectral vectors (SVs) into a small set (4, 8, or 16) of classes, each of which is associated with a codevector (CV) by which all input spectral vectors are encoded with the index of their best-matched codevector in the codebook.

The data inputs and outputs for codevector trainer are

1. **The input dataset:** An array constituted of spectral vectors.
2. **The output codevectors:** A small set of vectors, which encodes the input spectral vectors with the minimum distortion measure.
3. **The output index map:** A 2D array whose size is the same as the spatial image of the input datacube. Each value in a spatial location of the index map is an index referring to the best-matched codevector by which the specific input spectral vector at this location is encoded.

The codevector training is illustrated in Fig. 7.12.

In order for the vector training process to determine the best-match codevector and then to encode each spectral vector in the input dataset, an iterative process is used. This is referred to as the vector training process.

When the vector coding process completes, two compressed datasets are produced: A small set of CVs (4, 8, or 16) and an index map (with size equal to the number of row by the number of columns).

These datasets are saved and ultimately sent to the ground via downlinking channel. It can be seen that on the ground, when each index (in the index map) is replaced by its associated codevector (referred to as the encoded datacube), a very good reconstruction of the original input datacube is obtained. SAMVQ and HSOCVQ algorithms define the next steps that will better approximate the original cube.

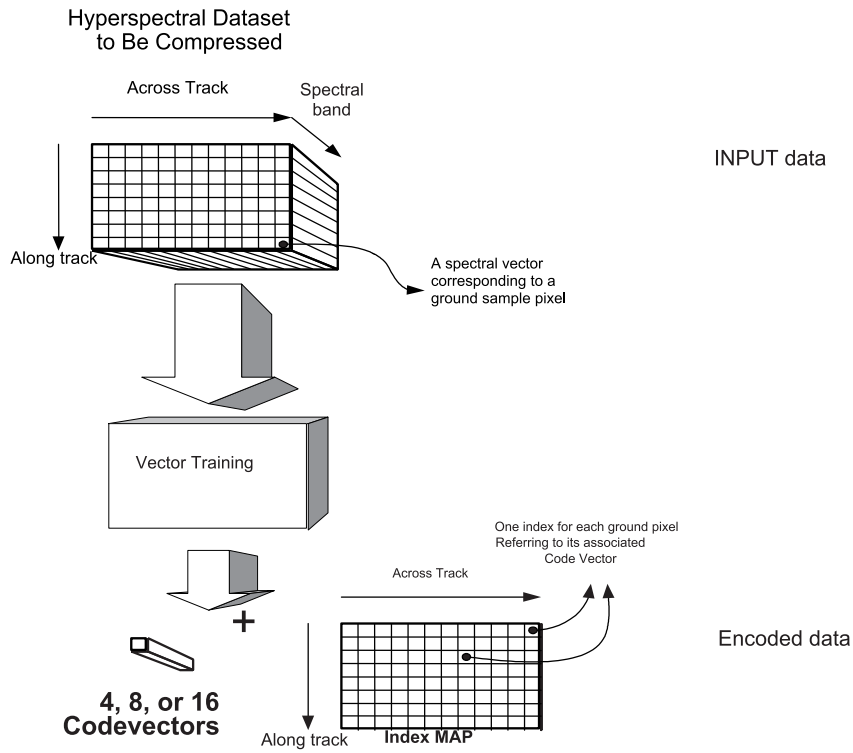


Figure 7.12 Vector training processing.

7.6.2 SAMVQ

From the vector training process discussion, if the reconstructed datacube is subtracted from the original input datacube, an error cube is obtained (represented in Fig. 7.13).

The error cube is then fed to the vector training processing, just like the hyperspectral input data shown in Fig. 7.12, and another set of CVs and index maps is generated. A complete loop (vector training processing, cube reconstruction, and residual process) is referred to as a SAMVQ stage.

The SAMVQ compression algorithm has a fixed or variable number of stages (based on a fidelity measure, such as PSNR). The compressed data at each stage includes a codebook and an index map. The index map and codebook of each stage are then sent to the ground for reconstruction. Figure 7.14 shows the overall view of SAMVQ processing.

For the first stage of the SAMVQ compression, the vector training processing uses the hyperspectral input data. When the residual process takes place at the end of the first stage approximation, the reconstructed cube is subtracted from the hyperspectral input data to generate the error cube (of stage 1), which is indicated by arrow 1. When the compression processing

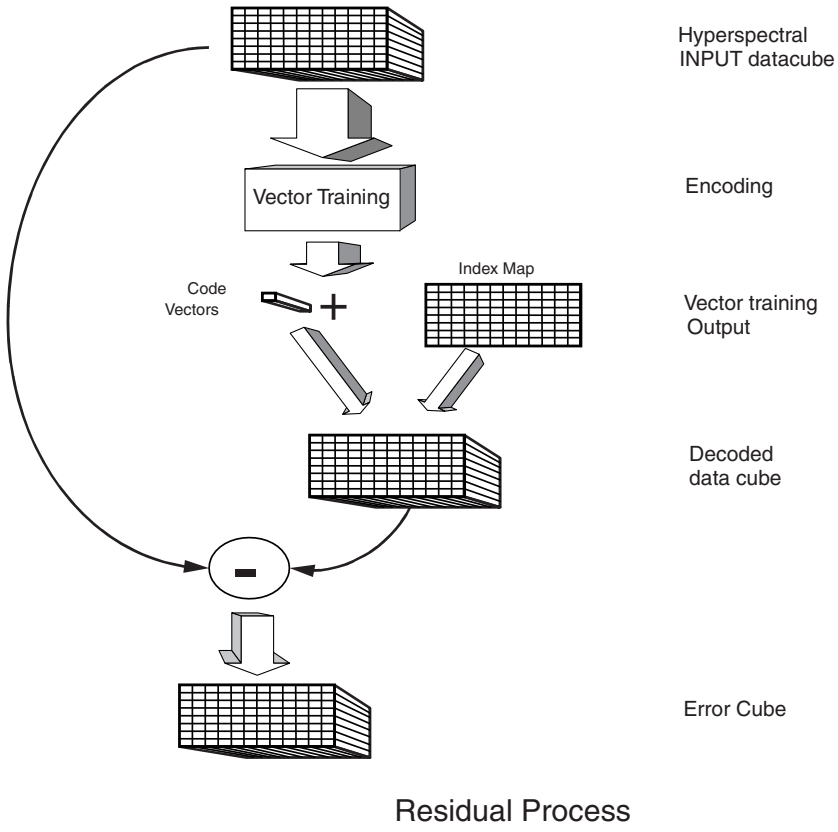


Figure 7.13 Error cube (residual cube) generation process.

takes place at the second stage, the input data is the error cube from stage 1, which is indicated by arrow 2. At the end of the second stage compression on residual, the reconstructed cube (reconstructed error cube of stage 1) is subtracted from the source error cube of the stage (indicated by arrow 3). This generates the error cube of the error cube.

The process goes on for M stages. M sets of index maps and codevectors (one set per stage) are generated and sent to the ground for reconstitution.

7.6.3 HSOCVQ

Figure 7.15 illustrates the hierarchical self-organizing clustering process of HSOCVQ. It can be seen from the figure that the hyperspectral input data is fed to the vector training processing as described previously. The output of that block is a set of codevectors (four in this example) and an index map specifying which codevector is associated with each pixel.

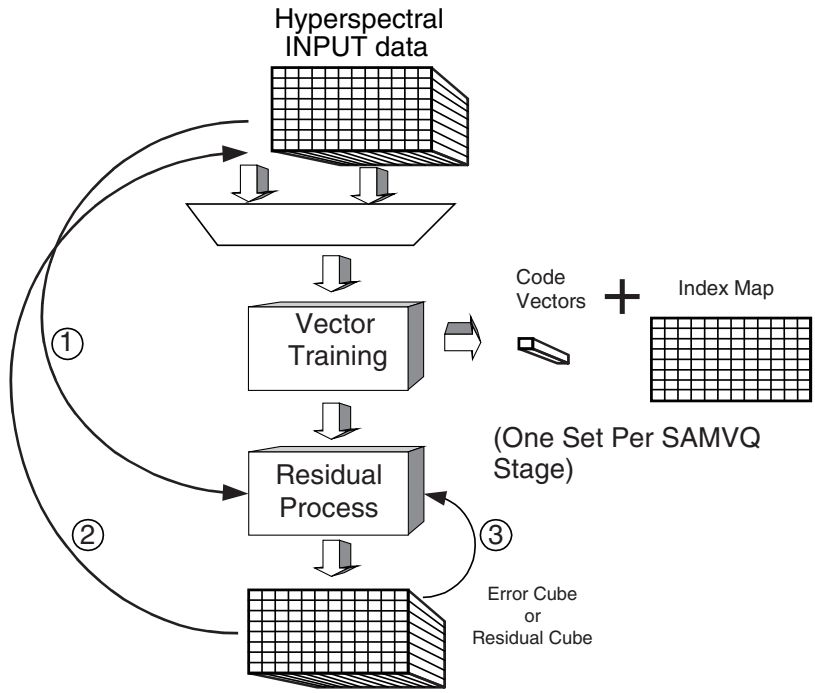


Figure 7.14 Overall view of SAMVQ processing.

Descending in the figure, each of those four clusters (red, yellow, blue, and green) are then treated separately (as a distinct group) because each of these clusters will be analyzed again by the vector training processing.

Using the red cluster as an example here, in the second depth of vector training processing, the cluster is subdivided into four subclusters. This process goes on (tree expansion) until the clusters meet one or more of the following criteria:

1. The number of spectral vectors per cluster is smaller than a predetermined value,
2. The PSNR value of that cluster meets a predefined value, and
3. The RMSE value of that cluster meets a predefined value.

As a cluster converges, the codevectors found by the vector training processing in that cluster are sent to the ground (or to a temporary memory storage external to the CE) along with the pixel referenced by these codevectors.

It can be seen that all clusters and all pixels will eventually converge, and thus all of the pixel's indices will be determined. Similarly, all codevectors related to all converged cluster will be determined and sent to the ground for decompression.

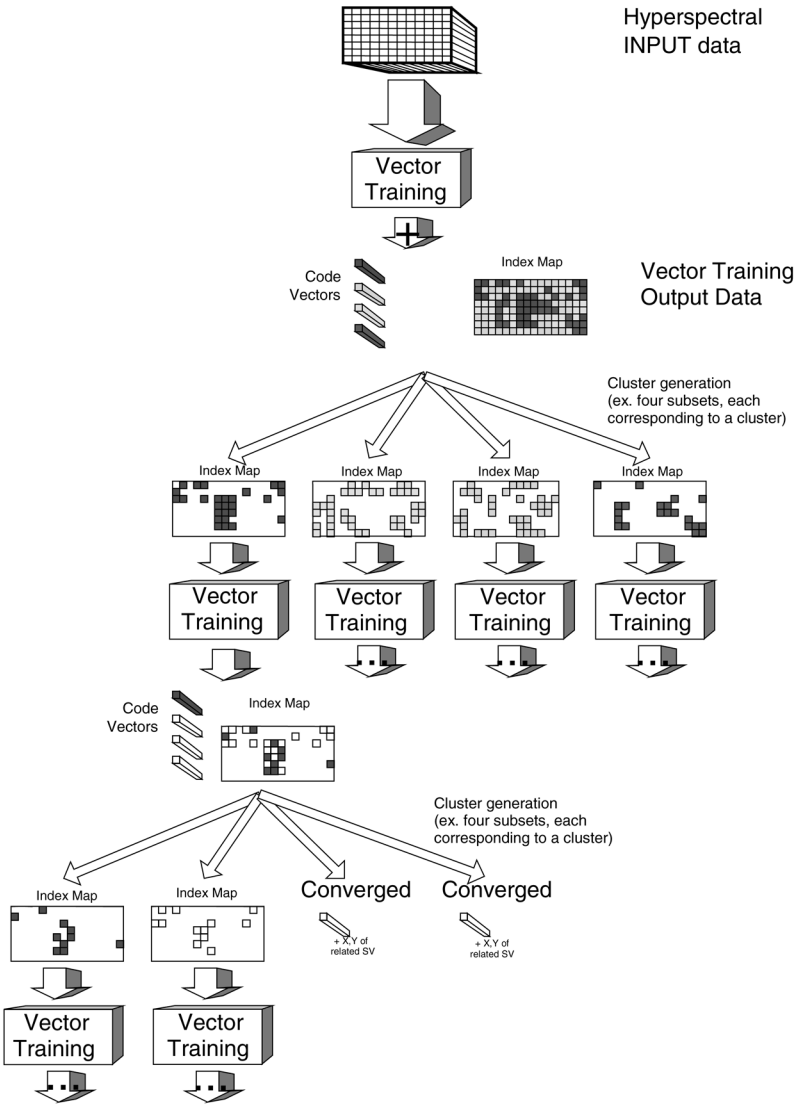


Figure 7.15 Overall view of the HSOCVQ process. For a color version of this figure, see Plate 7 in the color plate section of this book.

7.7 Scenario Builder: A Real-Time Data Compression Emulator

This section describes a real-time data compression emulator, which is referred to as Scenario Builder. It supports hardware configurations (i.e., scenarios) that are "user trials" in the search for an optimal system. It emulates a compressor that uses either cluster SAMVQ or recursive HSOCVQ from a complete family of optimized and synthesized (based on real hardware data) VHDL designs. Scenario Builder is a software tool that enables reliable

prediction of an onboard compressor architectural configuration for a hyperspectral mission. It provides users the ability to determine the optimal hardware implementation.

7.7.1 Scenario Builder overview

Even though both the hardware and software implementations of cluster SAMVQ or recursive HSOCVQ provide identical results, their performances have no common evaluation metrics.

The software models can effectively be used to select the most-appropriate algorithm (or combination of algorithms) to be used for optimal results, but the software models cannot be used to judge the optimized hardware implementation. The best system that supports the mission requirements may lie between the optimal software and hardware implementation.

Scenario Builder provides a software environment that is capable of emulating real data that flows through real hardware models with real hardware timings. It is an emulation of a real-world hardware prototype without producing the hardware.

In the proposed compression system, the real input data is hyperspectral datacubes in band-interleaved-by-pixel (BIP) format. Any BIP-formatted datacubes can be used with any user-defined scenario. The real hardware models are clock-cycle-accurate models of the VHDL implementation in hardware, and the real hardware timings are true operational frequencies extracted from the synthesized work required as the real FPGA chipsets are produced. Scenario Builder is a powerful tool for evaluating any mission requirements.

An overview of Scenario Builder and its components is shown in Fig. 7.16. There are three steps: scenario definition, scenario simulation, and scenario validation.

7.7.2 Scenario Builder applications

Scenario Builder is built over seven applications, shown in the rectangular boxes in Fig. 7.17. The overall design architecture defines the interaction between all of these applications and the data flows between them.

The applications and their characteristics include the following:

1. **Scenario Builder GUI:** The application of a graphical user interface (GUI) is written in Java language, using the Ptolemy (latest version) simulation framework. It helps a user build a scenario and configures each of its components. Then, it streams the scenario into an extensible markup language (XML) model file; XML is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.

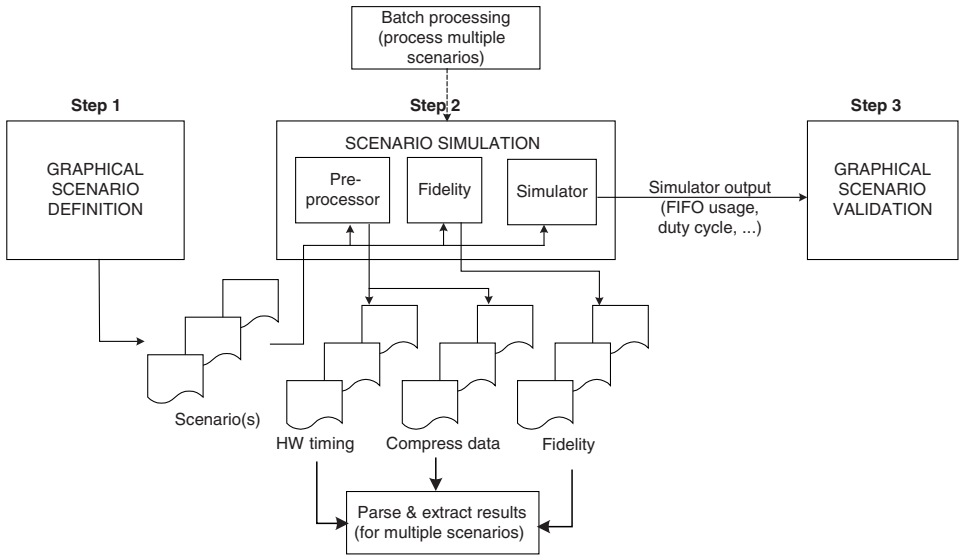


Figure 7.16 Scenario Builder overview.

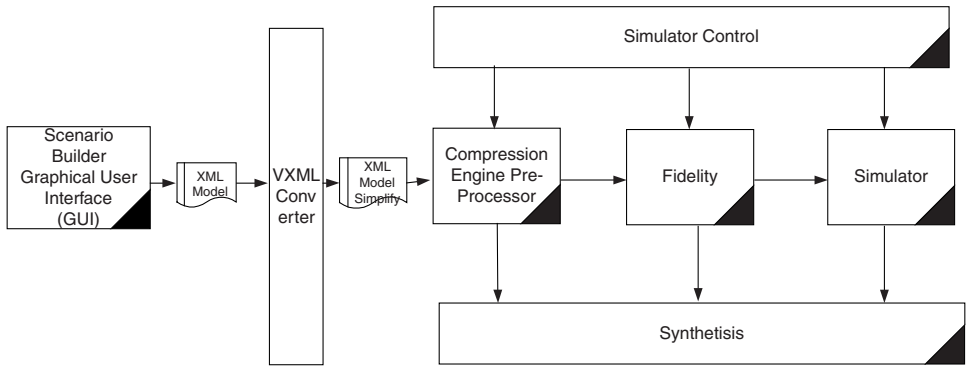


Figure 7.17 Seven applications of Scenario Builder.

2. **VXML converter:** This application is also written in Java language, and its main purpose is to convert the XML model file generated by the Ptolemy framework (consisting of overhead information) and simplify it into a CE preprocessor format.
3. **Compression engine preprocessor:** This application is written in C++ language. It analyzes the simplified scenario model according to the hardware topology and its specification. It then generates the compressed data by compressing the original datacube.
4. **Fidelity:** This application, written in C++ language, uses the compressed data generated in the previous application to reconstruct a compressed

datacube and compares it to the original. After the comparison, it generates statistical measures such as CR, SNR, PSNR, %E, etc.

5. **Simulator:** This application is written in C++ language and uses the DiscreetC framework to provide simulation capabilities. It generates information using the Ptolemy framework (.vcd files). The framework can be displayed in visual graphics for the compression engine state, the first-in first-out (FIFO) buffer usage, the duty cycle, etc.
6. **Simulator control:** This application, written in C++ language, controls applications 3–5. It starts these applications in different processes and links them with a stream pipe. It monitors the execution of the applications and handles errors reported from them.
7. **Synthesizer:** This application, written in C++ language, loads each report and information file from applications 3–5 and synthesizes them into one XML report.

7.7.3 Architecture and data flow of Scenario Builder

Figure 7.18 shows the architecture of Scenario Builder. The simulated component software-configurable-items (CSCIs) are expanded in the figure.

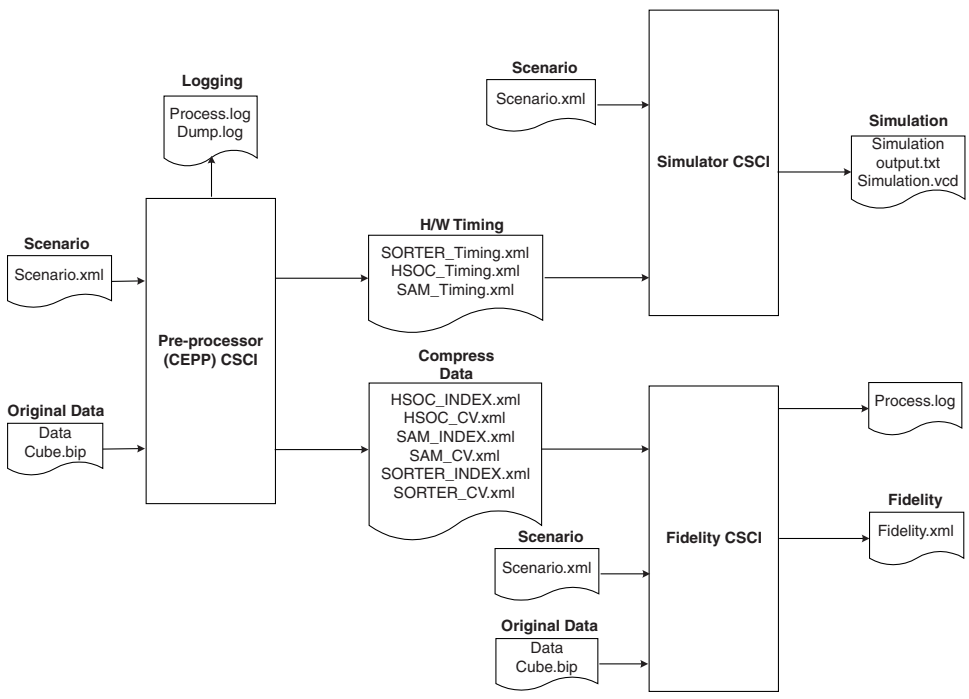


Figure 7.18 The architecture and data flow of Scenario Builder.

The scenario simulator is parsed into 3 CSCIs (the three rectangular boxes in the figure):

1. The preprocessor CSCI analyzes the user's scenarios according to the hardware topology and its specification. It generates the compressed data by compressing the original datacube. When performing the previous task, the preprocessor monitors the compression process based on hardware implementation and generates timing metrics.
2. The fidelity CSCI uses the compressed data generated by the preprocessor to reconstruct a datacube and compares it to the original one. Then, from the comparison, it generates quality metrics.
3. The simulator CSCI uses the timing metrics generated by the preprocessor to simulate a hardware model of a compression engine. The simulator generates FIFO buffer usage, compression engine duty cycle, and compression timing, and it detects FIFO buffer bottleneck.

7.7.4 SORTER engine and cluster SAMVQ compression engine

As described in Chapter 5, both cluster SAMVQ and recursive HSOCVQ divide an input datacube into subsets of spectral vectors by clustering the spectral vectors of the datacube based on the similarity of the spectral vectors to overcome the constraints of CPU processing speed, memory, and power. Each subset of spectral vectors is then fed to a CE for parallel processing. The operation of clustering spectral vectors of the datacube is carried out by a dedicated CE to handle the heavy processing and the large number of spectral vectors; this engine is referred to as "SORTER." The consequent operation of cluster SAMVQ is carried out by a second-level CE, which can operate by using either an external RAM to handle a large subset of spectral vectors or its own internal RAM to handle a relatively small subset of spectral vectors.

Scenario Builder supports five different types of CEs, and each type has its unique characteristics, limitation, and capability:

- SORTER,
- SAMVQ with external RAM,
- HSOCVQ with external RAM,
- SAMVQ without external RAM, and
- HSOCVQ without external RAM.

Figure 7.19 shows the global view of SORTER. SORTER receives an input cube from a hyperspectral sensor. The characteristics of the input cube are constants, i.e., the cube is of a constant number of bands and of constant spatial size. The number of spectral vectors in the cube is $> 2^{16}$ (64K) SVs.

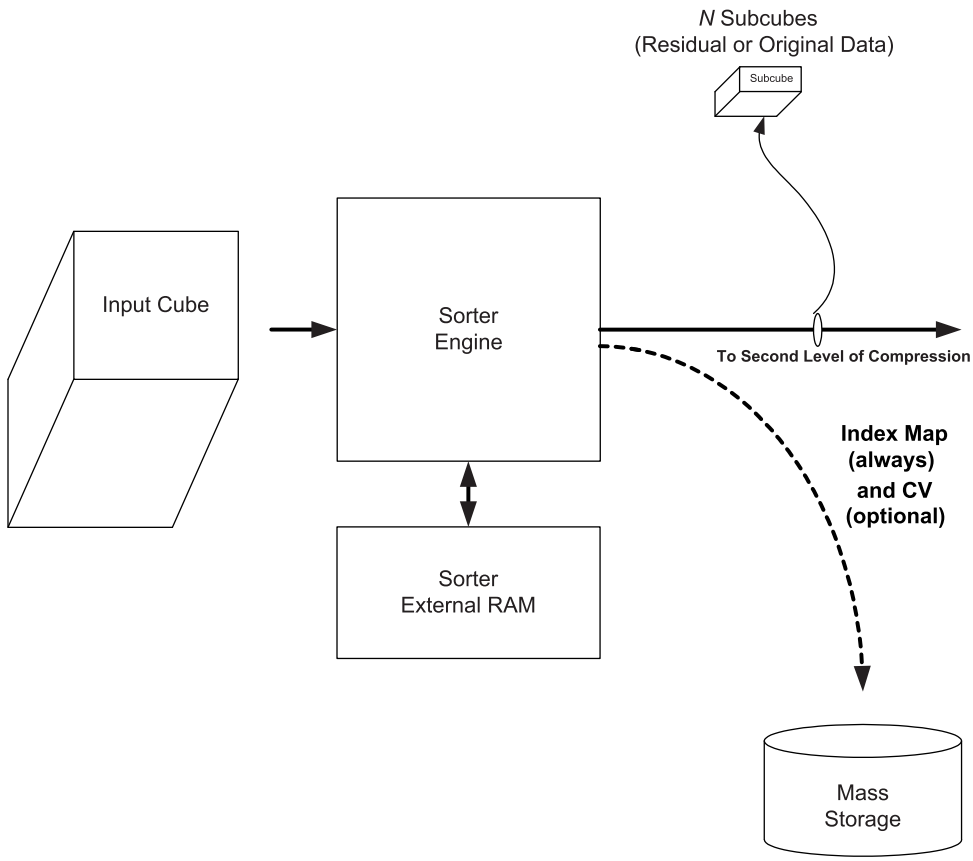


Figure 7.19 SORTER compression engine overview.

Globally, the SORTER operations can be described as follows:

- Read the input cube.
- Classify the datacube into N subsets (i.e., clusters) of spectral vectors based on spectral similarity, whereby N codevectors are generated, and a cluster map (also referred to as index map, i.e., subset ID of each spectral vector of the datacube) is created.
- Transfer the cluster map to the mass storage for transmission to the ground.
- Send the N subsets to the consequent CEs for compression using SAMVQ or HSOCVQ.
- Optionally, calculate the difference (i.e., residual) between the subsets and reconstructed subsets using the cluster map and the N codevectors, and send the N residual subsets to the consequent CEs for compression using SAMVQ or HSOCVQ. In this scenario, the N codevectors also need to be transferred to the mass storage for transmission to the ground.

The following guidelines were used in the design of SORTER:

- The subsets are formed based on the similarity of the spectral vectors.
- The sizes of the subsets need to be as close as possible in order to distribute the subsets to the next-level CEs with even size for better use of the CE's resource.
- None of the spectral vectors in the input datacube should be read more than twice.

In order to accomodate the sizes of subsets with small variances, the initial codevectors used to classify the input datacube need to be representative of the input cube. Selection of these "ideal" codevectors need not require reading the complete datacube but rather a representative subsampling of the entire datacube.

Reading a specific number of spectral vectors with spatially equidistant subsampling from the input cube and then performing LBG on these subsampled spectral vectors provides a good subsampling strategy. If all of these spectral vectors can be stored within the SORTER CE internal memory, then the codevector-selection processing time will be optimized. Once the initial codevectors are selected, all spectral vectors of the input cube are read and classified per their closest codevector. Each spectral vector is read only once in the clustering process with this strategy.

If the residual option is chosen in SORTER, the residual data needs to be calculated and transferred to the next-level CE. The residual calculation is performed as soon as the best-match codevector is determined to avoid reading a spectral vector twice. The external RAM bottleneck can be easily avoided if the number of codevectors in use is large enough.

To speed up the codevector training process and save memory usage, subsampled spectral vectors are formed and used to train the codevectors. First, n equally spaced spectral vectors are read from the input datacube and used for codevector training; this equally spaced selection of the spectral vectors produces diagonal patterns, as shown in Fig. 7.20. This selection method provides excellent spatial subsampling of the input datacube for codevector training.

Once the spectral vector RAM is filled with subsampled spectral vectors, the LBG iterative process takes place to update the initial estimate codevectors, resulting in N codevectors that can best fit the subsampled spectral vectors. The N codevectors are now ready to be used in the clustering process. The iteration threshold (standard epsilon used to exit LBG processing) used in the LBG process is 1/1024 or smaller. The choice of N has to be established using the following driving parameters:

- N needs to be large enough so that no unnecessary external RAM bottleneck is created in the clustering process.

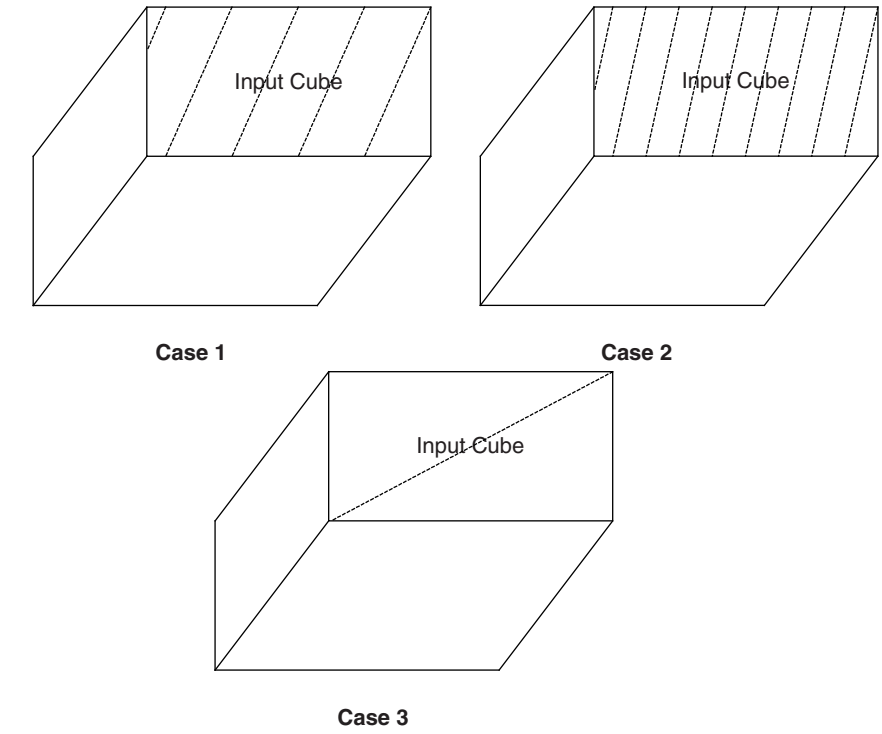


Figure 7.20 Subsampling an input datacube with equally spaced spectra selection for codevector training.

- N must be selected so it creates subset sizes that are typically close to the consequent CE internal memory capability to reduce RAM bottlenecks in the consequent CE. Because the variance of the subset sizes is small, all subsets (or packets) should not produce excessive RAM bottleneck.

Once N codevectors have been trained and stored in the SORTER CE internal memory, the classification process can begin. All spectral vectors of the input cube are read and compared to the N codevectors; the scalar distance (Euclidean) is used to determine the best-matched codevector. The index of the codevector is then assigned to each spectral vector to form the index map. When all spectral vectors have been matched with the closest codevector, the SORTER process is complete.

Cluster SAMVQ by definition operates on clusters created by SORTER, hence the cluster SAMVQ engine (or SAMVQ baseline engine) could only be a second-level engine.

Baseline SAMVQ might also be used as the first-level engine (as a sorter), generating an index map and residual datacube for the second level of processing. This configuration may seem interesting, but the SORTER operations can provide better results than baseline SAMVQ as a SORTER.

Still, if a user wants to try baseline SAMVQ in the first level (in stage mode), the codebook size must be increased so that the RAM bottleneck disappears. The stage number to be performed should also be maintained in the 1–3 range.

7.7.5 Recursive HSOCVQ compression engine

Recursive HSOCVQ is an updated version of HSOCVQ for onboard use that would enable easier implementation of the algorithm in hardware. It also incorporates interesting features regarding the single-event upsets (SEUs) sensitivity of the design, which has been described in Chapter 5. Without going into the details of the recursive variation, the original idea could save both processing time and external storage capacity.

This section defines the configurations whereby recursive HSOCVQ can be used effectively and where it could not be supported; Figure 7.21 is used to facilitate the description. Scenario Builder analyzes four configurations (i.e., cases) and validates their operational ability and effectiveness. In the figure, a CCD instrument is used as an example of a hyperspectral data source.

Case 1: In this case, the first-level compression using a CE of SORTER, residual SAMVQ, or cluster HSOCVQ generates uncorrelated compressed data. The HSOCVQ CE at the second level (final box) cannot implement the recursive feature of the algorithm because the compressed data is uncorrelated. Performing the comparison using codevectors in the previous region is irrelevant when the sequential packets are uncorrelated (see detailed description of recursive HSOCVQ in Chapter 5).

Case 2: As in case 1, the recursive HSOCVQ feature cannot be implemented in case 2 because there is no correlation between packets from one packet to another (note that bandsplitting is used). The first packet might be high-number bands, then medium-number band, etc. (final box).

Case 3: This configuration is a valid one, where recursive HSOCVQ can be efficiently used (final box). It is supported by Scenario Builder. All sequential packets relate to the same bands, and the data being fed to the CE is raw data (as opposed to residual data).

Case 4: This configuration can be used with the following precautions (final boxes):

- All bands split (split 1, 2, or *n*) need to be fed to a specific CE (CE 1, 2, or *n*),

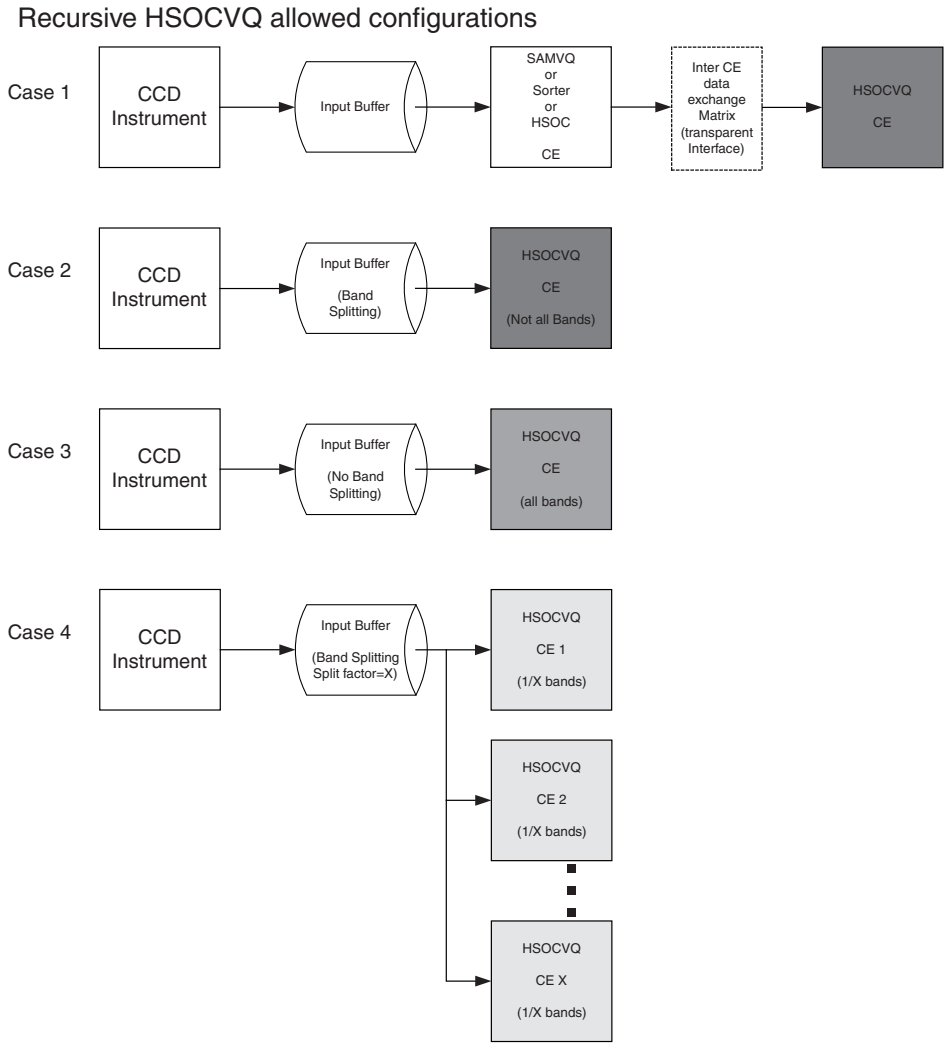


Figure 7.21 Configurations validated by Scenario Builder for recursive HSOCVQ.

- No other packets except its related band packet should be sent to that CE, and
- The CE cannot skip an imaging session (sequential packet) for best performance.

Scenario Builder does not support the HSOCVQ recursive variation over this configuration directly. In order to test the scenario, a user needs to perform n simulations, where n is the band splitting factor used in case 4 and where each simulation operates on the bands of interest (n splits).

The configuration of case-3 recursive HSOCVQ is valid and can be implemented. The following defines the parameters that provide the best results using that configuration.

It has been mentioned earlier that the minimal input cube size that could be used is 64K spectral vectors. It is also known that the CE internal memory for HSOCVQ is quite limited (ranging from 512 to 4,000 spectral vectors depending on the number of bands in use), which leads directly to the conclusion that the vast majority of spectral vectors will reside outside the CE and that the CE will have to perform numerous RAM accesses to satisfy the recursive HSOC processing.

The disadvantage of this configuration is that the external RAM accesses become a bottleneck for HSOCVQ processing. The RAM bottleneck must be analyzed in the two modes in which recursive HSOCVQ operates: compare with previous region codevectors (CWPRC) and HSOCVQ tree expansion (baseline HSOCVQ).

Within CWPRC processing, there are three cases:

- First frame, first region,
- First frame NOT first region, and
- Not first frame.

Having frames of at least 512 spectral vectors (one cross-track line of the CCD instrument) and a codevector reuse rate of 20–30% minimum from one region to the first frame of a new region, it was found that there is no bottleneck in the RAM interface because a codevector is required every 20–30% of 512, i.e., every 100–150 clock cycles, which is more than enough to fetch a new spectral vector.

For basic HSOCVQ processing, the worst-case scenario is when the tree expansion operates on a large number of unconverged spectral vectors (after CWPRC has completed). There are two options to reduce that bottleneck (as neither the number of bands in use nor the number of spectral vectors under processing can be reduced): use faster external RAM or use a large number of CVs as the HSOCVQ tree expands.

These measures are not always possible or efficient. Using faster external RAM quickly reaches a limit. Using a large number of codevectors for the HSOCVQ tree expansion could be good processing-wise but is not very efficient where the HSOCVQ lower-level tree processing is concerned.

In conclusion, recursive HSOCVQ or even baseline HSOCVQ is not the most-efficient engine in the first level of compression (except when used in fidelity mode). The bottleneck can be compensated for by the efficiency of the algorithm.

7.7.6 Scenario Builder products

This section describes the Scenario Builder products and their implementations. Overall views of the mechanisms and concepts that were used to define

the products are provided first, followed by the details and operational information.

As described earlier, Scenario Builder is a software environment that is capable of emulating real data that flows through real hardware models with real hardware timings. A Scenario Builder product can be viewed as an emulation of a real-world hardware prototype without producing that hardware. It supports many hardware configurations, i.e., scenarios. These scenarios are "user trials" for designing an optimal system, which the hardware prototyping cannot offer.

7.7.6.1 SORTER

SORTER, described in Section 7.7.4, is the best engine for the first level of SAMVQ compression. It does not generate RAM bottlenecks even with very large input-cube sizes when the cluster number is set to the typical value $N =$ 32 codevectors or more. Its efficiency resides in the fact that

- The initial codevectors are determined without using the external RAM block (spectral vectors are read only once for the codevector estimation process).
- Each spectral vector of the input datacube is read only once in the clustering process.
- The SORTER engine can generate residual data to the next-level SAMVQ compression, and generate and then send the index map and codevectors to the mass storage for ground reconstruction.
- The hardware timings (and the frequency of operation) used for the SORTER are based on SAMVQ with a RAM engine, as that engine offers all of the elements required for the SORTER operation (SORTER is, in fact, a SAMVQ engine with internal RAM and a slightly different algorithm controller).

7.7.6.2 SAMVQ engine with external RAM

The SAMVQ engine with external RAM implements the SAMVQ algorithm by using both its internal memory and the external RAM block attached to it. The engine is most efficient when operating on a reduced number of spectral vectors or on a reduced number of bands (which increases its internal RAM capacity as per). Because the minimal input cube to be processed is in the range of more than 64,000 spectral vectors, the SAMVQ engine with external RAM necessarily requires a front-end SORTER at level-1 compression. It is suggested for best performance that the number of spectral vectors fed to the engine does not exceed 2 times the SVs' capacity.

Because Scenario Builder requires a SAMVQ engine with external RAM of variable characteristics (i.e., different numbers of bands), the frequency of operation used and the CE internal RAM capacity will be derived linearly.

The SAMVQ engine with external RAM is required to be at the second level of compression (with SORTER in the first level) to remove any spatial blocking and boundary effects. The engine can operate in threshold or in level mode.

7.7.6.3 HSOCVQ engine with external RAM

The HSOCVQ engine with external RAM implements the HSOCVQ/recursive HSOCVQ algorithm by using both its internal memory and the external RAM block attached to it. The engine is most efficient when operating on a reduced number of spectral vectors or on a reduced number of bands (which increases its internal RAM capacity), or when the number of codevectors in use overcomes the RAM bottleneck. The engine can operate on residual or raw data, and can be fed by SORTER.

7.7.6.4 Clock-accurate hardware timing

As software compression is performed with the data file specified in the scenario, the application software computes the exact number of clock cycles that would be required by the real hardware CE. This computation is dictated by the VHDL (hardware) implementation of the algorithm in the specific CE in use.

Two special cases need to be identified with regards to the hardware formula used to evaluate the hardware performance of the CE in use: serial link duty cycle and external RAM throughput.

There has been some discussion concerning the link to be used to feed RAM and compressed data to/from the CE. That link was initially serial, enabling very dense usage of a CE on a PCB. Based on this discussion, EMS chose to characterize that link in terms of throughput. The serial-link duty cycle encodes that measure.

It is based on the scaling of a 32-bit word feeding the CE at every clock cycle. This means that if the specified serial-link duty cycle is 3, then the throughput (reception or transmission throughput) is 3×32 bits $\times$ Foper CE. For example, a CE operating at 75 MHz with a duty cycle of 1 leads to a throughput of 2.4 Gbps. The serial-link duty cycle value can range from 100 to 0.001.

In the same vein, the external RAM throughput has been encoded on a reference where 128 bits of data are fed at every clock cycle into the CE (or being output by the CE to the external RAM bank). As an example, having a CE operating at 75 MHz with an external RAM throughput of 0.33 leads to $0.33 \times$ Foper CE $\times$ 128 bits $= 3.17$ Gbps. The external RAM throughput value can range from 100 to 0.001.

This encoding scheme for the serial links and for the external RAM interface performance enables testing different RAM devices or different

implementation for the CE RAW or the compressed data upload and download.

7.7.6.5 Hardware bottleneck emulation

Scenario Builder can be either a GUI interface or a batch processing (command line with text reporting) interface.

It is used to judge the capacity of the proposed scenario hardware configuration to sustain the CCD Instrument output rate without overflow. The GUI interface provides an animated (and graphical) representation of all element usage:

- All CE duty cycles (from 0–100%),
- The FIFO or RAM instantaneous and maximum capacity during simulation, and
- Some other quality statistics.

The batch processing provides the same information in a text file. For that purpose, as shown in Fig. 7.16, step 3 of Scenario Builder uses two main inputs:

- Hardware timings from step-2 simulation and
- Fidelity data from step-2 simulation

Some additional notes:

The main criterion to validate a specific scenario (besides the level of quality of the compressed data) is the FIFO maximum capacity during simulation (or the CE duty cycle approaching 100%, which is the same phenomenon). In order to properly assess the validity of the specific scenario, multiple cubes should be sent by the CCD instrument.

If an overflow occurs, many solutions are possible. For example, increase the number of CEs in the compression step that cause the bottleneck, or increase the threshold of that bottleneck CE (or decrease its level for an HSOCVQ in level mode or its stage number for a SAM engine used in stage mode, etc.). If SORTER is used, and the second-level compression overflows, increase the SORTER splitting factor, modify the original scenario, and test again.

7.7.7 Scenario simulation user interface

A scenario can be defined in the graphical scenario definition (step 1 in Fig. 7.16). Figure 7.22 shows the directory structure and a typical scenario with a producer, one FIFO, two stages of compression (one SORTER and one HSOCVQ), and consumers. These components are parameterized through the graphical interface. With this information, the graphical scenario definition generates a scenario file in XML format to be used by the scenario simulation. The scenario simulation uses the scenario template to validate the

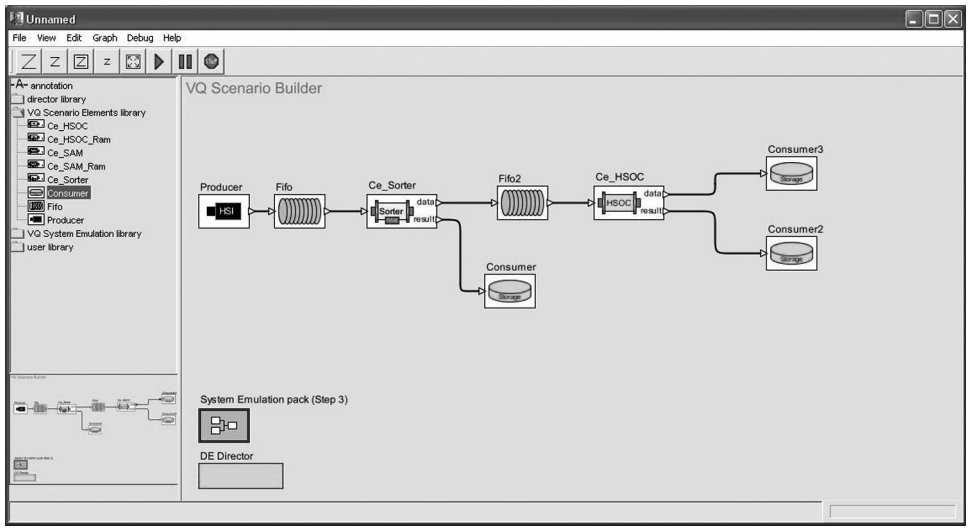


Figure 7.22 User graphical scenario display.

scenario against its syntax. However, not every scenario is supported, as described in Section 7.7.5. Some internal constraints also exist: each CE composing a stage has to be identical, and each block generated by the producer has to be the same size.

Each simulation uses its own directory structure where it finds its input files and stores its output files; the structure shown in Fig. 7.22 is an example. In this example, because the scenario defines two stages of compression (one SORTER and one HSOCVQ), the directory contains two subdirectories (CE1 and CE2) where the output files of these two compression engines are stored.

7.8 Using Scenario Builder to Optimally Design Onboard Data Compressor Architecture

Scenario Builder provides a means for a user to trade-off for the most promising architecture of an onboard compressor that implements either the SAMVQ or HSOCVQ algorithm. As described in Section 7.7, Scenario Builder permits for variations of

- Hyperspectral imager configuration,
- SORTER CE usage,
- HSOCVQ CE usage,
- SAMVQ CE usage,
- Compression ratio, and
- Compression fidelity.

7.8.1 Parameters of the design

Based on the technology at the development time, Scenario Builder supports the following parameters:

- 640 pixels per cross-track line,
- 250 Hz frame rate,
- 12 bits/pixel,
- 80 bands per spectral vector, and
- Minimal cube size of 64,000 spectral vectors, each with 80 spectral bands.

The example shown here uses the parameters of a hyperspectral sensor from the Canadian Hyperspectral Environment and Resource Observer (HERO) mission.[10]

There are a total of 60 bands in the VNIR (400–1000 nm) region that fit the vector length (80) of the compression engines. There are a total of 156 bands in the SWIR (900–2500 nm) region. These bands are split into two subgroups of 78 bands each, referred to as SWIR A and SWIR B, in order to fit the vector length (80) of the compression engines. These two subgroups of spectral bands are compressed independently. The number of pixels in a cross-track line is 640 pixels. The focal plane frame rate is 230 Hz. The data is 12-bit quantization.

When an input datacube size is in the range of 64,000 spectral vectors (i.e., 100 cross-track lines acquired by the satellite with 640 ground samples per line), the compression of that datacube at the first level must be completed in less than 400 ms, otherwise data overflows are likely to occur. Table 7.2 lists the processing time for HSOCVQ or SAMVQ to compress an input datacube of 64,000 spectral vectors (CE operating at 100-MHz clock cycle).

The only engine that can be used as the front-end CE is a SORTER (the next section elaborates on this avenue). The second-level compression will use either a SAMVQ or an HSOCVQ engine.

7.8.2 SORTER as the front-end compressor

The parameters that can be tuned for SORTER when used as the front-end compressor are as follows:

- Input cube size,
- Number of bands,
- N number of clusters or subsets,
- (Non-)group mode, and
- Residual mode.

As it is understood the first two parameters have already been set to the following constraints: input cube size potentials = 64, 80, 96, 112, 128, and

Table 7.2 Processing time for HSOCVQ or SAMVQ in compressing an input datacube of 64,000 spectral vectors (CE operating at 100-MHz clock cycle).

Type of Engine	Mode of Operation	Processing Time (ns)	Overload Factor for a Single CE	Number of CEs Required for Real-Time Operations
SAMVQ with RAM	Fixed stage (1 stage) with 32 CVs	677,297,760	1.7	2
HSOCVQ with RAM	Fixed fidelity mode (1 level) with 32 CVs	1,538,234,890	3.8	4
SAMVQ with RAM	Fixed stage (2 stages) with 32 CVs	944,953,720	2.4	3
HSOCVQ with RAM	Fixed fidelity mode (1 level) with 64 CVs	2,070,788,330	5.2	6
SAMVQ with RAM	Fixed stage (1 stage) with 64 CVs	1,045,981560	2.6	3
HSOCVQ with RAM	Fixed fidelity mode (2 levels) with 32 CVs	2,034,219,160	5.1	6
SORTER with RAM	Fixed fidelity mode with 32 CVs	70,622,440	0.17	1
SORTER with RAM	Fixed fidelity mode with 64 CVs	100,947,150	0.25	1

256K spectral vectors, and number of bands = 80, the number of clusters N will then be tried for $N = 4, 8, 16, \ldots, 512$.

The group mode that will be used (SORTER will generate N subsets) will be individually compressed by the second-level CE.

The residual mode will be used (the residual values are already available in the SORTER), leaving only N as the true varying parameter. From these parameters, the evaluation results are listed in Table 7.3. It is understood from the results in the table that the preferred operational case for SORTER is a single CE with the biggest SORTER number of clusters N and a duty cycle above 50% and below 100%.

7.8.3 Second-level compressor

The second level of compression can be inferred to be either a HSOCVQ or a SAMVQ engine in threshold mode because these engines are the ones that will define final fidelity and CR (they are the last ones to process the data).

The methodology used will specify for both engines a threshold for which the overall CR is around 20:1. This threshold will vary depending on the SORTER programming parameters. Table 7.4 lists HSOCVQ performance as the second-level CE.

Table 7.3 SORTER performance as the front-end (first-level) compression engine.

Input Cube Size (K SV of 80 bands)	Number of Clusters (*N*)	Processing Time (ns)	Duty Cycle of SORTER	Number of CEs Required
64	4	47857270	11.964	1
80	4	59531610	11.906	1
96	4	71371450	11.895	1
112	4	83792260	11.970	1
128	4	95466000	11.933	1
256	4	190517640	11.907	1
64	8	49668990	12.417	1
80	8	60849990	12.17	1
96	8	71864440	11.977	1
112	8	84034900	12.005	1
128	8	95709390	11.964	1
256	8	191089190	11.943	1
64	16	53813630	13.453	1
80	16	66774590	13.355	1
96	16	81049750	13.508	1
112	16	94340540	13.477	1
128	16	105982660	13.248	1
256	16	209333630	13.083	1
64	32	70622440	17.656	1
80	32	86146600	17.229	1
96	32	103634010	17.272	1
112	32	115214070	16.459	1
128	32	134674010	16.834	1
256	32	344686620	15.89	1
64	64	100947150	25.237	1
80	64	115025360	23.005	1
96	64	142227150	23.705	1
112	64	182588360	26.084	1
128	64	176939600	22.117	1
256	64	344686620	21.543	1
64	128	155029890	38.757	1
80	128	206944690	41.389	1
96	128	214163740	35.694	1
112	128	247685250	35.384	1
128	128	281176040	35.147	1
256	128	530842190	33.178	1
64	256	268449410	67.112	1
80	256	314586140	62.917	1
96	256	402677990	67.113	1
112	256	412052150	58.865	1
128	256	468637980	58.58	1
256	256	895272270	55.955	1
64	512	453280490	113.32	2
80	512	535134340	107.027	2
96	512	658924470	109.821	2
112	512	719774340	102.825	2
128	512	833062480	104.133	2
256	512	1613630440	100.852	2

Table 7.4 HSOCVQ performance as the second-level CE.

Input Cube Size (K SV)	Number of Clusters (*N*)	HSCCVQ Threshold in Use	Overall Compression Ratio	Overall SNR, PSNR	HSOCVQ CE Duty Cycle
64	128	7.0	16.36	40.27,51.00	16.547
64	128	8.0	23.78	39.51,50.24	15.518
64	256	7.0	14.51	40.80,51.53	13.40
64	256	8.0	20.09	40.04,50.77	12.38
80	128	7.0	19.56	39.98,50.69	17.696
80	128	8.0	28.48	39.19,49.91	16.178
80	256	7.0	16.77	40.60,51.32	14.674
80	256	8.0	23.45	39.91,50.63	13.546
96	128	6.0	11.73	41.25,51.96	19.952
96	128	7.0	20.08	40.10,50.81	17.616
96	256	7.0	18.51	40.53,51.24	14.762
96	256	8.0	26.01	39.63,50.34	13.129
112	128	6.0	12.93	41.18,52.87	19.328
112	128	7.0	21.83	40.03,51.72	17.040
112	256	7.0	19.38	40.68,52.37	14.244
112	256	8.0	29.13	39.87,51.56	12.981
128	128	6.0	13.15	41.29,53.28	20.356
128	128	7.0	22.20	40.11,52.10	17.964
128	256	6.0	12.44	41.68,53.66	16.477
128	256	7.0	20.64	40.56,52.55	14.437
256	128	6.0	14.83	40.92,52.82	22.790
256	128	7.0	25.89	39.88,51.78	20.132
256	256	6.0	13.99	41.39,53.28	18.648
256	256	7.0	24.58	40.23,52.13	15.931

7.8.4 Proposed system

From the descriptions in previous sections, the system depicted in Fig. 7.23 is selected as the proposed compression system configuration for the example mission.

This system can perform real-time compression of hyperspectral data with the following characteristics (foreseen HERO VNIR and SWIR characteristics):

- 80 bands per spectral vector,
- 640 cross-track pixels,
- 250-Hz frame rate,
- 12 bits/pixel, and
- 64,000-spectral vector minimal cube size.

The different devices are programmed as follows:

Hyperspectral sensor parameters

Rate:	250 Hz
Cube path:	Must point to source datacube *.bip

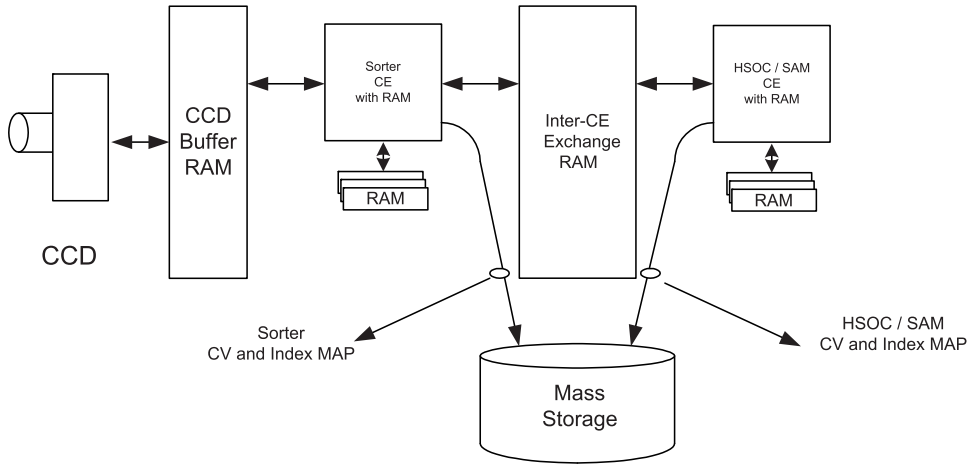


Figure 7.23 Best architecture for the HERO mission onboard-compression specification.

Cube *Nr*:	xxx (cube dependant)
Cube *Nc*:	yyy (cube dependant)
Cube *N* bands:	80
Block band offset:	0 (or other values)
Block *N* bands:	80
Block size:	64,000 and above

SORTER parameters

Clock period:	10 (ns)
Xram throughput:	1.0
Cube size:	64,000 and above
Cluster size (SSF):	256
Nb Stage:	1
Nb bands:	80
Nb stage:	0
RMSE threshold:	0

SAMVQ parameters

Clock period:	10 (ns)
Xram throughput:	0
Cube size:	64,000 and above
Cluster size (SSF):	256
Nb stage:	0
Nb bands:	80
RMSE threshold:	D

HSOCVQ parameters

Clock period:	10 (ns)
Xram throughput:	1.0
Cube size:	64,000 and above
Number of frames:	1
Frame size:	64,000
Nb bands:	80
Nb levels:	0
RMSE threshold:	TBD

From these, the system provides the following results:

- Overall CR,
- Overall fidelity (RMSE, PSNR, SNR, %R, etc.),
- SORTER duty cycle,
- SAMVQ duty cycle, and
- HSOCVQ duty cycle.

References

1. Qian, S.-E. and A. Hollinger, "Method and System for Compressing a Continuous Data Flow in Real-Time Using Cluster Successive Approximation Multi-Stage Vector Quantization," U.S. Patent No. 7,551,785, issued on June 23, 2009.
2. Qian, S.-E. and A. Hollinger, "Method and System for Compressing a Continuous Data Flow in Real-Time Using Recursive Hierarchical Self-Organizing Cluster Vector Quantization," U.S. Patent No. 6,798,360 B1, issued on September 28, 2004.
3. Liptak, B. G., *Instrument Engineers' Handbook: Process Control and Optimization*, pp. 11–12, CRC Press, Boca Raton, FL (2006).
4. Golshan, K., *Physical Design Essentials: An ASIC Design Implementation Perspective*, Springer, New York (2007).
5. Wisniewski, R., *Synthesis of Compositional Microprogram Control Units for Programmable Devices*, pp 153, University of Zielona Góra, Zielona Góra, Poland (2009).
6. Mano, M. and C. Kime, *Logic and Computer Design Fundamentals*, 2nd Ed., Prentice Hall, Upper Saddle River, NJ (2001).
7. Linde, Y., A. Buzo, and R. M. Gray, "An algorithm for vector quantizer designs," *IEEE Trans. Comm.* **28**, 84–95 (1980).
8. Qian, S.-E., A. Hollinger, and L. Gagnon, "Codevector Trainers and Real-Time Compressor for Multi-Dimensional Data," U.S. Patent No. 8,107,747 B2, issued on Jan. 31, 2012.

9. Qian, S.-E., A. Hollinger, and L. Gagnon, "Data Compression Engines and Real-time Wideband Compressor for Multi-dimensional Data," U.S. Patent No. 7,251,376 B2, issued on July 31, 2007.
10. Hollinger, A., M. Bergeron, M. Maszkiewicz, S.-E. Qian, K. Staenz, R. A. Neville, and D. G. Goodenough, "Recent Developments in the Hyperspectral Environment and Resource Observer (HERO) Mission," *Proc. IGARSS 2006*, 1620–1623 (2006).

Chapter 8
User Acceptability Study of Satellite Data Compression

8.1 User Assessment of Compressed Satellite Data

To deal with the extremely high datarate and huge data volume generated aboard a hyperspectral satellite, lossless and lossy data compression techniques have been developed;[1 24] these techniques can significantly reduce the amount of data onboard and on-ground. Chapters 4 and 5 of this book describe two near-lossless data compression techniques, referred to as successive approximation multistage vector quantization (SAMVQ) and hierarchical self-organizing cluster vector quantization (HSOCVQ), that compress hyperspectral data with a high compression ratio and restrict the compression error at the same level or even smaller than the intrinsic noise of the original data. This low-level compression error is expected to have a minor to negligible impact on ultimate applications of the data, so this kind of compression is considered to be near-lossless compression. Even so, they are still lossy compression algorithms. It is essential to assess the usability of the compressed data and to examine acceptability to users in terms of their end products and remote sensing applications. It is critical that the compression techniques preserve the information content of hyperspectral data, as a loss of information content would decrease the value of the data.

Studies of the usefulness of the compressed data and the impact of the developed vector quantization (VQ) data compression techniques on various hyperspectral data applications have been carried out and reported.[25 35] For example, the evaluation of the effect of the early VQ compression algorithms on the retrieval of red-edge indices and surface reflectance have been reported.[25,28,29] In the work reported by Qian et al.,[25] hyperspectral datacubes acquired using AVIRIS and CASI were used to evaluate the impact of the compression. The overall error in the red-edge indices due to compression was below 3.0% and 2.0% for CASI and AVIRIS datacubes, respectively, for compression ratios up to 100:1. Errors were uniformly distributed throughout vegetated areas. In the work reported by Hu et al.,[28,29]

for atmospheric correction of hyperspectral images using best-estimate input parameters, the mean and standard deviation of the surface reflectance over nine specified regions of interest (ROIs) from a CASI datacube were compared with those of the reflectance from the reconstructed datacubes at compression ratios of 42:1 to 104:1. Hypothesis tests at the 95% confidence level found no evident differences in the shapes of the average surface reflectance spectra. It was found that the uncertainty in the retrieved surface reflectance caused by the uncertainties in the atmospheric correction input parameters was larger than the uncertainty caused by the compression algorithm.

The near-lossless compression algorithms SAMVQ and HSOCVQ were also evaluated and reported.[26,27,30–35] The evaluation was carried out using three evaluation systems with compression performed at different stages of the data-flow chain of an imaging spectrometer.[26] The first evaluation system had the onboard compressor placed immediately following the A/D converter (i.e., using sensor digital count as input), the second evaluation system had the onboard compressor placed after the detector's gain and offset (called scaling) were corrected, and the third evaluation system had the onboard compressor placed after a full calibration (i.e., using radiance as input). Datacubes acquired using AVIRIS, CASI, and Probe-1 were tested; statistical tests were performed on the radiance and reflectance datacubes for all three systems. Preliminary evaluation results of the impact of the compression on red-edge, leaf chlorophyll content, and spectral unmixing were reported.

The performance of SAMVQ was evaluated versus a 3D compression algorithm designed for hyperspectral imagery using the JPEG2000 standard.[27] It was found that the SAMVQ algorithm outperforms the JPEG2000 algorithm by 17 dB of PSNR for the same compression ratios. The preservation of both spatial and spectral features was evaluated qualitatively and quantitatively. SAMVQ outperformed JPEG2000 in both spatial and spectral feature preservation. The spectrum of a spatial sample in the scene of the SAMVQ-compressed data was compared with that from the original data at the same location—the reconstructed spectrum curve overlaps the original spectrum curve very well. Furthermore, the spectral angle mapper (SAM) was used to measure the preservation of spectral features for an entire datacube, wherein the spectral angle between each spectrum of the reconstructed datacube and the spectrum of the original datacube at the same location was calculated. A SAM image was generated, and the SAM values were generally smaller than 0.003 rad with respect to the range from 0 to π. The SAM image did not show any evident spatial pattern due to compression.

A study on the effect of the SAMVQ algorithm on hyperspectral data in the retrieval accuracies of crop chlorophyll content for precision agriculture was presented by Hu et al.[30] A detailed study of the impact of the SAMVQ

compression algorithm on the retrieval of crop chlorophyll content and leaf area index (LAI) in precision agriculture was also reported by Hu et al.[31]

The impact of the HSOCVQ algorithm on mineral mapping products and on retrieval of atmospheric water vapor and canopy liquid water content has been examined elsewhere.[32,33] The influence of the near-lossless compression algorithms on the classification accuracy of forest species using Hyperion data was reported by Dyk et al.[34]

A work reported by Serele et al. [35] evaluated the preservation of spatial features of hyperspectral data after compression using variogram analysis. It was found that there is no apparent difference between both the semivariances and variogram ranges derived from the original data and those derived from the compressed data.

In order to systematically assess the usability of compressed data and the effect of the compression algorithms on hyperspectral data applications, a multidisciplinary user acceptability study was carried out and reported.[36] A total of 11 hyperspectral data users, covering a wide range of remote sensing application areas, participated in the study in two phases. The compressed and original datacubes were evaluated by the users in a double-blind test using their well-defined remote sensing algorithms or products. The users ranked and either accepted or rejected the datacubes based on predefined criteria; this chapter summarizes the work done in this study. Only the evaluation results for the SAMVQ algorithm are addressed, although the evaluation results for the HSOCVQ algorithm are briefly summarized in Section 8.6.

8.2 Double-Blind Test

Double-blind testing was adopted in this study in order to reduce error, self-deception, and bias in the assessment of the impact of compression on hyperspectral data. It is a test where neither the evaluator nor the subject knows which item is the control group and which item is the test group. In this study, an original datacube is the control group. The compressed datacubes are the test group. The subjects are the hyperspectral data users who derived remote sensing products from the datacubes using their algorithms and provided either rankings and acceptance or rejection of the datacubes based on predefined criteria. The evaluators are the people at the contracted company who managed this study and summarized the users' acceptability results.

A set of datacubes was created that included both the compressed datacubes and the original datacube. Each datacube in this set was labeled with a random number. In this chapter, each datacube in this set is referred to as a blind datacube, as neither the users nor the evaluators knew which was a compressed datacube and which was the original datacube. (Blind datacubes

are shown here with their compression status for the sake of reporting the results, as the evaluation study has been completed.)

8.3 Evaluation Criteria

The criteria for acceptability were established prior to the commencement of the users' evaluation. Whenever possible in this study, the products derived from the blind datacubes were assessed and ranked according to their agreement with the ground truth. The effect of data compression was assessed both qualitatively and quantitatively.

Qualitatively, each blind datacube was classified as acceptable or unacceptable according to the adequacy of information needed to derive the products and for making decisions for the specific application. Quantitatively, statistical tests were applied to identify whether the difference between the products' values derived from the blind datacubes and the ground truth is statistically significant. A blind datacube was considered acceptable if the difference was not found to be significant. Different users developed their own quantitative rule to assess the significance, taking into account such things as uncertainty of their mathematical model used to derive the products or the stability of the product.

In this study, an attempt was made to avoid comparing the products derived from the blind datacubes with those derived from the original datacube, whenever possible. When making comparisons to the original datacube, users can focus on minute changes whose significance is not well assessed. Because an original datacube is not exempt from sensor noise and uncertainties introduced during the preprocessing steps (e.g., radiometric calibration and atmospheric correction), there is a propagation of errors into the products derived from the original data.

Another reason to avoid this comparison is because the compression algorithms SAMVQ and HSOCVQ can be set to "near-lossless" mode. In this mode, the error introduced during compression is at a level equal to or lower than the intrinsic noise in the original data, as described in Chapter 5. It is expected that this level of error has a small-to-negligible impact on the quality of the data compared to the errors introduced in preprocessing steps. The experimental results reported by Qian et al.[3] indicate that VQ compression sometimes improves the data quality for derived products because the compression algorithm can be viewed as a nonlinear filter that eliminates artifacts (e.g., salt-and-pepper noise, etc.).

8.4 Evaluation Procedure

Figure 8.1 shows the evaluation procedure used in this study. Eleven hyperspectral data users were selected to cover a wide range of products,

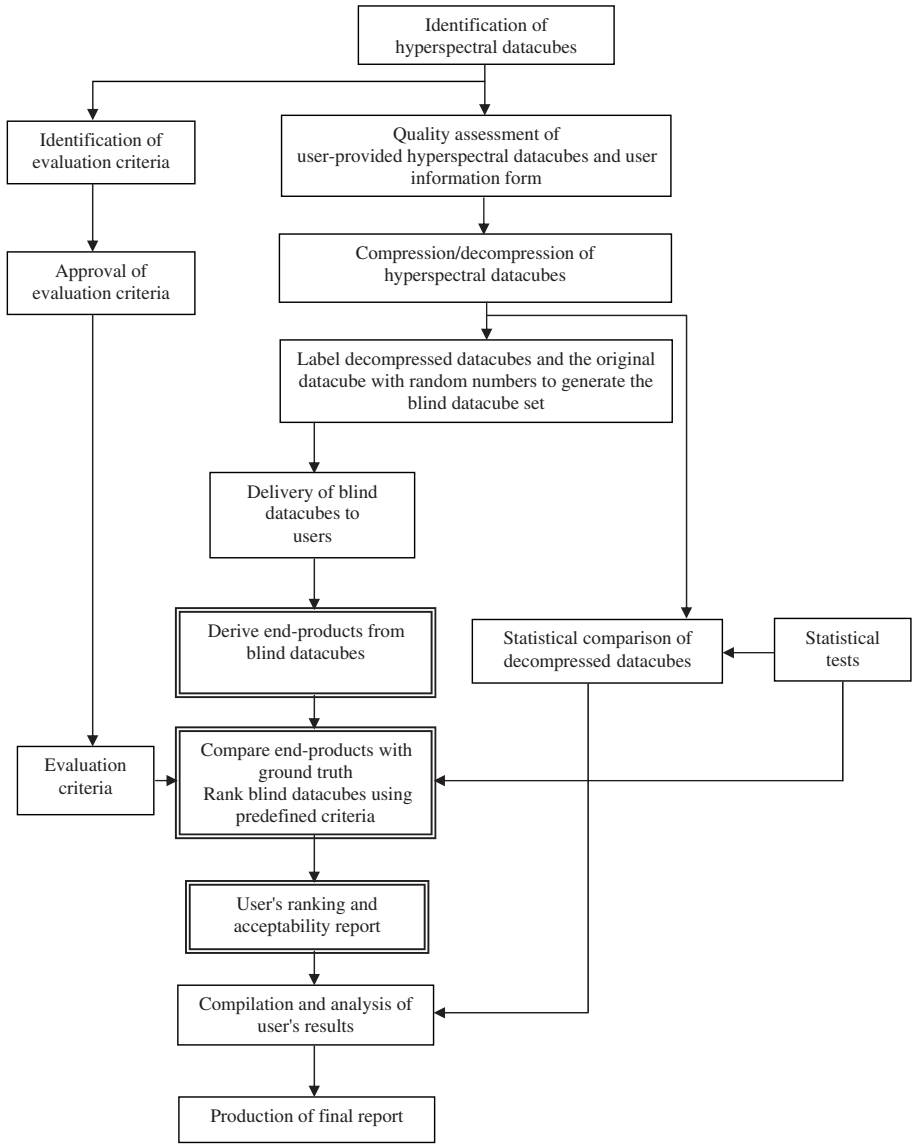


Figure 8.1 Evaluation procedure of the user acceptability study (reprinted from Ref. 36).

application areas, and sensors. Table 8.1 provides detailed information about the users. A form was sent to each of the users to collect all necessary information about the datacube(s) and the product algorithm(s) used in this study. Prior to compression, datacubes were screened using a quality assurance procedure. Each datacube was examined, and anomalies such as negative values and spurious spikes were removed (these anomalies could influence the compression fidelity and ultimately bias the evaluation results).

Table 8.1 Information about the users, datacubes, application area, algorithms, and products (represented from Ref. 36).

User #	Affiliation	Hyperspectral Sensor	Location Data Name	Data Size (Lines × Pixels × Bands)	Data Level	Application Area	Algorithm(s)	Product(s)
1	York University	CASI	Greenbelt farm, Ottawa, Canada	981 × 405 × 72	DN	Precision agriculture	Leaf area index	LAI fraction
2	Tecsult, Inc.	TRWIS-III	Lac-Saint-Jean, Canada	512 × 256 × 247	Radiance	Forest regeneration	Maximum likelihood classification Spectral angle mapping	Classification maps
3	University of Alberta	ASD Spectrometer (Probe-1)	Rock samples in laboratory	200 × 290 × 128	Reflectance	Geology	Spectral unmixing Spectral angle mapping	Fraction maps Classification maps
4	Defence Research and Development Canada – Valcartier (DRDC-V)	SFSI-II	N/A	100 × 446 × 240	Radiance	Target detection	Spectral unmixing	Fraction maps
5	Noranda Inc./ Falconbridge Limited	Probe-1	North-Central Chile	1000 × 512 × 128	Radiance	Mineral exploration	Spectral feature fitting	Fraction maps
6	Satlantic Incorporated	PHILLS2	Atlantic Ocean	606 × 499 × 122	Radiance	Ocean, ship, & wake detection	Support vector machines	Binary location of ship wakes
7	R. J. Burnside & Associates, Limited	TRWIS-III	Cuprite, NV, USA	800 × 200 × 384	Radiance	Mineral exploration	Spectral unmixing	Fraction maps
8	CSIRO - Exploration & Mining (Australia)	Hymap	Mt. Fitton Talc Mines, Australia	1000 × 512 × 126	Radiance	Mineral exploration	Voltron SAMspade	Pseudo-fraction maps Composite classification maps
9	MacDonald Dettwiler and Associates (MDA)	Probe-1	Punta, CA, USA	570 × 512 × 128	Radiance	Target detection	Spectral unmixing Target mensuration	Fraction maps
10	Pacific Forest Centre (PFC)	Hyperion	Great Victoria Watershed District, Canada	801 × 256 × 242	DN (B1)	Forest & species classification	Maximum likelihood classification	Classification maps
11	Canada Centre for Remote Sensing (CCRS)	AVIRIS	Cuprite, NV, USA	512 × 614 × 224	Radiance	Mineral exploration	Spectral unmixing	Fraction maps

Each datacube provided by a user was compressed using both the SAMVQ and HSOCVQ algorithms at compression ratios (CRs) of 10:1, 20:1, and 30:1 (20:1, 30:1, and 50:1 for users who participated in phase I only). The compressed data was then decompressed to produce the reconstructed datacubes for evaluation; note that these datacubes are of the same size and format as the original, but they have undergone compression. The CRs were selected in order to cover reasonable compression performances for limited distortions. The reconstructed datacubes together with the original datacube were labeled with random numbers by an individual who was neither a user nor an evaluator in order to generate a blind datacube set for the purpose of double-blind testing.

In order to examine the quality of the reconstructed datacubes, statistical hypothesis tests were applied to assess whether the compression significantly affected the mean and variance of ROIs provided by the users. The F, T, and Z tests and RMSE were used to examine the significance on a spectral band by spectral band basis. The general conclusion was that the changes found for the mean and variance of ROIs were not significant.

The three double-lined boxes in Fig. 8.1 outline the tasks performed by the users. First, products were derived from the blind datacubes using the user's product algorithm(s). The products were then compared with the ground truth or products derived from the original data to rank and accept or reject the blind datacubes based on predefined criteria and statistical tests. Finally, a user ranking and acceptability report was written and sent to the evaluators. After collecting all of the users' reports, the evaluators compiled and analyzed the users' results, and produced the final report for the study.

8.5 Multidisciplinary Evaluation

A total of 11 users participated in this study. Table 8.1 summarizes the information about the users, type of hyperspectral sensor, location where test data was collected, data processing level, datacube size, application area, algorithms, and products. Among the users, five are commercial companies, two are universities, and four are government entities. They cover a wide range of hyperspectral data application areas, including agriculture, geology, oceanography, forestry, and defence. A total of nine different hyperspectral sensors are covered, including the spaceborne hyperspectral sensor Hyperion. This section briefly describes the evaluation and results for SAMVQ on a user-by-user basis.

8.5.1 Precision agriculture

User #1 (York University) evaluated the impact of the data compression on the derivation of crop leaf area index (LAI) from hyperspectral data. A total of 27 blind datacubes compressed using both SAMVQ and HSOCVQ

generated from three test datacubes were evaluated in two phases. In this chapter, only the evaluation results for SAMVQ on one test datacube carried out in phase II are presented to constrain the length of the chapter. The test datacube was acquired using CASI over the crop fields at the former Greenbelt farm of Agriculture and Agri-Food Canada, Ottawa in the intensive field campaign on June 26, 2001. The CASI sensor was configured for 72 spectral bands with a spectral resolution of 7.6 nm in the wavelength range of 408–947 nm. The spatial resolution is approximately 2 m $\times$ 2 m. The test datacube was in raw digital number (DN). Figure 8.2 shows the datacube color-composite image for three bands. The three bands displayed were used to estimate the crop LAI. The LAI values in the 14 sites (ground truth) were measured by Agriculture and Agri-Food Canada during the field campaign.

Four blind datacubes, including the three compressed/decompressed using SAMVQ and the original (identified as #278, #519, #730, and #866 arbitrarily), were delivered to the user. Because the test datacube is in DN, the blind datacubes were first calibrated using calibration coefficients determined in the laboratory to produce radiance and then converted to reflectance using the CAM5S atmospheric correction software.[37] The georeferencing was performed on the reflectance datacubes. Four LAI images were derived from the reflectance datacubes using the algorithm developed in the work.[38]

Visual examination of the spatial pattern of the LAI images was used to qualitatively assess the impact of data compression. There are two plots in the wheat field, each having very different soil conditions. The wheat in the

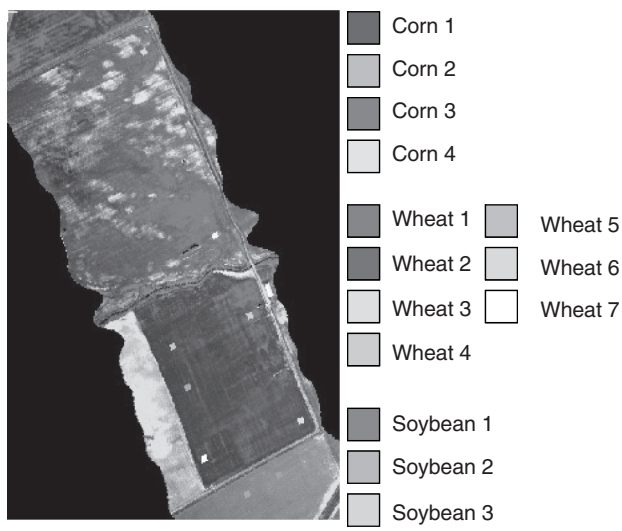


Figure 8.2 The CASI datacube RGB image [band #41 (708 nm) as red, band #24 (580 nm) as green, and band #8 (460 nm) as blue] with 14 sites where the ground truth was collected. The three bands were used to estimate the crop LAI (reprinted from Ref. 36). For a color version of this figure, see Plate 8 in the color plate section of this book.

northeastern area showed more vigor than the wheat in the northwestern area. As a consequence, it is anticipated that the wheat LAI in the northeast is higher than that in the northwest. For each LAI image, if the spatial pattern of the LAI is the same as expected, then the image is acceptable.

The correlation (expressed by the coefficient of determination R^2) between the derived LAI from each blind datacube and the measured LAI (ground truth) was calculated and used to quantitatively assess the impact of data compression. From an application perspective, a blind datacube is considered acceptable if over 90% of the variation ($R^2 \geq 0.9$) in the measured LAI values can be explained by the derived LAI. This criterion is selected based on the accuracy of the LAI retrieved from the original reflectance datacube.[38] Statistical Z-tests[29] were performed to assess whether the mean and variance of the LAI values in each of the 14 ROIs (where the ground truth had been collected) derived from each of the blind datacubes are significantly different from the ground truth. Both the 1 and 5 percent significance levels were used.

Four LAI images derived from the four blind datacubes are shown in Fig. 8.3. The compression ratios corresponding to the blind datacubes are given in the caption for the figure. The spatial patterns for these images are similar; user #1 qualitatively accepted all the blind datacubes based on visual inspection.

The R^2 between the ground truth and the LAI derived from each of the blind datacubes is shown in Fig. 8.4. The absolute and relative RMSE between the ground truth and derived LAI were also calculated and shown in the figure; note that the correlation between the ground truth and derived LAI is quite high for all of the blind datacubes (all of the R^2 are over 0.95). Table 8.2 lists the

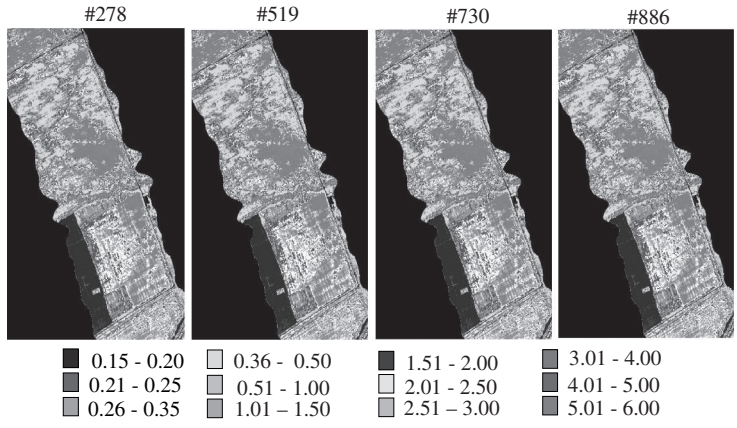


Figure 8.3 The LAI images derived from the four blind datacubes. The compression ratios associated with blind datacubes are as follows: 10:1 (#519), original (#730), 20:1 (#278), and 30:1 (#886) (reprinted from Ref. 36). For a color version of this figure, see Plate 9 in the color plate section of this book.

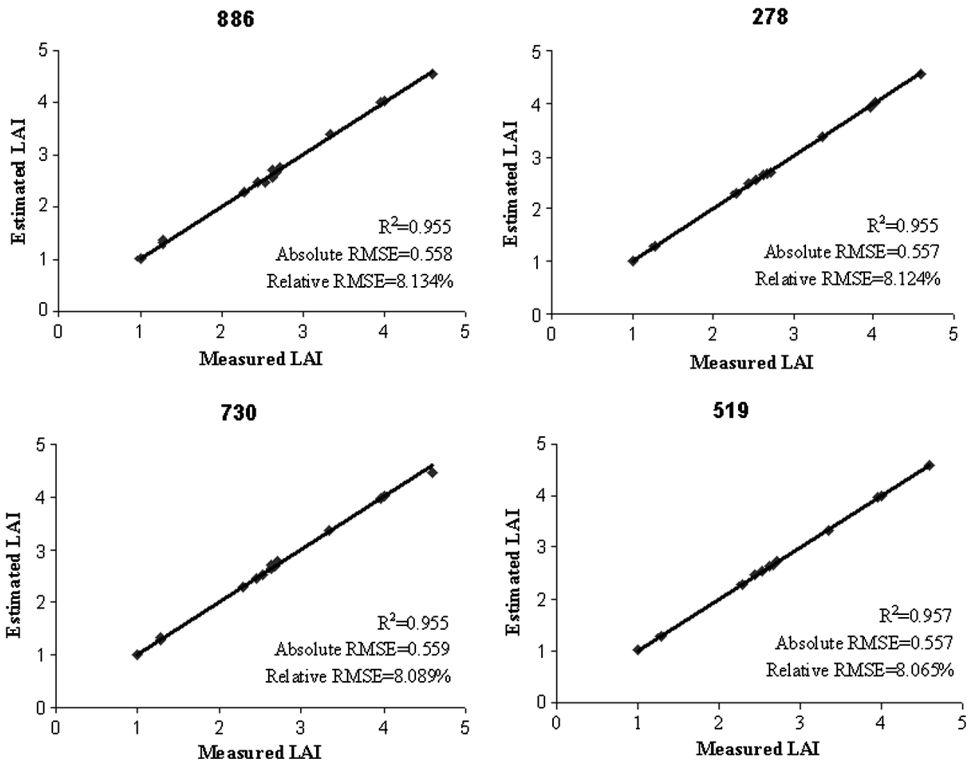


Figure 8.4 The correlation, absolute RMSE, and relative RMSE (%) between the derived LAI from the blind datacubes and the ground truth. The compression ratios associated with blind datacubes are as follows: 20:1 (#278), 10:1 (#519), original (#730), and 30:1 (#886) (reprinted from Ref. 36).

Table 8.2 Evaluation results, ranking and acceptability for user #1 (reprinted from Ref. 36).

Blind Datacube #	Qualitative Acceptable?	Quantitative Measure			Rank	Overall Acceptable?
		R^2	Absolute RMSE	Relative RMSE		
278 (20:1)	Yes	0.955	0.557	8.124%	3	Yes
519 (10:1)	Yes	0.957	0.557	8.065%	1	Yes
730 (Original)	Yes	0.955	0.559	8.089%	2	Yes
886 (30:1)	Yes	0.955	0.558	8.134%	4	Yes

evaluation results, the ranking, and the acceptability for this user. The absolute RMSEs between the ground truth and derived LAI are smaller than 0.56, whereas the measured LAI values (ground truth) vary from 0.87 to 5.3 with a standard deviation of 1.4. The relative RMSEs are all smaller than 8.2%. The experimental results showed that the mean and variance of the LAI values for

all of the 14 sites derived from each of the blind datacubes are not significantly different from the ground truth. In conclusion, user #1 quantitatively accepted all of the blind datacubes based on the predefined criteria. The ranking of the datacubes (evaluated according to agreement with the ground truth) is 10:1, original, 20:1, and 30:1.

It is observed from the quantitative evaluation results that the accuracy of the product LAI derived from the original datacube (#730) is not necessarily better than that derived from the compressed datacubes when they are both compared to the ground truth. The R^2 of the LAI derived from the original is virtually the same as that derived from the compressed data. Its absolute RMSE and relative RMSE are in some cases even worse than those derived from the compressed data.

8.5.2 Forest regeneration

User #2 (Tecsult, Inc.) assessed the impact of data compression on the mapping of forest regeneration ecosystems in a boreal forest environment. The test hyperspectral datacube was acquired using the TRWIS-III sensor on September 2, 2001 at 150-km northwest of Saint Jean Lake in the province of Quebec, Canada. The instrument was configured with a ground pixel size of 3 m $\times$ 7 m and spectral range of 371–2501 nm. The original datacube comprises 384 spectral bands, 256 cross-track pixels, and 512 along-track pixels. Bands associated with atmospheric absorption features or with low SNRs were removed, leaving 247 out of the 384 spectral bands.

Extensive preprocessing was performed on the data, which consisted of registration of the VNIR and the SWIR FPAs, sensor attitude correction (roll, pitch, and yaw), correction for the illumination variation across the FOV, and correction for a slight shift in the band-center position from the expected values and correction of low-response pixels (striping). The datacube was processed to at-sensor radiance.

The product algorithms used were the SAM, the principal component transformation, and the maximum likelihood classification. A postclassification filter was applied to smooth the output maps. The end product was a forest regeneration map sorted by groups of species, density, and tree height.

The qualitative evaluation by this user was based on visual inspection of the classification maps derived from the blind datacubes compared to those from the original datacube. Both spatial extent and spatial distribution were compared. Visual inspection did not reveal any differences between the classification maps.

For the quantitative evaluation, an error matrix was calculated using ground control points (ground truth). The ground control points have roughly uniform areas and have been validated using a standard interpretation of 1/5000 aerial photography. An error matrix was derived for all of the blind datacubes. This user's evaluation results indicated that the

Table 8.3 Average classification accuracy for the maximum likelihood classifier (MLC) and the spectral angle mapper (SAM) algorithm compared to the ground truth, ranking, and acceptability of user #2 (reprinted from Ref. 36).

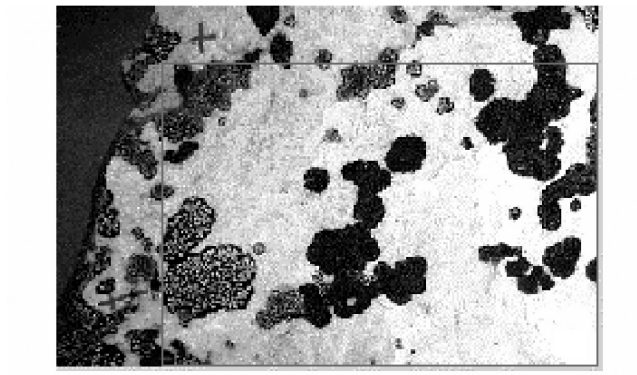
Blind Datacube #	MLC Accuracy	SAM Accuracy	Rank	Acceptable
139 (Original)	71%	66%	3	Yes
938 (20:1)	72%	66%	2	Yes
444 (30:1)	73%	66%	1	Yes
385 (50:1)	72%	65%	3	Yes

compression did not alter the overall classification accuracy, even with large CRs; the evaluation results and ranking are presented in Table 8.3.

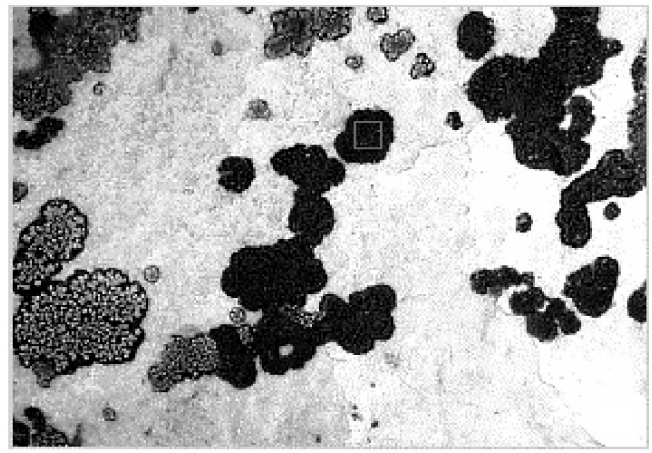
Although the average classification accuracy for all classes is seen to be excellent, one might be wary of the classification accuracy of specific classes. As such, a classification matrix for both omission and commission errors was calculated, but results show no specific tendency. Specific CRs might have some classes that are worse than others; however, no single class appeared to be systematically affected by the compression. On average, commission and omission errors for the compressed datacubes were very similar to those for the original datacube, which itself sometimes presented the largest omission or commission error for a given class. Based on the classification accuracy, this user found that blind datacube #444 (30:1) produced the best results and was ranked as #1. Blind datacube #938 (20:1) was ranked as #2, blind datacube #139 (original), and #385 (50:1) were both ranked as #3.

8.5.3 Geology

User #3 (University of Alberta) assessed the impact of data compression on geological rock mapping. The datacube was acquired using an ASD^{TM} spectrometer in a laboratory by scanning a rectangular region of a rock (quartzite) face (see Fig. 8.5), which is partially coated with different lichens (predominantly green and black crustose lichen). Lichens are common coatings on exposed rock surfaces, and mapping rock units partially covered by lichens is of particular interest when performing rock unit mapping. The ASD spectrometer data was resampled to simulate the Probe-1 sensor, which covers a spectral range from 436–2501 nm with 128 bands. The water vapor feature from 1796–1996 nm was masked. The benefit of using this datacube is that the datacube was collected under a well-controlled laboratory environment with known endmembers (e.g., rocks and lichen species) and "ground truth." The products derived from the original datacube were the ground truths in this case and used as the metrics to assess the impact of data compression. The spectrometer produced reflectance data. No calibration or atmospheric correction preprocessing steps were required. The evaluation using this datacube assessed the impact of only one error source, i.e., the error



(a)



(b)

Figure 8.5 (a) Digital photo of rock sample, with a (b) subregion marked by a rectangle (reprinted from Ref. 36).

introduced by the compression, as no preprocessing was applied to the datacube. This evaluation provides high confidence that the results are due solely to the compression.

There were three major classes—quartzite, green lichen, and black lichen—in the scene of the datacube. A total of 11 endmembers (5 for quartzite, 2 for green lichen, 3 for black lichen, and 1 for shadow) were extracted from the datacube; they were a complete set of spectra that could reconstruct the datacube through linear spectral mixing with minimal error.

Two algorithms were used to assess the impact of data compression: the spectral angle mapper and linear spectral unmixing (LSU). The former is sensitive only to the spectral shape, and the latter is sensitive to both the spectral shape and the amplitude. The color-composition SAM images and

Table 8.4 Quantitative evaluation results, ranking, and the acceptab lity for user #3 (reprinted from Ref. 36).

Blind Datacube	Correlation of Fraction Image Derived from Compressed and Original Datacubes			Correlation of SAM Image Derived from Compressed and Original Datacubes			Classification Accuracy (%)		Rank	Accept-able?
	Quartzite	Green Lichen	Black Lichen	Quartzite	Green Lichen	Black Lichen	LSU	SAM		
180 (10:1)	0.9984	0.9917	0.9982	0.9999	0.9999	0.9999	98.69	99.98	2	Yes
927 (20:1)	0.9987	0.9932	0.9906	0.9999	0.9999	0.9999	98.81	99.98	1	Yes
988 (30:1)	0.9979	0.9908	0.9867	0.9999	0.9999	0.9999	98.51	99.84	3	Yes

fraction images derived from the blind datacubes and those derived from the original datacube were visually identical and qualitatively acceptable.

In order to quantitatively evaluate the effect of data compression, the correlation between the fraction images derived from each of the blind datacubes and from the original datacube was calculated. The correlation between the SAM images derived from each of the blind datacubes and from the original datacube was also calculated; Table 8.4 lists the quantitative evaluation results. It can be seen that the fraction images and SAM images derived from each of the blind datacubes have a very high correlation with those derived from the original datacube (ground truth). The coefficients are above 0.98.

An unsupervised classification was performed on both the fraction images and the SAM images to establish the optimal thresholds for mapping the distribution of quartzite, green lichen, and black lichen in the datacube. The general distribution of the three classes on the classification maps obtained from the fraction images and SAM images derived from each of the blind datacubes were all very similar to the distribution for the original datacube. A confusion matrix was calculated between each map generated from a blind datacube and from the original datacube. The overall classification accuracies for the two products are listed in Table 8.4—all better than 98%. This user considered all of the compressed datacubes acceptable, as the user believes that an overall classification error of less than 5% is usually acceptable in applications of hyperspectral data. Other sources of error (e.g., instrument stability, calibration, atmospheric correction, etc.) can contribute even larger errors to the final products.

8.5.4 Military target detection

User #4 (Defence Research and Development Canada–Valcartier) evaluated the impact of data compression on military target detection. Figure 8.6 shows the synthetic targets present in the test datacube: seven pieces of awning with varying sizes ranging from 12 m $\times$ 12m to 0.2 m $\times$ 0.2 m, four pieces of

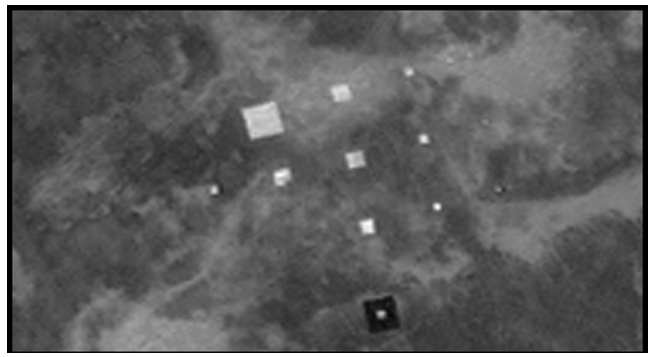


Figure 8.6 Synthetic targets present in the datacube (reprinted from Ref. 36)

polythene, four pieces of white tarp, and four pieces of white cotton with varying sizes ranging from 6 m to 0.5 m were deployed. In addition, a 3 m × 3 m piece of white tarp was placed on a large vinyl turf mat (11 m × 14 m). The test datacube was acquired using the Short-Wave Infrared Full-Spectrum Imager II (SFSI-II) flown on June 7, 2002 at an altitude of 2900 m with a ground sample size of 3.5 m × 3.5 m and 240 spectral bands between 1200–2450 nm with a band interval of 5 nm. The sky was clear with a few cirrus clouds.

The raw data was first preprocessed to remove periodic noise, dark current, slit curvature (smile), and keystone. A vicarious calibration was then performed using calibration coefficients derived from a previous SFSI-II survey to convert the raw data into radiance. Before the radiance datacube was sent for compression/decompression, it was processed to remove system noise.

Nine endmembers were selected that correspond to the five materials of the synthetic targets and the four ground features (forest, gravel road, sand, and grass). This endmember selection helped reduce the average unmixing error. Nine ROIs were selected and used to extract the endmember spectra from each blind datacube. Constrained spectral unmixing was performed with the nine endmembers using the ISDAS unmixing tool.[39]

This user used the detection results obtained from the original datacube as the benchmark to evaluate the impact of compression. They were viewed as equivalent to the ground truth because the targets in the test datacube were synthetic and their spatial and spectral features were well understood and validated. The evaluation was performed on a ROI-by-ROI basis.

For the qualitative evaluation, the following two criteria were used to assess the impact (one point was scored if a criterion was met):

1. All targets seen in the fraction images derived from the original datacube must be seen in the fraction images derived from the blind datacubes, and

2. No targets other than the ones seen in the fraction images derived from the original must be present in the fraction images derived from the blind datacubes.

In the quantitative evaluation, a T-test was performed to determine whether the distribution of the fraction images derived from the blind datacubes is significantly different from that of the fraction images derived from the original datacube. One point is scored if the difference is not significant. The percent standard error (%SE) was used to measure the relative average deviation of the fraction images derived from the blind datacubes; it is defined as follows:

$$\%SE = \frac{\frac{1}{n}\sqrt{\sum_{i\ 1}^{n}\left(f_i \quad \hat{f}_i\right)^2}}{\overline{f}} \times 100\%, \tag{8.1}$$

where f_i is the fraction of an EM for a pixel in a ROI derived from the original datacube, $\hat{f}_i$ is the fraction of the EM for the same pixel derived from a blind datacube, $\overline{f}$ is the mean fraction of the EM for the ROI of the original datacube, and n is the number of pixels in the ROI. One point is scored if the %SE of a ROI is less than 5%.

Table 8.5 lists the scores for the ROIs of the five targets based on the qualitative and quantitative criteria defined above; the maximum score for each ROI is 4. Blind datacube #119 (10:1) received 18 points out of 20 and was ranked as #1, whereas datacubes #721 and #172 received very close scores (14 and 13). This user accepted all three compressed datacubes.

8.5.5 Mineral exploration 1

User #5 (Noranda/Falconbridge) evaluated the impact of data compression on the identification of minerals in mineral exploration applications. The test hyperspectral datacube was acquired using ESSI Probe-1 in a mineral area in north-central Chile. The data was acquired on March 28, 1999 at 15:46 GMT. The spatial resolution for this datacube is 10 m × 10 m and covers a spectral

Table 8.5 Score for each ROI, ranking, and acceptability for user #4 (reprinted from Ref. 36).

Blind Datacube	ROI 1 (awning) Size 71	ROI 2 (polythene) Size 29	ROI 3 (plastic tarp) Size 29	ROI 4 (cotton) Size 28	ROI 5 (vinyl mat) Size 80	Total Score per Datacube	Rank	Acceptable?
119 (10:1)	3	4	3	4	4	18 (90%)	1	Yes
172 (30:1)	3	2	2	3	3	13 (65%)	3	Yes
721 (20:1)	3	3	2	3	3	14 (70%)	2	Yes

range from 438–2504 nm with 128 spectral bands. The datacube had been calibrated to at-sensor radiance. The minerals goethite, hematite, alunite, kaolinite, and illite were contained in the test data and were used to assess the blind datacubes.

A blind datacube was first spectrally subset into a VNIR subcube (bands 1–31 and 39–45) and a SWIR subcube (bands 100–124). The VNIR subcube was used to map goethite and hematite, whereas the SWIR subcube was used to map alunite, kaolinite, and illite. An internal average relative reflectance calibration was applied to normalize each datacube to the scene average spectrum. Reference spectra extracted from the JPL and USGS mineral spectral libraries were used as the "ground truth." The spectral feature fitting (SFF) algorithm was applied to compare the fit of the spectra of a blind datacube to the corresponding reference spectra at each selected wavelength in a least squares sense. The SFF is an absorption-feature-based methodology.

The following qualitative criteria were used to broadly classify each of the blind datacubes as acceptable, marginal, or unacceptable in terms of their suitability for mineral exploration:

1. Presence of system artifacts (e.g., striping) in the datacubes,
2. Texture of the product images (e.g., smooth, speckly, or blocky),
3. Spectral quality, and
4. Visual comparison of the products derived from the blind datacubes to those derived from the original datacube.

All of the blind datacubes were considered qualitatively acceptable.

Two quantitative evaluations were used to rank the products generated from the blind datacubes. They are referred to as "band math evaluation" and "mean and standard deviation evaluation." The former evaluation method assesses each blind datacube by comparing the number of inconsistencies in the derived products (i.e., mineral fraction planes) against those derived from the original datacube. The number of inconsistencies is the sum of the "committed" and "omitted" pixels in the ROI compared to the mask in the original datacube for the mineral being evaluated. The latter evaluation method compares the mean and standard deviation for each ROI against those of the original datacube. Table 8.6 summarizes the evaluation results of this user. All of the blind datacubes were considered to

Table 8.6 Ranking and acceptability for user #5 (reprinted from Ref. 36).

Blind Datacube #	Qualitative Acceptable	Quantitative Ranking	Overall Acceptable
159 (original)	Yes	1	Yes
638 (30:1)	Yes	4	Yes
957 (20:1)	Yes	3	Yes
980 (10:1)	Yes	2	Yes

be acceptable; their ranking based on the overall quantitative score is original, 10:1, 20:1, and 30:1.

8.5.6 Ocean ship and wake detection

User #6 (Satlantic, Inc.) evaluated the impact of compression on hyperspectral data applications for ocean ship wake detection. Ship wakes are bright patterns arising from the ship's movement and propellers. They are attributed to the reflectance of the solar irradiance and to the concentration of air bubbles in the water. Three hyperspectral datacubes of size 499 × 606 pixels were used. They were acquired using PHILLS2, which was flown aboard an airplane during July and August of 2001, at an altitude of roughly 9 km with a GSD of approximately 9 m.

Each test datacube contained the spectra of one vessel, the spectra of its associated wake, and the spectra of water (background). One of them was used to obtain a total of 109 training spectra (9 wake signatures and 100 water signatures) for training support vector machines (SVMs). The trained SVM was then applied to the three test datacubes to generate three 2D classification maps, which were used as the benchmarks to evaluate the blind datacubes. Four blind datacubes were created from each of the three test datacubes. This user evaluated a total of 12 blind datacubes. The classification was performed on region averages for each blind datacube that were either calculated on a moving 10 × 10 pixel window centered on each pixel (moving average classification, or MAC), or nonoverlapping regions of 10 × 10 pixels (average classification, or AC).

The effect of compression was assessed by comparing the wake-pixel classification map derived from a blind datacube with that derived from the original datacube because the ground truth was not available. If more than 80% of the ship wake pixels of a blind datacube were properly identified, the blind datacube was then considered acceptable. If 40–80% of the ship-wake pixels were properly identified, the blind datacube was considered marginally acceptable.

Table 8.7 lists the classification accuracy of wake pixels for both the MAC and the AC classification. It can be seen that the percentage is always better than 80% using the MAC and better than 90% using AC. The AC approach yielded better results, sometimes as good as 100%. Overall statistics were excellent because less than 0.5% of the pixels were attributed to the wrong class for all of the classified spectra of all of the blind datacubes. However, for test datacube 006/2100, the MAC classification methodology introduced 10% error, even for the same original datacube (#663). The difference between the products derived from the two identical original datacubes (the one that the user archived and the one in the blind datacubes) indicates that the product algorithm is not stable or repeatable.

Table 8.7 Evaluation results, ranking, and acceptability for user #6 (reprinted from Ref. 36).

Test Datacube	Blind Datacube	Wake Detection		Ranking	Acceptability
		MAC	AC		
006/2100	568 (50:1)	80%	90%	3	Acceptable
	623 (30:1)	80%	90%	3	Acceptable
	663 (original)	90%	100%	1	Acceptable
	751 (20:1)	83%	90%	2	Acceptable
008/6000	234 (original)	100%	100%	1	Acceptable
	239 (30:1)	95%	100%	3	Acceptable
	635 (50:1)	95%	100%	3	Acceptable
	674 (20:1)	96%	100%	2	Acceptable
008/6660	177 (50:1)	96%	100%	2	Acceptable
	240 (30:1)	96%	100%	2	Acceptable
	383 (original)	100%	100%	1	Acceptable
	564 (20:1)	96%	100%	2	Acceptable

A statistical test analysis was performed on ROIs of both ship wakes and non-ship wakes (i.e., background). At the 5% significance level, the means and variances of ship wake ROIs for all of the blind datacubes were not significantly different from the means and variances for the original datacubes. Using the predefined criterion, this user considered all of the blind datacubes to be acceptable.

8.5.7 Mineral exploration 2

User #7 (Burnside & Associates, Limited) evaluated the impact of compression on spectral unmixing for mineral exploration applications. The test datacube was acquired using a TRWIS-III hyperspectral sensor at the renowned Cuprite site in Nevada, USA covering the spectral range from 400–2500 nm in 384 contiguous spectral bands. A subset of the acquired datacubes consisting of 200 pixels cross-track by 800 lines along-track of radiance data was used.

The blind datacubes were atmospherically corrected before the remote sensing algorithm was applied. Most of the subsequent processing was done with ENVI, and the ISDAS software[39] was used for automatic endmember extraction and unmixing. Up to thirty endmembers were extracted from the blind datacubes. The SWIR region of the spectra was especially important as it contains most of the distinguishing features for minerals. The end products are fraction maps (from 0–100%).

The qualitative evaluation was simply performed by the visual inspection of the fraction maps. A comparison with the original datacube was made, and the fraction maps for major mineral constituents derived from the blind datacubes were examined. They were not significantly different from those derived from the original datacube.

The quantitative evaluation was performed on four selected ROIs. These regions were predominantly composed of the following endmembers: buddingtonite, kalunite, chalcedony, and halloysite. This user evaluated the impact of compression strictly based on the qualitative results. No statistical tests were performed, and thus no quantitative results are available. This user considered the datacube compressed at 20:1 to be acceptable, the datacube compressed at 50:1 to be marginally acceptable, and the datacube compressed at 30:1 to be unacceptable.

8.5.8 Mineral exploration 3

User #8 (CSIRO/Exploration & Mining) evaluated the impact of compression on mineral mapping of mica and dolomite using hyperspectral data. The test datacube was a spatial subset of a HyMap survey flown for CSIRO in 2000 over the Mt. Fitton talc mines of southern Australia. It consists of 1000 lines $\times$ 512 pixels $\times$ 126 bands at a 5-m spatial resolution with spectral coverage from 400–2500 nm; it has been calibrated to at-sensor radiance with a 12-bit data resolution. This location was selected because the geology of the area is well documented by field studies and the evaluation of field-, air-, and spaceborne instruments undertaken by the user. The two most-spatially important minerals in the scene, mica and dolomite, were studied.

Four blind datacubes (identified as #148, #389, #486, and #637) were processed using the same methodology as was applied to the original datacube. Atmospheric correction was carried out on each blind datacube using HyCorr.[44] A water-vapor fraction image was generated and used to examine compression effects at the atmospheric correction stage; an evaluation of the water vapor statistics for selected pixels was also carried out to provide a quantitative comparison.

The endmembers of minerals mica and dolomite were extracted using Voltron, a fast and automated mineral-mapping algorithm developed by the user. The spatial location and spectral characterization of mica endmembers were analyzed. The shape and depth of the spectra from the same location of the mica endmembers ROI in each blind datacube were compared with those of the spectra from the same location in the original data to assess the effects of the compression algorithm on endmember spectra. Mineral mapping was carried out on two levels: as a composite mineral map using Voltron, and on an individual mineral basis for mica and dolomite using SAMspade, a SAM-like algorithm.

The products described earlier were created and allocated a score to represent whether the effects of the compression on that particular product were acceptable, marginal, or unacceptable. These scores, tallied in Table 8.8, convey the overall user-acceptability ranking of the blind datacubes. The datacube #637 (10:1) obtained the maximum score of 12, the same as datacube #486 (original), and was considered acceptable. Datacube #148 (20:1) obtained a score of 6 and was considered marginally

Table 8.8 Score, ranking, and acceptability for user #8 (reprinted from Ref. 36).

Evaluation Method	Blind datacubes #			
	148 (20:1)	**389 (30:1)**	**486 (Original)**	**637 (10:1)**
Water vapor image	0	0	2	2
EM classes	1	0	2	2
EM spectra	2	1	2	2
Composite mineral map	1	0	2	2
Thematic mica map	1	1	2	2
Thematic dolomite map	1	0	2	2
Total Score	6	2	12	12
Ranking	2	3	1	1
Overall Acceptability	Marginal	Unacceptable	Acceptable	Acceptable

Scores: 2 = acceptable, 1 = marginal, and 0 = unacceptable.

acceptable. Datacube #389 (30:1) obtained a score of 2 and was considered unacceptable.

8.5.9 Civilian target detection

User #9 (MacDonald Dettwiler and Associates) evaluated the impact of data compression on civilian target detection. The hyperspectral data was acquired using Probe-1 covering a wavelength range from 440–2507 nm with a ground sampling distance of 5 m over Santa Barbara, California on Aug. 30, 1998. The size of the test datacube is 512 pixels $\times$ 570 pixels $\times$ 128 bands, which is a subset of a flight line. The test datacube contains three targets (terra cotta roofing material, industrial oil tanks, and vehicles) and seven backgrounds: ground, sand, healthy vegetation, asphalt, concrete, shadow, and dirt road.

The test datacube was calibrated to radiance before being sent for compression/decompression. The ground truth for this site was not available. In this evaluation, the results obtained from the original datacube were used to assess the results from the blind datacubes. Because the results obtained from the original datacube had not been validated against the ground truth, this user considered a 20% difference in detection or false alarm rate of the blind datacubes acceptable.

The pixels in the ROIs were used to define the endmember for each target. The adaptive spectral unmixing tool provided by the HypOT suite was used to unmix both the original and each blind datacube. After unmixing a datacube, the fraction images for each endmember were examined for qualitative evaluation. ENVI's rule classifier was used to generate classification maps from the fraction images. Each pixel was classified based on the maximum EM fraction over a prespecified threshold. Scatter plots for target endmembers were used to assess the correlation between the fraction images for the original and blind datacubes. Target

fraction images were examined closely for quantitative variations in target area, detection, and false alarm rates.

Classification maps generated from the fraction images of the blind datacubes were similar to those of the original datacube, which indicated consistent unmixing results for the background endmembers. By generating scatter plots comparing the unmixing results for the original datacube and the blind datacubes for the target endmembers, the terra cotta and industrial oil tank endmembers showed good organization along the diagonal (above the 0.25 detection limit) and good probability of detection (above 80%). For the terra cotta and the industrial oil tank, a target detection and mensuration algorithm was used to identify contiguous pixels and return an area estimate in square meters. For the vehicle target, the estimates were measured from fraction values for single pixels.

Based on the results of the rule classification, scatter plots, area estimates, and probability of detection, this user considered the datacube compressed at 10:1 acceptable and the datacubes compressed at 20:1 and 30:1 unacceptable.

8.5.10 Forest species classification

User #10 (Pacific Forest Centre) evaluated the impact of compression on classification accuracy for forestry inventory applications. The test datacube was acquired using Hyperion aboard NASA's EO-1 satellite over the Greater Victoria Watershed District (GVWD) on September 10, 2001. The size of the datacube is 256 pixels wide by 6460 lines with 242 spectral bands; it has been processed to level 1b. In this study, a subset extracted from the center of the scene with the full width by 801 lines by 195 bands was used. The 195 bands cover a spectral range of 438–2396 nm at an average FWHM of 10 nm.

The classification results for the original datacube were used as the benchmark to evaluate the impact of compression. Prior to classification, the datacube was "cleaned" using an algorithm called BAD PIXEL CORR (BPC)[40] to remove abnormal pixels and stripes. The cleaned datacube was transformed using the forward MNF transform. Eigenchannels 2–12 were then used in the species classification; the first eigenvalue channel was excluded as it contained a gradient due to residual smile effects and radiometric errors across the detector array of Hyperion. A supervised classification was performed using two-thirds of the pixels for calibration (training) and the remaining one-third for validation (check). The input classes consisted of seven nonforest classes and ten forest classes of five species of varying densities and ages.[34] The 17 input classes were aggregated into ten final classes. Confusion matrices for each classification were generated and compared. The average and overall accuracies of both the nonaggregated and aggregated classes were recorded for both the training and check data. The classification accuracies of the original datacube were 93.4% (training) and

89.7% (check) for the aggregated classes, and 88.1% (training) and 80.4% (check) for all 17 classes.

The compressed/decompressed datacubes at CRs of 10:1 and 20:1 were evaluated by this user. The classification accuracies of the 10:1 compressed datacube were 92.8% (training) and 89.2% (check) for the aggregated classes, and 87.6% and 79.2% for all 17 classes. They reduced between 0.5–1.2% compared to those obtained from the original datacube.

It should be noted that two different data "cleaners" were used in this evaluation due to a drop off in the experimental procedure. They introduced an uncertainty in classification accuracy of up to 0.6%. The BPC[40] was used to remove negative values and stripes in the original datacube by the user before performing the classification on the original data. It identifies bad pixels and lines and removes them using the average of two spatial neighboring pixels. Any negative values are set to zero. The datacube sent to the evaluator for compression/decompression had not undergone the BPC. In order to reduce the effect of abnormal pixels on compression performance, the evaluator used another algorithm called "remove-negative-spikes" (RNS)[10] to remove the negative values and spikes before compression. The RNS algorithm was set to replace the spikes and negative values by interpolation of the spectral neighboring values. It has been found that the difference between the classification accuracies obtained from the original datacube "cleaned" using the two different data cleaners was 0.4% on average. This difference made a considerable contribution to the reduction of between 0.5% and 1.2% in classification accuracy for the compressed datacubes. This user considered the 10:1 compressed datacube acceptable and the 20:1 compressed datacube unacceptable.

8.5.11 Endmember extraction in mineral exploration

User #11 (Canada Centre for Remote Sensing) evaluated the impact of compression on endmember extraction and spectral unmixing in mineral exploration applications. The test datacube was collected on June 12, 1996 using AVIRIS at an approximately 20-m ground resolution from an ER-2 aircraft, with a spatial size of 512 lines by 614 pixels by 224 spectral bands, each about 10 nm wide, in the 400-nm to 2500-nm wavelength range. The site lies within the Cuprite mining district of Nevada, USA. This site has been used as a test area for mineral mapping in hyperspectral remote sensing for many years.

The test datacube was calibrated to radiance before the evaluation. The original datacube was converted to surface reflectance using a procedure based on a look-up table approach with tunable breakpoints.[41] Prior to the conversion to reflectance, the wavelengths covering the strong atmospheric water absorption regions at 1380 nm and 1870 nm were eliminated due to the dominance of noise in these regions. For the same reason, the first six bands

and the last five bands were also excluded, resulting in a reduction in wavelength coverage to between 428 nm and 2458 nm.

Endmembers required for the spectral unmixing were extracted from the original datacube using an automated method, iterative error analysis (IEA). Once the endmembers were extracted, the datacube was unmixed with the endmembers using a constrained linear technique.[42,43] Fraction images corresponding to each of the endmembers were created.

This user evaluated three Cuprite datacubes compressed using the HSOCVQ algorithm.[33] Only one datacube compressed using SAMVQ at a CR of 20:1 has been evaluated.

In order to evaluate the impact of compression, the SAM and the average percent-relative absolute difference (APRAD) were used to measure the spectral variations of the endmembers extracted from the original and compressed datacubes. APRAD is defined as follows:

$$APRAD = \frac{1}{n_b}\sum_{b\ 1}^{n_b} PRAD(b), \tag{8.2}$$

$$PRAD(b) = 100\frac{\sqrt{(em_O(b) \quad em_D(b))^2}}{em_O(b)}, \tag{8.3}$$

where $em_O(b)$ and $em_D(b)$ is the endmember reflectance in band b of the original datacube and of the decompressed datacube, respectively. The root mean square error (RMSE) between the fraction images derived from the original datacube and those derived from the decompressed datacube was calculated to measure the impact of compression on the fraction images.

Table 8.9 lists the results for five endmembers, each of which corresponds to a unique mineral in the scene. Figure 8.7 shows the reflectance spectra of the endmembers extracted from the original and from the decompressed datacube. The results indicate that the difference between the endmember spectra derived from the original datacube and from the decompressed datacube is insignificant.

Table 8.9 SAM and APRAD between the endmember spectra extracted from the original Cuprite and the compressed datacube (20:1), and the RMSE between their corresponding fraction maps (reprinted from Ref. 36).

EM	SAM(Radian)	APRAD(%)	Fraction Map RMSE(0.0 1.0)
Chalcedony	0.004	0.277	0.007
Alunite	0.007	0.769	0.020
Kaolinite	0.008	0.930	0.044
Dickite	0.015	2.251	0.028
Montmorillonite	0.010	1.463	0.046

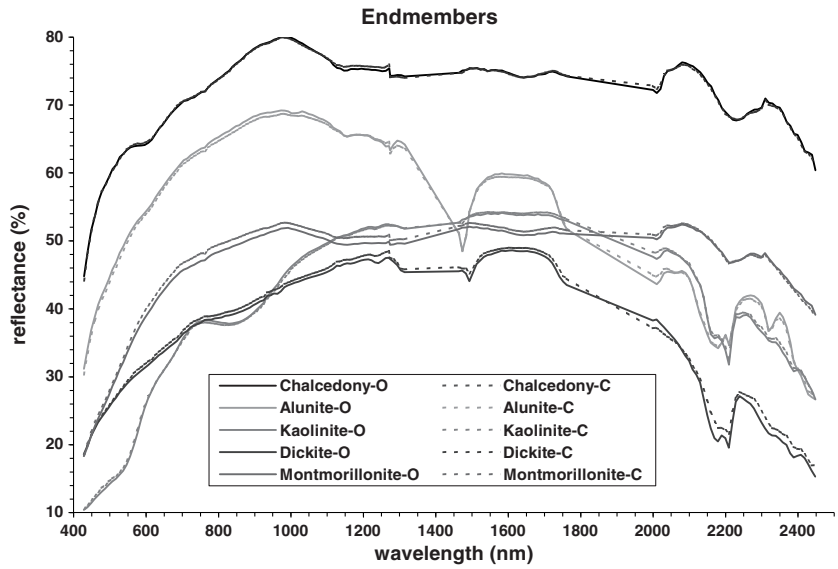


Figure 8.7 Reflectance spectra of the endmembers extracted from the original Cuprite datacube (solid line) and from the compressed datacube (dotted line) (reprinted from Ref. 36).

8.6 Overall Assessment Result and Ranking

In order to reduce the datarate and volume aboard a hyperspectral satellite, the author has developed two real-time multidimensional data compression techniques: SAMVQ[12,21,24] and HSOCVQ.[13,14,21] A multidisciplinary user acceptability study has been carried out to evaluate the quality of the compressed data using these techniques and the impact of the compression on hyperspectral data applications. Double-blind testing was used to reduce error, self-deception, and bias in the assessment of the effect. Sections 8.2–8.5 summarize the work done in the disciplinary user acceptability study.

A total of 11 hyperspectral data users participated in this study. Among them, five are commercial companies, two are universities, and four are government users. They covered a wide range of hyperspectral data application areas, including agriculture, geology, oceanography, forestry, and military target detection, and a wide range of remote sensing algorithms and products. A total of nine different hyperspectral sensors were used, including the spaceborne hyperspectral sensor Hyperion.

This study attempted, whenever possible, to avoid comparing the products derived from the blind datacubes with those derived from the original datacube because, when comparing to the original datacube, users can focus on minute changes whose significance is not well assessed. Because an original datacube is not exempt from sensor noise and uncertainties introduced during the calibration and atmospheric correction preprocessing

steps, there is a propagation of errors in the products derived from the original data. Four out of the eleven users (i.e., users #1–4) had the ground truth available and used it as the metric to assess the products derived from the blind datacubes. They qualitatively and quantitatively accepted all of the compressed datacubes at CRs of 10:1 to 50:1 using both SAMVQ and HSOCVQ because they provided the same amount of information as the original for their applications. Users #5–11 did not have the ground truth available; they evaluated the effect of compression by comparing the products derived from the blind datacubes with those derived from the original datacube. Two of them (users #5 and #6) accepted all of the compressed datacubes evaluated, whereas the remaining users accepted or marginally accepted the compressed datacubes. These users rejected 11 datacubes out of the 90 compressed datacubes evaluated.

Table 8.10 summarizes the evaluation results for this study. For SAMVQ, a total of 7 blind datacubes at a compression ratio of 10:1 were evaluated. The users accepted them all. A total of 17 blind datacubes at a compression ratio of 20:1 were evaluated. The users accepted 14 of them, marginally accepted 1, and rejected 2. A total of 15 blind datacubes at a compression ratio of 30:1 were evaluated. The users accepted 12 of them and rejected 3. A total of 9 blind datacubes at a compression ratio of 50:1 were evaluated. The users accepted 8 of them and marginally accepted 1.

For HSOCVQ,

- A total of seven blind datacubes at a CR of 10:1 were evaluated: users accepted five of them and rejected two.
- A total of 16 blind datacubes at a CR of 20:1 were evaluated: users accepted 14 of them and rejected two.
- A total of ten blind datacubes at a CR of 30:1 were evaluated: users accepted eight of them, marginally accepted one, and rejected one.
- A total of nine blind datacubes at a CR of 50:1 were evaluated: users accepted eight of them and marginally accepted one.

Users #1 and #2 observed that the accuracy of the products derived from the original datacube is not necessarily better than that of the products derived from the compressed datacubes. Both of them did not rank the original datacube as the first out of the four blind datacubes for SAMVQ based on agreement with the ground truth. These evaluation results prove that an original datacube is not exempt from the sensor noise and uncertainties in the preprocessing steps (i.e., calibration and atmospheric correction, etc.). They all contribute additional noise (or errors) to the products derived from the original data. Because the SAMVQ compression algorithm can be adjusted to introduce compression noise at a level consistent with the intrinsic noise of the original data, it is expected that this level of noise has a small to negligible impact on the data quality compared to the noise introduced in other preprocessing steps.

Table 8.10 Summary of user acceptability (reprinted from Ref. 36).

User #	Application Area	Product(s)	Compared to Ground Truth?	SAMVQ				HSOCVQ				Comments
				10:1	20:1	30:1	50:1	10:1	20:1	30:1	50:1	
1	Precision agriculture	LAI fraction	Yes		A A A A	A A A A	A A A		A A A A	A A A	A A A	
2	Forest regeneration	Classification maps	Yes		A	A	A		A	A	A	Participated Phase I only
3	Geology	Fraction map Classification map	Yes		A A	A A	A A		A A	A	A	
4	Target detection	Fraction map	Yes	A	A	A		A	A			
5	Mineral exploration	Fraction map	No	A	A	A		A	A			
6	Ocean, ship, & wake detection	Binary location of ship wakes	No		A A A	A A A	A A A		A A A	A A A	A A A	Participated Phase I only
7	Mineral exploration	Fraction map	No		A	N	M		A	N	N	Participated Phase I only
8	Mineral exploration	Fraction map Composite classification map	No	A	M	N		N	N			
9	Target detection	Fraction map	No	A	N	N		N	N			
10	Forest & species classification	Classification map	No	A	N							Not double blinded
11	Mineral exploration	Fraction map	No		A			A	A	M		
Percentage of Acceptable (%)				100	82.4	80.0	88.9	71.4	87.5	80.0	88.9	
Percentage of Marginal Acceptable (%)				0.0	5.8	0.0	11.1	0.0	0.0	10.0	0.0	
Percentage of Unacceptable (%)				0.0	11.8	20.0	0.0	28.6	12.5	10.0	11.1	

A: Acceptable M: Marginal Acceptable N: Not Acceptable Shade: Not Applicable

User #1 ranked the datacube with the CR of 10:1 first, the original second, the 20:1 third, and the 30:1 fourth. User #2 ranked the datacube with the CR of 30:1 first, the 20:1 second, and both the original and the 50:1 third.

These ranking results were not surprising when double-blind testing was performed and the derived products were assessed by comparing them with the ground truth because both self-deception and bias were eliminated. The ranking is probably due to the following four reasons: (1) Both the original data and the ground truth are not 100% accurate because they contain a certain degree of error upon collection (the original data is not necessarily the

closest to ground truth), (2) uncertainty in the preprocessing steps as mentioned previously can be a dominant factor, (3) the stability of the product algorithms can be another reason, and (4) data quality for deriving products may be improved after compression. Previous experience indicates that the VQ compression process can be viewed as a nonlinear filter that eliminates artifacts (such as salt-and-pepper noise etc.). This sometimes improves the data quality for derived products.

User #3 utilized a test datacube collected under a well-controlled laboratory environment. The test datacube produced by the instrument was at the reflectance level, and no calibration or atmospheric correction preprocessing steps were applied to the datacube. The products derived from the original datacube were the ground truths in this case and used as the metrics to assess the impact of data compression. The overall classification accuracies corresponding to all the compressed datacubes were all better than 98%. This user's evaluation results indicate that both the SAMVQ and HSOCVQ algorithms have no impact on the derivation of fraction images and SAM images in this application. These results have higher reliability because the original datacube is free of the preprocessing uncertainties (i.e., calibration, atmospheric correction) that usually occur in airborne and spaceborne hyperspectral datacubes. This user assessed the impact of only one error source—the error introduced by the compression—on the products.

User #4 used a test datacube with synthetic targets in the scene of the datacube and accepted all of the datacubes compressed using both SAMVQ and HSOCVQ. The EMs of the targets and ground truth were known. The products derived from the original datacube were used as metrics to assess the products derived from the compressed datacubes, as they had been well validated with the ground truth.

Users #5 and #6 accepted all of the datacubes compressed using both SAMVQ and HSOCVQ, although they did not compare the products with the ground truth. User #5 used the reference spectra extracted from the JPL and USGS mineral spectral libraries as the "ground truth" and compared the least squares fit of the spectra of a blind datacube to the corresponding reference spectra at each selected wavelength.

User #6 evaluated three test datacubes for ship and wake detection. For each test datacube, the percentage accuracy of wake detection derived from the original datacube in the blind datacube set was always ranked first. This is not surprising because the original datacube was being used as the benchmark. The products derived from the user-archived original datacube and from the original datacube in the blind datacube set should be exactly the same, as the two original datacubes are identical. The difference between the products derived from the two identical original datacubes indicates that the product algorithm is not stable or repeatable. For test datacube 006/2100,

the correction percentage of wake detection derived from the original datacube in the blind datacube set using moving average classification (MAC) lost 10% compared with that from the user-archived original datacube.

User #7 did not perform statistical tests for the quantitative analysis. This user evaluated the effect of compression strictly based on qualitative results. For three datacubes compressed using SAMVQ, they accepted the datacube compressed at 20:1, marginally accepted 50:1 and rejected 30:1. The fact that the datacube compressed at 30:1 was ranked lower than the datacube compressed at 50:1 probably relates to the stability of the product algorithms and the uncertainty in the preprocessing steps.

User #8 accepted the datacube compressed at 10:1 using SAMVQ because it obtained the same score as the original data. The user marginally accepted 20:1 using SAMVQ and rejected the rest of the datacubes. User #9 accepted the datacube compressed at 10:1 using SAMVQ and rejected the rest of the datacubes.

Both users #10 and #11 did not undergo a double-blind test. User #10 evaluated only two datacubes compressed using SAMVQ (10:1 and 20:1), whereas user #11 evaluated one datacube compressed using SAMVQ (20:1) and three datacubes using HSOCVQ.

In summary, the evaluation results obtained by performing double-blind testing and comparing the derived products with ground truth should have the highest weight because double-blind testing reduces error, self-deception, and bias in the evaluation process, and the use of ground truth as a benchmark removes uncertainties in the products derived from the original data caused by the intrinsic instrument noise of the data and preprocessing steps. The evaluation results obtained by performing double-blind testing and by using the products derived from the original data as benchmark should have median weight because the uncertainties caused by the intrinsic instrument noise of the data and preprocessing steps are still in the benchmark. The evaluation results obtained by neither performing double-blind testing nor using ground truth as a benchmark should have the lowest weight.

The location of the compression in the data flow chain of an imaging spectrometer system is an important issue in a user acceptability study. Three evaluation systems with the compression intervening at different locations in the data flow chain were proposed.[26] The first system had the onboard compressor located immediately following the A/D converter (i.e., using sensor digital count as input), the second after the detector gain and offset (called scaling) is corrected, and the third after a full calibration (i.e., using radiance as input). In this study, most users evaluated the impact of the compression using datacubes at the radiance-processing level as input (i.e., system 3); only two users evaluated data at the DN-processing level (i.e., system 1).

8.7 Effect of Lossy Data Compression on Retrieval of Red-Edge Indices

This section evaluates the effects of lossy VQ hyperspectral data compression algorithms using red-edge indices as end remote sensing products.[25] Three CASI datasets and one AVIRIS dataset from vegetated areas were tested.

A VQ data compression algorithm for compressing 3D hyperspectral datacubes called 3DVQ and its three speed-improved compression algorithms were examined. Five red-edge products representing the NIR reflectance shoulder (*Vog*1), the NIR reflectance maximum (*Red rs*), the difference between the reflectance maximum and the minimum (*Red rd*), the wavelength of the reflectance maximum (*Red lo*), and the wavelength of the point of inflection of the NIR vegetation reflectance curve (*Red lp*) were retrieved from each original datacube and from their decompressed datacubes to evaluate the impact of the lossy compression algorithms.

8.7.1 Test datacubes

Hyperspectral datacubes used in this section were acquired by CASI[45] in July 1996 and by AVIRIS[46] in August 1996 from the Southern Study Area located north of Prince Albert, in Saskatchewan, Canada (through the BOREAS project).[47] This data has been converted to units of scene reflectance using the Imaging Spectrometer Data Analysis System (ISDAS) atmospheric-correction software.[39] Three CASI datacubes of 400 pixels by 800 lines by 72 bands by 16 bits were extracted from the BOREAS dataset acquired using the CASI sensor and named after the BOREAS tower sites because the datacubes are all from small areas near the tower sites; they are called *yjps* (Young Jack Pine site), *ojps* (Old Jack Pine site), and *fens* (Fen site), respectively.

An AVIRIS datacube of 614 pixels by 512 lines by 205 bands by 16 bits was extracted from two datasets that were collected over approximately the same ground targets as the CASI datasets. It is composed of the bottom half of one AVIRIS scene and the top half of the next scene. The datacube is combined in this way so as to include both the Young Jack Pine and Old Jack Pine sites, which appear in the CASI datacubes. The original 224 AVIRIS bands were reduced to 205 bands by removing bands that are affected by high atmospheric absorption, as the atmospheric correction cannot address the low signal level in these bands.

8.7.2 Red-edge indices

Two red-edge indices were used for evaluation.

1. Vogelmann red-edge reflectance ratio index (*Vog*1):[48]

This index is correlated with pigment and given by the expression

$$Vog1 = \frac{R(740\,\text{nm})}{R(720\,\text{nm})}, \tag{8.4}$$

where $R(\lambda)$ is the surface reflectance at wavelength λ.

2. Inverse Gaussian red-edge spectral parameters:[49]
   The red-edge parameters or the inverted Gaussian model of the reflectance curve between 670–780 nm are defined by the following equation:

$$R(\lambda) = R_s \quad (R_s \quad R_o)e^{\frac{(\lambda \ \lambda_o)^2}{2\sigma^2}}, \tag{8.5}$$

   where R_s is the reflectance maximum, R_o is the reflectance minimum, λ_o is the spectral position of the reflectance minimum, λ_p is the spectral position of the inflection of the Gaussian red-edge reflectance curve, and $\sigma = \lambda_p - \lambda_o$ is the Gaussian curve width parameter.

Figure 8.8 depicts the definition of the curve fit and the parameters of vegetation reflectance red-edge in terms of two reflectance parameters (R_s and R_o) and three spectral parameters (λ_p, λ_o, and σ).

Both of these indices[50 52] are useful in vegetation applications; they are selected for evaluation because (1) the test datacubes are from vegetated areas, (2) research groups utilizing hyperspectral imagery for forestry applications are currently using the $Vog1$ index, and (3) the inverse Gaussian red-edge spectral parameters require hyperspectral imagery and cannot be determined from broadband imagery.

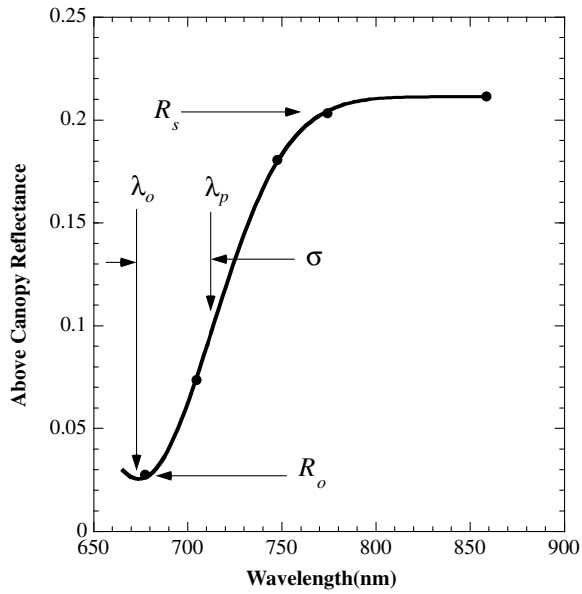


Figure 8.8 Illustration and curve fit of the inverse Gaussian red-edge parameters (reprinted from Ref. 25).

This section computes and examines the following five independent parameters (called hyperspectral products):

1. *Vog*1: Vogelmann red-edge reflectance ratio index,
2. *Red rs*: The reflectance value at the maximum point in the NIR,
3. *Red rd*: The difference between reflectance at maximum point in the NIR and reflectance at the minimum point in the red,
4. *Red lo*: The wavelength at which the reflectance minimum occurs, and
5. *Red lp*: The wavelength of the inflection point of the Gaussian fit.

8.7.3 Evaluation using red-edge products

This section describes the hyperspectral data compression systems to be evaluated, gives the statistical measures used to show the accuracy of the products due to compression, and, finally, considers the proper retrieval of the products.

The 3DVQ algorithm described in Chapter 4 treats the spectrum of a ground sample in the datacube to be compressed as a vector (also referred to as "spectral vector") and compresses the spectral vectors using the LBG algorithm.[53] It is the basic compression system in the VQ-based, hyperspectral-data-compression algorithm series and does not use any techniques for improving the speed and the best PSNR. It was used as a *reference* in the development of improved systems in order to compare performances. This section uses it as the reference compression system to examine how lossy VQ hyperspectral data compression affects the selected remote sensing applications.

Chapter 4 describes three processing-speed-improved, lossy-VQ hyperspectral-data-compression systems that integrate the MSCA,[4] training set subsampling,[5] and the SFBBC.[1] System 1 uses the MSCA with 2% training set subsampling (see Section 4.8.5); system 2 uses the MSCA with 2% training set subsampling and SFBBC for codebook training (see Section 4.8.6); and system 3 uses the MSCA with 2% training set subsampling and SFBBC for both codebook training and for coding (see Section 4.8.7). The experimental results show that the overall processing speed of the systems can be improved by a factor of around 1,000 at an average PSNR penalty of 1.5 dB. In this section, they are selected for evaluation to examine how the improved systems further impact specific remote sensing applications.

In this section, the standard deviation of percentage difference between product values retrieved from the original and from the decompressed data,

$$Std\_Dev = \sqrt{\frac{1}{n \quad 1}\left\{\sum_{i \quad 1}^{n}\left(100\frac{\hat{x}_i \quad \hat{x}_i}{x_i}\right)^2 \quad \frac{1}{n}\left(\sum_{i \quad 1}^{n}100\frac{\hat{x}_i \quad x_i}{x_i}\right)^2\right\}}, \qquad (8.6)$$

is used to evaluate the product error induced by lossy compression, where $\hat{x}_i$ is the product value at pixel i retrieved from the decompressed data, x_i is the product value at pixel i retrieved from the original data, and n is the number of pixels in the scene of a datacube.

The product algorithms are applied to the original and decompressed datacubes on a pixel-by-pixel basis. Because the red-edge indices are only valid for vegetated pixels, the products will be computed in a region dominated by vegetation coverage to reduce the influence of nonvegetated pixels on the evaluation. In the CASI datacubes, a subscene of 256×256 pixels within the scene of a test datacube was selected for computation of the products. The subscene selected contains over 90% vegetated pixels.

Two additional mechanisms were used to prevent computations from nonvegetated pixels in the subscene from affecting the results: (1) The product algorithms perform "reality checks" on all pixels in a subscene. If a pixel is nonvegetated, it is flagged and will not be used in subsequent analysis. (2) The statistics computation algorithm ignores pixels for which the product value does not lie within an expected range. These ranges are given in Table 8.11.

During the generation of the statistics, the algorithm also records: (1) The number of pixels for which both the original datacube and the decompressed datacube produced viable product values. (2) The number of pixels for which neither the original datacube nor the decompressed datacube produced viable product values. (3) The number of pixels for which only the original datacube or the decompressed data produced viable product values (tracked separately). These pixel counts are also used to ensure that the above screening methods are working properly.

8.7.4 Evaluation results and analysis

This section's evaluation is first applied to the three CASI hyperspectral datacubes and then validated using the AVIRIS datacube.

8.7.4.1 From CASI datacubes

The three CASI test datacubes were compressed using the four systems described above with eight different-size codebooks. Each system yields

Table 8.11 Viable data range of the five red-edge products (reprinted from Ref. 25).

Product Name	Minimum Accepted Value	Maximum Accepted Value
*Vog*1	0.0	10.0
Red rs [%]	0.0	60.0
Red rd [%]	0.0	60.0
Red lo [nm]	660.0	700.0
Red lp [nm]	693.0	733.0

Table 8.12 Eight codebook sizes and their corresponding CRs on the CASI test datacubes (reprinted from Ref. 25).

Codebook Size	32	64	128	256	512	1024	2048	4096
Compression Ratio	225.2	184.9	154.4	129.1	106.2	84.2	62.7	43.1

the same CR but different reconstruction fidelity for the same-size codebook. Table 8.12 lists the compression ratios corresponding to each codebook size. The compression results (fidelity versus compression) are shown in Fig. 8.9. The reference system yields the best PSNR for each of the test datacubes, but it is the slowest. Systems 1 and 2 produce similar PSNRs that are close to those of the reference system. System 3 produces the worst PSNR of the systems, but it is the fastest. The compressed data was decompressed to produce the reconstructed datacubes for evaluation.

Five red-edge products—*Vog*1, *Red rs*, *Red rd*, *Red lo*, and *Red lp*—were retrieved from each original CASI datacube and from their decompressed datacubes, which were produced by the four compression systems at eight CRs. Thus, a total of 33 product images were generated per product because there are 32 decompressed datacubes corresponding to one original datacube. The standard deviation of percentage difference between the product retrieved from the original and those retrieved from the decompressed data were computed based on Eq. (8.6).

Figure 8.10 shows graphs of standard deviations of percentage difference between the five red-edge products retrieved from the Young Jack Pine site datacube and its decompressed datacubes, and their average graph as a function of CRs. Each curve in a graph represents one compression system and has eight points, each of which corresponds to a CR. In general, the reference system induces the smallest product errors of the four compression systems. Systems 1 and 2 perform similarly—their curves are closer to the reference than to system 3. The product errors increase with increases in CR.

The dynamic range of the standard deviation for *Vog*1 is 1.8–2.8%. The dynamic range of the standard deviation of the two reflectance-related products is 1.7–7.0%. The dynamic range of the standard deviation of the two wavelength-related products is very small. It is smaller than 0.14%, which indicates that the lossy VQ compression has little effect on wavelength. The spectral information in the wavelength domain is well preserved during the process of this lossy compression. The curves of the product errors corresponding to different compression systems start to converge once the CR exceeds 106.

In the *Vog*1 graph, the standard deviations from compression systems 1 and 2 remain constant (around 2.2%) for CRs from 43 to 106. In contrast, the

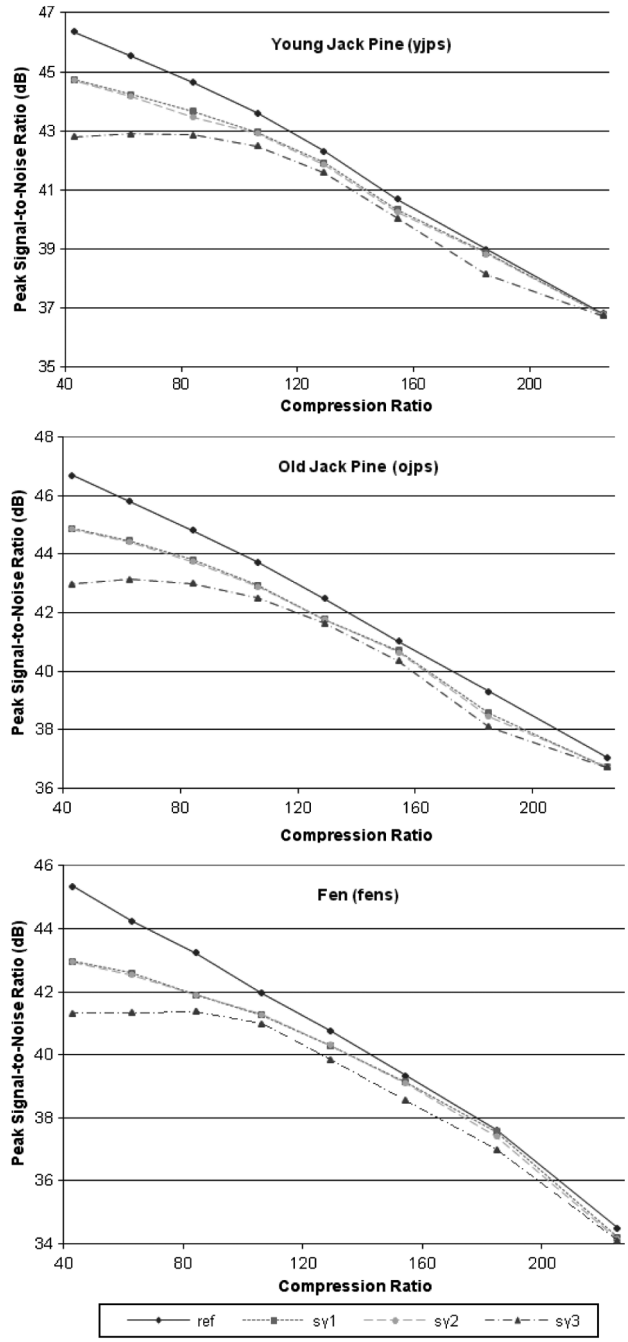


Figure 8.9 Compression ratio vs. PSNR of the four compression systems on three CASI datacubes (reprinted from Ref. 25).

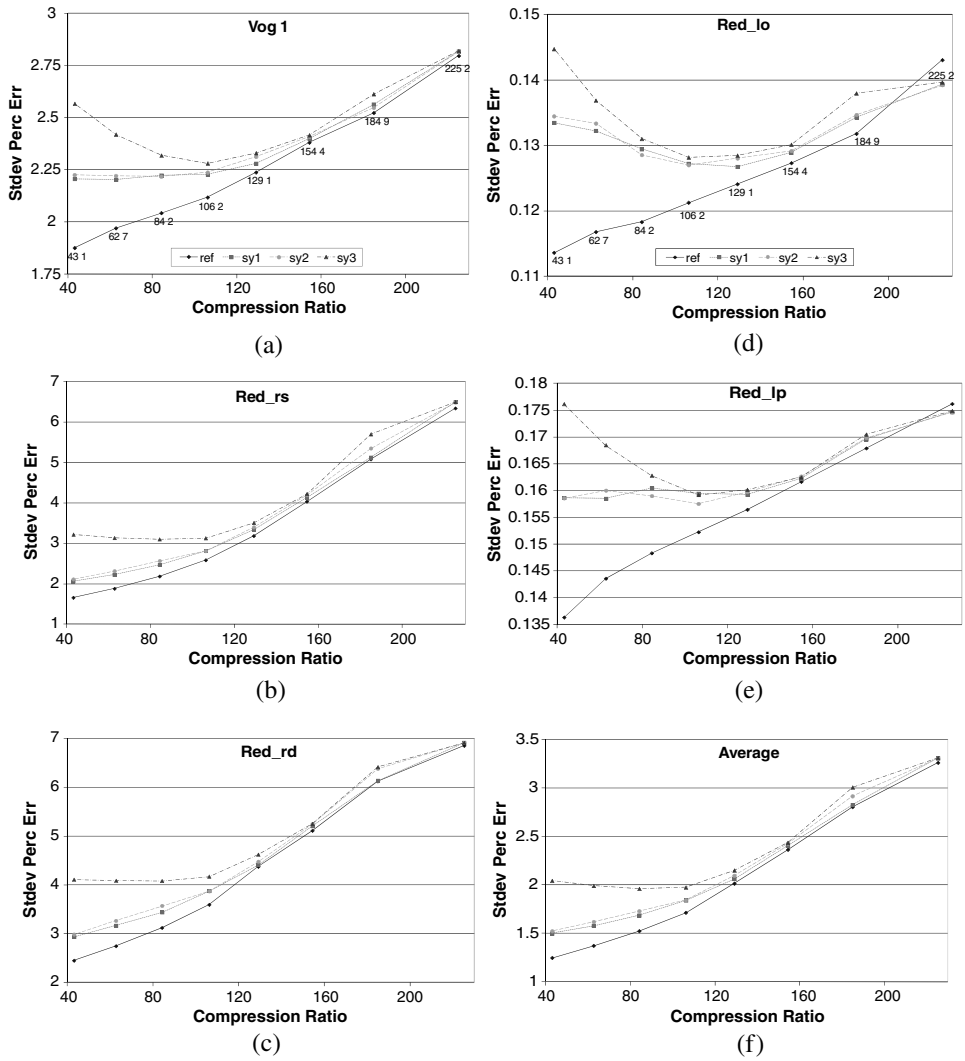


Figure 8.10 Standard deviation of percentage difference between products retrieved from the original and decompressed datacubes. The test datacube comes from the Young Jack Pine site (reprinted from Ref. 25).

standard deviation from the reference system increases from 1.8% to 2.1% at the same CRs. In this case, systems 1 and 2 with a CR of 106 are optimal from the point of view of the *Vog*1 application because they produce the best compression and induce the smallest error on the product. The curves of standard deviation versus CR are almost linear after the CR exceeds 106. The standard deviation values from compression system 3 at CRs of 43, 62, and 84 are worse than that at a CR of 106. This is probably caused by the instability of the compression system. It can be seen from the PSNR versus CR curves

shown in Fig. 8.9 that the improvement in PSNR becomes very small when the CR decreases below the compression ratio of 106. For a compression ratio of 43, the PSNR of system 3 is actually worse than it is at slightly higher CRs. Comparing the *Vog*1 curve with the PSNR curve, it can be seen that they demonstrate similar trends; the only difference is in their sensitivity. *Vog*1 is more sensitive than PSNR to this instability of the system. Fortunately, the absolute amount resulting from this instability is small (less than 0.3% in this example).

In the *Red rs* and *Red rd* graphs, the standard deviations of the systems are relatively large when the CR is high; they almost reach 6.5% and 7.0% in graphs *Red rs* and *Red rd*, respectively, when the CR is 225. They are more sensitive to compression than other products. The curves of standard deviation from compression systems 1 and 2 are very close to that from the reference system in both of the graphs. The product errors from system 3 remain constant at CRs from 43 to 106 in both graphs, and they are slightly above 3.0% and 4.0% in the two graphs. Below a CR of 106, this is caused by the poorer performance of system 3 compared to the other systems.

In the two wavelength-related products graphs *Red lo* and *Red lp*, all three speed-improved compression systems perform poorly with respect to the reference system when the CR is smaller than 106. In the graph *Red lo*, the standard deviation curves from these three systems bend upward when the CR is less than 106. The product errors decrease with the increase of the CR up to 106 rather than increase. In the graph *Red lp*, the standard deviations from system 3 are similar to those in graph *Red lo*, while the standard deviations from systems 1 and 2 slightly oscillate around a value of 0.16%. These results indicate that at a lower CR (<106), the wavelength-related products are more sensitive to the improved compression techniques than the reflectance-related products, although the absolute value of the errors is 15 to 40 times smaller than that of the reflectance products. System 3 is the poorest of the three improved systems with respect to the reference system.

Figure 8.3(f) graphs the average over the five products. It is presented to summarize the overall product error of a test datacube. The average product errors are dominated by products *Vog*1, *Red rs*, and *Red lp*, which have large values of standard deviation. It can be seen from this graph that systems 1 and 2 perform as well as the reference system in terms of the overall product errors. Systems 1 and 2 compress a datacube hundreds of times faster than the reference system. The differences between the overall errors from the reference and those from system 1 or 2 are below 0.25% at all CRs. They are insignificant, especially when CRs are greater than 106. It is recommended that system 1 or 2 be used with a CR of no more than 106 if one wants to obtain a relatively small error on these products. In this way, the overall

product error would be below 2.0%, which is an acceptable error level in most applications.

For example, spectral indices used for the estimation of chlorophyll content in higher plant leaves are typically based on the red-edge inflection point *Red lp* or reflectance ratios such as R_{750}/R_{700} and R_{750}/R_{550}.[54,55] A 2.0% error in the retrieved reflectance ratio could produce an error in the reflectance ratio of 4.0%. The resulting uncertainty of the estimated chlorophyll content *Chl a+b* using the results of the work in Gitelson and Merzlak [54] for maple leaves, for example, would be 0.56 μg cm $^{-2}$. This is much less than the estimated error in the regression using the R_{750}/R_{700} ratio of 3.6 μg cm $^{-2}$.

Each of the above experiments was also carried out on the Old Jack Pine datacube. The results showed similar trends, except the results are slightly better than those of the Young Jack Pine datacube, since the PSNRs obtained using the Old Jack Pine datacube are slightly better than those obtained using the Young Jack Pine datacube.

Figure 8.11 shows the graphs of standard deviations of the products retrieved from the Fen datacube. The trends of standard deviations of the products *Vog*1, *Red rs*, and *Red rd* are similar to those of the Young Jack Pine datacube, but their values are slightly worse (1.0% to 3.0% worse), which is because the PSNRs of the Fen decompressed datacubes are about 1.5 dB lower than those of Young Jack Pine datacube (see Fig. 8.9).

In general, products *Red lo* and *Red lp* from the Fen site are up to 0.14% worse than those from the Young Jack Pine site. As above, this is caused by slightly poorer PSNR of the decompressed datacubes. The *Red lo* and *Red lp* curves of the reference system from the Fen site are not as straight as those from the Young Jack Pine site, especially the *Red lo* curve. Unlike the Young Jack Pine site, the *Red lo* and *Red lp* curves from the Fen site compressed by the three improved systems are very close and do not bend upward explicitly in the low CR range. The differences between standard deviations from the improved systems and those from the reference are around 0.07% when CRs are 184 and 225 in the two products. They are relatively large compared to the difference of 0.006% at CRs between 84 and 154, although the absolute value is quite small.

The average standard deviations over the five products are plotted in graph (f) of Fig. 8.11. Similar to the Young Jack Pine site, they are dominated by products *Vog*1, *Red rs*, and *Red lp*. Systems 1 and 2 perform as well as the reference system. The differences between overall product errors from the reference system and from system 1 or 2 are below 0.5% at all CRs. For the Fen datacube, system 3 performs quite well; compared with the Young Jack Pine datacube, the degradation of performance in the low-CR range is not explicit.

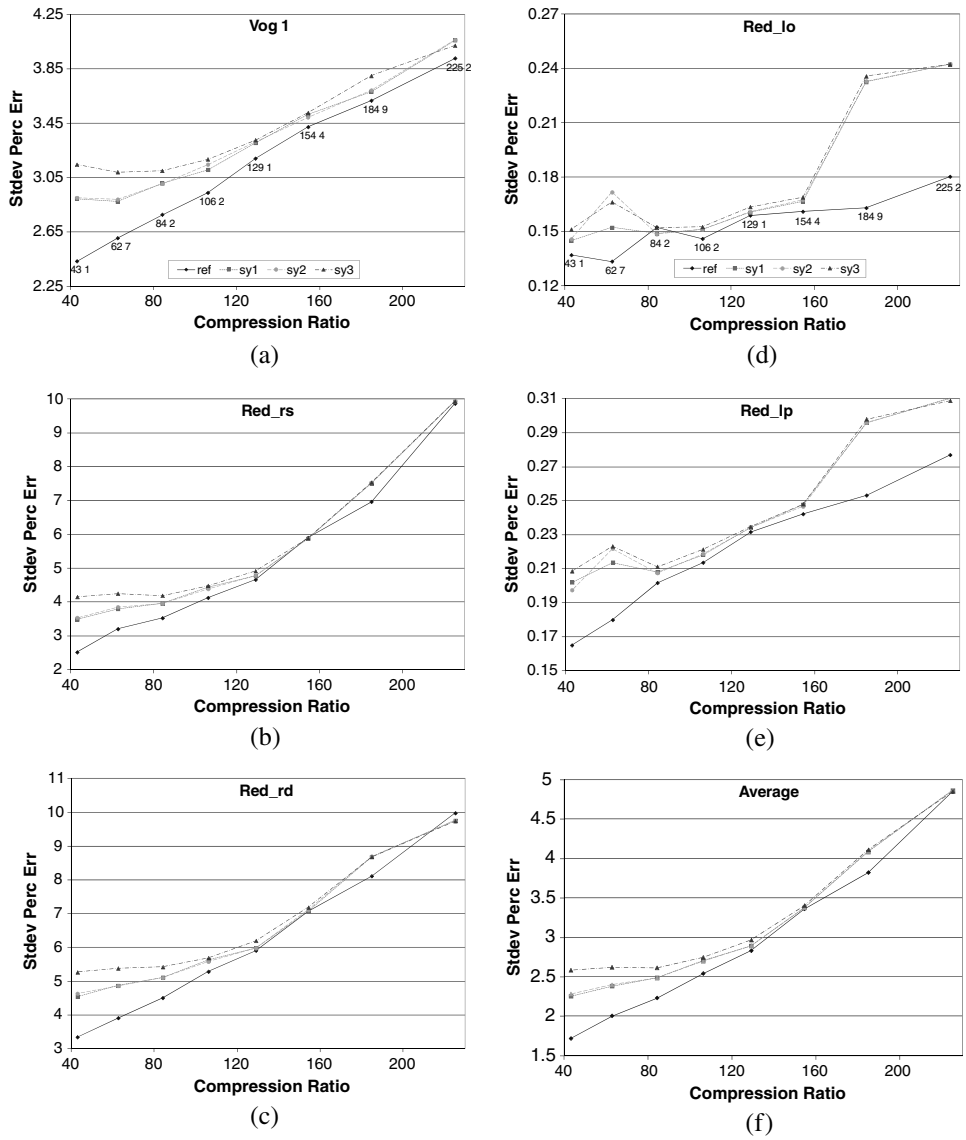


Figure 8.11 Standard deviation of percentage difference between products retrieved from the original and decompressed datacubes. The test datacube comes from the Fen site (reprinted from Ref. 25).

8.7.4.2 From AVIRIS datacube

The AVIRIS datacube was compressed using the four systems at four codebook sizes: 512, 1024, 2048, and 4096. With these four codebook sizes, the four systems yield the same CRs of 229, 159, 101, and 60, respectively. These CRs are selected for evaluation since they are in a similar range to those

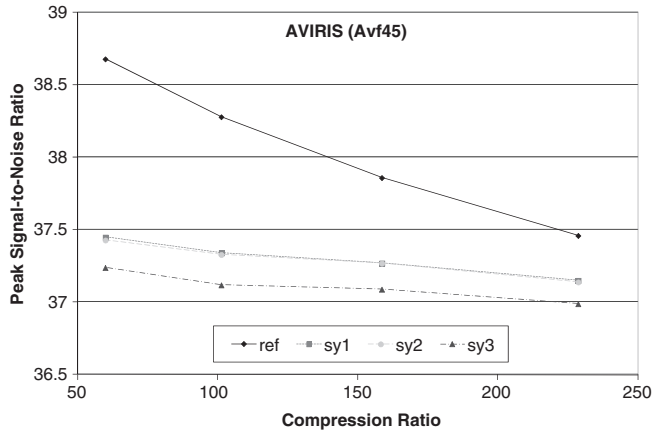


Figure 8.12 Compression ratio versus PSNR of the four compression systems on the AVIRIS datacube (reprinted from Ref. 25).

obtained using the CASI datacubes. Figure 8.12 shows the compression results in terms of the MSE distortion measure. The PSNRs of the decompressed data, produced by these four systems on the AVIRIS datacube, are smaller than those on the CASI datacubes. The decrease in PSNR with the increase in CR is also small. The PSNR produced by the reference system decreases only 1.3 dB from the smallest CR of 60 to the largest ratio of 229. The PSNRs produced by the improved systems are very close. Systems 1 and 2 produced almost the same PSNR at the four CRs, so that their curves of CR versus PSNR are merged together. The maximum difference between PSNRs produced by system 1 or 2 and by system 3 is only 0.2 dB.

The product errors retrieved from the original AVIRIS datacube and its decompressed datacubes, as a function of CR, are shown in Fig. 8.13. Compared with the graphs in Figs. 8.10 and 8.11, the graphs of the product errors from the AVIRIS datacube demonstrate similar trends to those obtained from the CASI datacubes. In general, the standard deviations of all the products from the AVIRIS datacube are smaller than those from the CASI datacubes. This probably results from the attributes of the AVIRIS datacube. Most notably, the AVIRIS datacube is approximately three times larger than the CASI datacubes. Therefore, higher CRs are expected. For CRs near 106, the CASI Fen and Young Jack Pine datacubes yield error statistics similar to the error statistics from the AVIRIS datacube. For CRs above 106, the overall product error increase for the AVIRIS datacube is minimal, while the overall product error increase for the CASI datacubes is quite significant. To achieve comparable CRs, the codebook size for the CASI datacubes must be much smaller than for the AVIRIS datacube (for example, to achieve a CR of 106, the CASI codebooks can only contain 512 codevectors, whereas the AVIRIS codebooks can contain 2048 codevectors). Therefore, the impact of

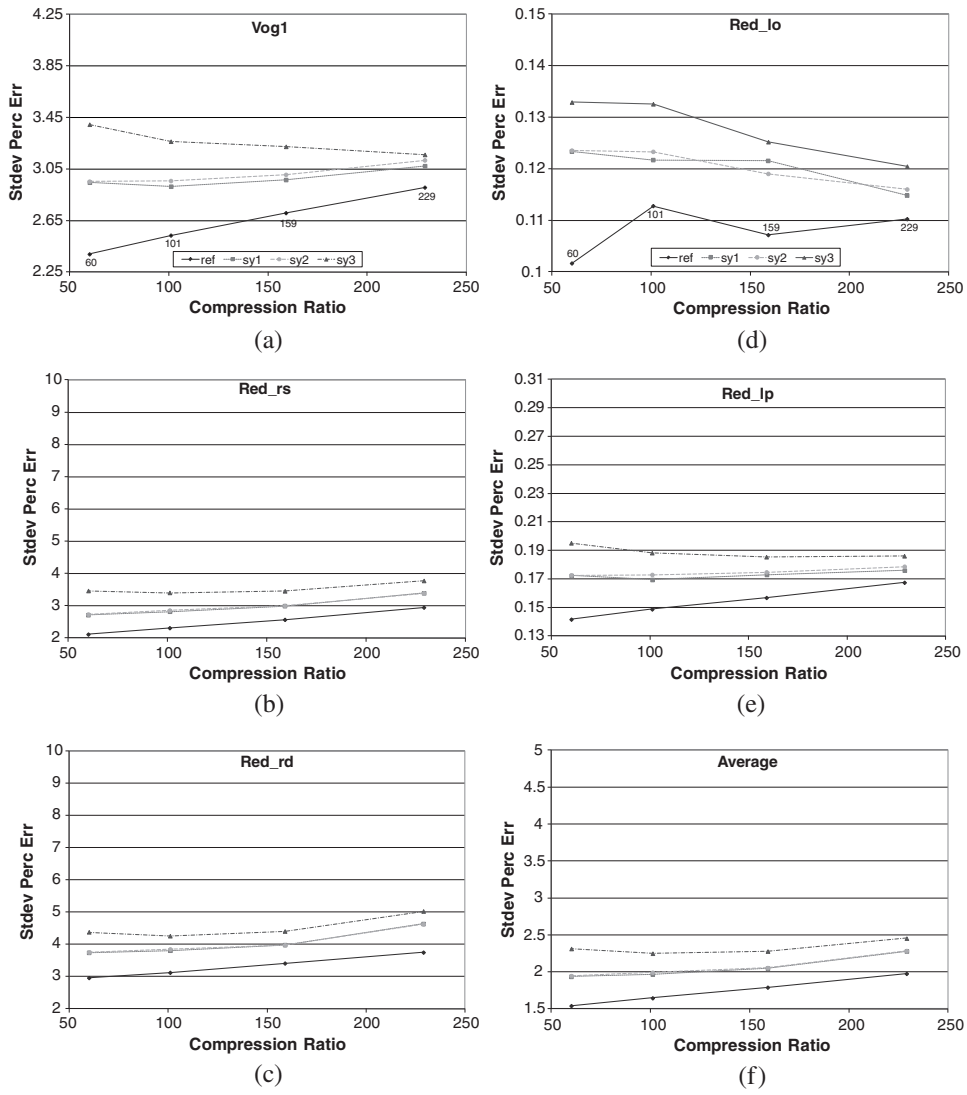


Figure 8.13 Standard deviation of percentage difference between products retrieved from the original and decompressed datacubes. The test datacube comes from AVIRIS (reprinted from Ref. 25).

CR on overall product error is more significant for the CASI datacubes than it is for the AVIRIS datacube. Figure 8.14 shows the average product errors of the CASI and AVIRIS datacubes compressed by system 2 for the scenario detailed earlier.

The CASI datacubes selected for evaluation in this section demonstrate worst-case results because the datacubes are small. If a datacube is large enough spatially, a high CR can be obtained even if a large codebook is used;

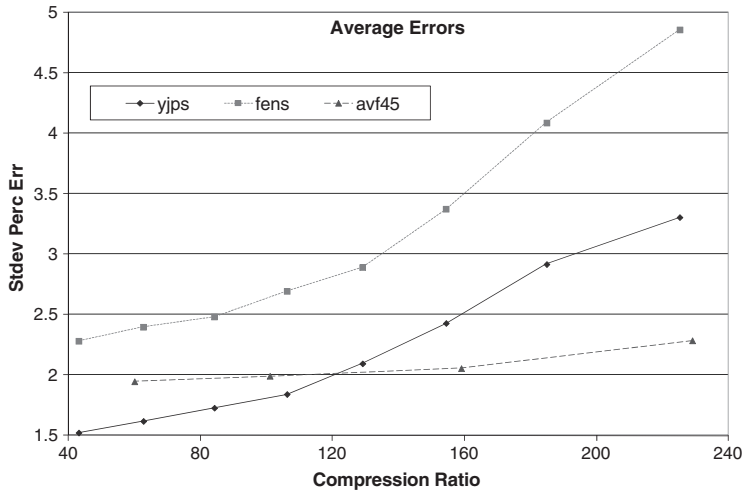


Figure 8.14 Comparison of overall product errors from CASI datacubes (Young Jack Pine and Fen sites) and the AVIRIS datacube compressed by system 2 (reprinted from Ref. 25).

this will yield minimal induced product errors. This validation test gives us more confidence on the product error statistics.

8.7.4.3 Spatial patterns of induced product errors

The previous analysis shows statistical results from comparisons between data products derived from the original and the decompressed datacubes. In this subsection, the spatial component of the product errors is presented. Figure 8.15 shows a series of pictures of spatial patterns of the product errors retrieved from the AVIRIS datacube. A quick-look image of the complete scene of the AVIRIS datacube, generated at 733 nm, is presented in Fig. 8.15(a). Figures 8.15(b)–(f) show the absolute value of the percentage error between each red-edge product calculated from the original datacube and from the decompressed datacube produced by system 3. System 3 yields the worst PSNR in terms of MSE distortion and the largest product errors of the four compression systems. Thus, the pictures of spatial patterns resulting from system 3 show the worst cases of the compression systems.

These pictures show that for all products, errors are relatively high along the north–south river and river bank, and along narrow logging roads, where mixed pixels would occur. For all products, errors are relatively high in clear-cut areas, in part because the area is primarily nonvegetated (some limited regrowth only) and because the regions tend to be nonuniform and would therefore contain highly mixed pixels (consisting of various amounts of soil, deadfall, and a variety of vegetation in different stages of growth). For the *Vog*1 product, the highest errors are generated over water bodies [the bright

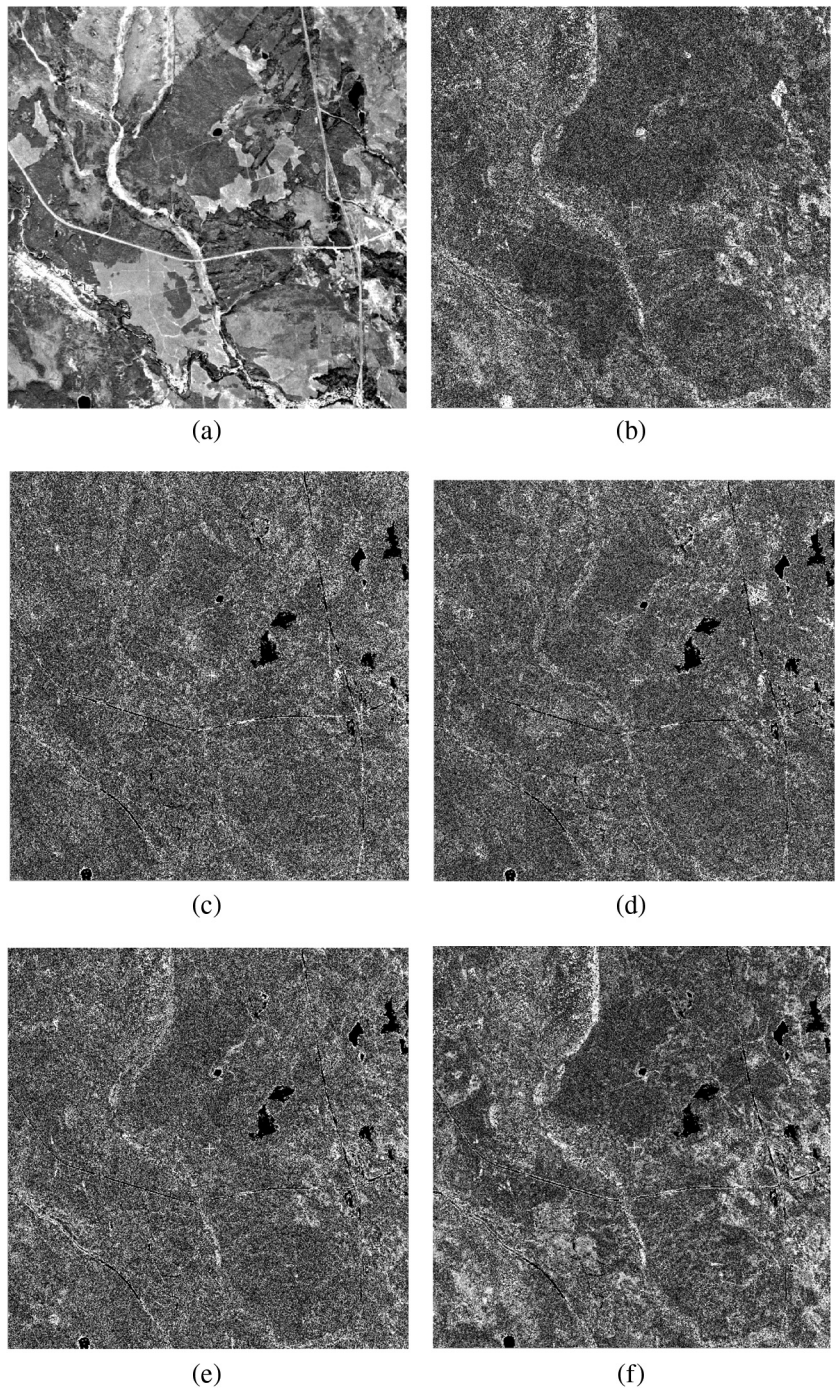


Figure 8.15 Spatial patterns in induced product errors of the AVIRIS data. (a) A quick-look image of the scene at wavelength 733 nm. (b)–(f) Images of the absolute value of the percentage error between each red-edge product calculated from the original datacube and from the reconstructed datacube compressed by system 3 using the following products: (b) *Vog*1, (c) *Red rs*, (d) *Red rd*, (e) *Red lo*, and (f) *Red lp* (reprinted from Ref. 25).

regions in Fig. 8.15(b)]. The high errors for nonvegetated pure and mixed pixels are inherent because the red-edge algorithm does not apply to the spectra for these pixels. Consequently, this error is of no significance: in the area of interest—vegetation—the error is uniformly distributed.

8.7.5 Summary of the evaluation

Four lossy VQ hyperspectral data compression algorithms were evaluated using red-edge indices as end products. The standard deviation of percentage difference between a product retrieved from an original datacube and that from its decompressed datacube was used as a measure to quantify the impact of compression on end products. Three CASI datacubes from vegetated areas were tested. An AVIRIS datacube collected over approximately the same ground targets at approximately the same time of the year was also tested to validate the experimental results.

Four lossy VQ hyperspectral data compression systems were examined. A basic compression system called the reference system, which does not use any techniques to improve the speed, yielded the best PSNR for all test datacubes, but it is the slowest method. Systems 1 and 2 produced PSNRs close to that of the reference. System 3 produced the worst PSNR, but it is the fastest. Five red-edge products (*Vog*1, *Red rs*, *Red rd*, *Red lo*, and *Red lp*) were retrieved from each original datacube and from their decompressed datacubes, and were analyzed.

For CASI datacubes, the reference induces the smallest product errors of the four compression systems. System 1 and 2 perform fairly similarly to the reference. System 3 performs similarly to the reference at high CRs. Product errors increase with the increase of CR. The amplitude of product error of the two wavelength-related products *Red lo* and *Red lp* (below 0.18%) is 15 to 40 times smaller than that of the two reflectance-related products *Red rs* and *Red rd*. This indicates that the lossy VQ compression has little impact on wavelength. The spectral information in the wavelength domain can be well preserved during the process of lossy compression. The overall product errors are dominated by *Vog*1, *Red rs*, and *Red lp*, which have relatively large values of induced error. The difference between the overall error from the reference and that from system 1 or 2 is below 0.5% at all CRs, which is insignificant. Systems 1 and 2 compress a datacube hundreds of times faster than the reference system. The overall product error induced by system 1 or 2 is below 2.0% for the Young Jack Pine and Old Jack Pine datacubes and below 3.0% for the Fen datacube when the CR is 106 and below. It is recommended that system 1 or 2 be used with a CR no greater than 106 to obtain a relatively small error in these products.

For the AVIRIS datacube, similar trends as in the CASI datacubes were observed. In general, the product errors from the AVIRIS datacube are smaller than those from the CASI datacubes. This probably results from the

greater compressibility of the AVIRIS datacube, which is due to the fact that it is 3 times larger than the CASI datacubes, and thus, higher CRs are expected. When the CR reaches a certain value, the CASI codebooks can contain only a small number of codevectors to achieve a compatible CR, whereas the AVIRIS codebooks can contain a large number of codevectors. Minimal induced product errors would be expected if a datacube is large enough spatially because a codebook can still contain a sufficiently large number of codevectors at a high CR.

Spatial patterns of the product errors of the AVIRIS datacube were also presented. For all products, errors are uniformly distributed in vegetated areas. Errors are relatively high in nonvegetated and mixed pixel areas.

References

1. Qian, S.-E., A. B. Hollinger, D. Williams, and D. Manak, "Fast 3D data compression of hyperspectral imagery using vector quantization with spectral-feature-based binary coding," *Opt. Eng.* **35**(11), 3242–3249 (1996) [doi: 10.1117/1.601062].
2. Qian, S.-E., A. B. Hollinger, D. Williams, and D. Manak, "A near lossless 3-dimensional data compression system for hyperspectral imagery using correlation vector quantization," *Proc. 47th International Astronautical Congress*, Beijing (1996).
3. Qian, S.-E., A. B. Hollinger, D. Williams, and D. Manak, "3D data compression system based on vector quantization for reducing the data rate of hyperspectral imagery," *Applications of Photonic Technology* **2**, 641–654, Plenum Press, New York (1997).
4. Qian, S.-E., A. B. Hollinger, D. Williams, and D. Manak, "3D data compression of hyperspectral imagery using vector quantization with NDVI-based multiple codebooks," *Proc. IEEE International Geoscience and Remote Sensing Symposium* **3**, 2680–2684 (1998).
5. Manak, D., S.-E. Qian, A. B. Hollinger, and D. Williams, "Efficient Hyperspectral Data Compression using vector Quantization and Scene Segmentation," *Canadian J. Remote Sens.* **24**, 133–143 (1998).
6. Qian, S.-E., A. B. Hollinger, and Y. Hamiaux, "Study of real-time lossless data compression for hyperspectral imagery," *Proc. IEEE International Geoscience and Remote Sensing Symposium* **4**, 2038–2042 (1999).
7. Qian, S.-E., A. B. Hollinger, D. Williams, and D. Manak, "Vector quantization using spectral index based multiple sub-codebooks for hyperspectral data compression," *IEEE Trans. Geosci. Remote Sens.* **38**, 1183–1190 (2000).
8. Qian, S.-E., A. B. Hollinger, M. Dutkiewicz, H. Tsang, and H. Zwick, "Fast Spectral Processing of Hyperspectral Image Cubes," *Proc. 14th*

International Conference and Workshop on Applied Geologic Remote Sensing, Las Vegas, USA (2000).

9. Qian, S.-E. and A. B. Hollinger, "Applications of wavelet data compression using modified zerotrees in remotely sensed data," *Proc. IEEE International Geoscience and Remote Sensing Symp. 2000* **6**, 2654–2556 (2000).
10. Qian, S.-E., M. Bergeron, and J. Ko, "User Manual of Program rm neg spk," Technical Report, Canadian Space Agency, Canada (2002).
11. Qian, S.-E., A. Hollinger, M. Bergeron, and I. Cunningham, "Real-time data compression onboard a hyperspectral satellite and user acceptability," *25th Canadian Remote Sensing Symposium*, Montreal, Canada (October 2003).
12. Qian, S.-E., A. Hollinger, and U.S. Patent No. 6,701,021 B1, "System and Method for Encoding/Decoding Multi-dimensional Data Using Successive Approximation Multistage Vector Quantization (SAMVQ)," *issued on March* **2**, 2004.
13. Qian, S.-E. and A. Hollinger, U.S. Patent No. 6,724,940 B1 "System and Method for Encoding Multi-dimensional Data Using Hierarchical Self-Organizing Cluster Vector Quantization (HSOCVQ)," issued on April 20, 2004.
14. Qian, S.-E., A. Hollinger, and U.S. Patent No. 6,798,360 B1, "Method and System for Compressing a Continuous Data Flow in Real-Time Using Recursive Hierarchical Self-Organizing Cluster Vector Quantization," *issued on September* **28**, 2004.
15. Qian, S.-E., "Hyperspectral data compression using a fast vector quantization algorithm," *IEEE Trans. Geosci. Remote Sens.* **42**(8), 1791–1798 (2004).
16. Qian, S.-E., M. Bergeron, and A. Hollinger, "Evaluation of near lossless feature of two on-board data compression algorithms: SAMVQ and HSOCVQ," *WSEAS Trans. Systems* **3**(5), 2153–2158 (2004).
17. Qian, S.-E., M. Bergeron, J. Lévesque, and A. Hollinger, "Impact of Preprocessing and Radiometric Conversion on Data Compression Onboard a Hyperspectral Satellite," *Proc. IGARSS 2005* **2**, 700–703 (2005).
18. Qian, S.-E., M. Bergeron, A. Hollinger, and J. Lévesque, "Effect of anomalies on data compression onboard a hyperspectral satellite," *Proc. SPIE* **5889**, 588902 (2005) [doi: 10.1117/12.613195].
19. Qian, S.-E., J. Lévesque, and R. A. Neville, "Evaluation of Impact of Noise Removal of Radiance Data on Onboard Data Compression of Hyperspectral Imagery," *Proc. WSEAS Remote Sens. Conf.*, 37–42 (November 2005).

20. Qian, S.-E., J. Lévesque, and R. A. Neville, "Effect of removing random noise of radiance data using smoothing on data compression onboard a hyperspectral satellite," *WSEAS Trans. Systems* **5**(1), 219–224 (2006).
21. Qian, S.-E., M. Bergeron, I. Cunningham, L. Gagnon, and A. Hollinger, "Near lossless data compression on-board a hyperspectral satellite," *IEEE Trans. Aerospace and Electronic Systems* **42**(3), 851–866 (2006).
22. Qian, S.-E., "Fast vector quantization algorithms based on nearest partition set search," *IEEE Trans. Image Process.* **15**(8), 2422–2430 (2006).
23. Qian, S.-E. and Allan Hollinger, "Current status of satellite data compression in Canadian Space Agency," *Proc. SPIE* **6683**, 668304 (2007) [doi: 10.1117/12.740633].
24. Qian, S.-E. and A. Hollinger, U.S. Patent No. 7,551,785 "Method and System for Compressing a Continuous Data Flow in Real-Time Using Cluster Successive Approximation Multistage Vector Quantization," issued on June 23, 2009.
25. Qian, S.-E., A. B. Hollinger, M. Dutkiewicz, H. Tsang, H. Zwick, and J. Freemantle, "Effect of lossy vector quantization hyperspectral data compression on retrieval of red edge indices," *IEEE Trans. Geosci. Remote Sens.* **39**, 1459–1470 (2001).
26. Qian, S.-E., B. Hu, M. Bergeron, A. B. Hollinger, and P. Oswald, "Quantitative evaluation of hyperspectral data compressed by near lossless on-board compression techniques," *Proc. IEEE International Geoscience and Remote Sensing Symp.* **3**, 1425–1427 (2002).
27. Qian, S.-E., M. Bergeron, C. Serele, and A. B. Hollinger, "Evaluation and comparison of JPEG 2000 and VQ based on-board data compression algorithm for hyperspectral imagery," *Proc. IEEE International Geoscience and Remote Sensing Symposium 2003* **3**, 1820–1822 (2003).
28. Hu, B., S.-E. Qian, and A. B. Hollinger, "Impact of vector quantization compression on the surface reflectance retrieval: A case study," *Proc. IEEE International Geoscience and Remote Sensing Symposium 1999* **2**, 1174–1176 (1999).
29. Hu, B., S.-E. Qian, and A. B. Hollinger, "Impact of lossy data compression using vector quantization on retrieval of surface reflectance from CASI imaging spectrometry data," *Canadian J. Remote Sens.* **27**, 1–19 (2001).
30. Hu, B., S.-E. Qian, D. Haboudane, J. R. Miller, A. B. Hollinger, and N. Tremblay, "Impact of vector quantization compression on hyperspectral data in the retrieval accuracies of crop chlorophyll content for precision agriculture," *Proc. IEEE International Geoscience and Remote Sensing Symposium 2002* **3**, 1655–1657 (2002).

31. Hu, B., S.-E. Qian, D. Haboudane, J. R. Miller, A. B. Hollinger, and N. Tremblay, "Retrieval of crop chlorophyll content and leaf area index from decompressed hyperspectral data: the effects of data compression," *Remote Sens. Environ.* **92** (2), 139–152 (2004).

32. Staenz, K., R. Hitchcock, S.-E. Qian, and R.A. Neville, "Impact of on-board hyperspectral data compression on mineral mapping products," *Proc. ISPRS Commission VII Symposium on Resource and Environmental Monitoring*, Hyderabad, India (2002).

33. Staenz, K., R. Hitchcock, S.-E. Qian, C. Champagne, and R. A. Neville, "Impact of on-board hyperspectral data compression on atmospheric water vapor and canopy liquid water retrieval," *Proceedings of the International Symposium on Spectral Sensing Research*, Santa Barbara, CA (2003).

34. Dyk, A., D. G. Goodenough, S. Thompson, C. Nadeau, A. B. Hollinger, and S.-E. Qian, "Compressed Hyperspectral Imagery for Forestry," *Proc. IEEE International Geoscience and Remote Sensing Symposium 2003* **1**, 294–296 (2003).

35. Serele, C., S.-E. Qian, M. Bergeron, P. Treitz, A. Hollinger, and F. Cavayas, "A comparative analysis of two compression algorithms for retaining the spatial information in hyperspectral data," *Proceedings of the 25th Canadian Remote Sensing Symposium*, Montreal, Canada (2003).

36. Qian, S.-E., A. Hollinger, M. Bergeron, I. Cunningham, C. Nadeau, G. Jolly, and H. Zwick, "A multi-disciplinary user acceptability study of hyperspectral data compressed using onboard near lossless vector quantization algorithm," *Int. J. Remote Sens.* **26**(10), 2163–2195 (2005).

37. O′Neill, N. T., F. Zagolski, M. Bergeron, A. Royer, R. J. Miller, and J. Freemantle, "Atmospheric correction validation of CASI images acquired over the BOREAS southern study area," *Canadian J. Remote Sens.* **23**, 143–162 (1997).

38. Haboudane, D., J. R. Miller, E. Patty, P. Zarco-Tejada, and I. Strachan, "Hyperspectral vegetation indices and novel algorithms for predicting green LAI of crop canopies: modeling and validation in the context of precision agriculture," *Remote Sens. Environ.* **90**, 337–352 (2004).

39. Staenz, K., T. Szeredi, and J. Schwarz, "ISDAS - A System for Processing/Analyzing Hyperspectral Data," *Canadian J. Remote Sens.* **24** (2), 99–113 (1998).

40. Han, T., D. G. Goodenough, A. Dyk, and J. Love, "Detection and correction of abnormal pixels in Hyperion images," *Proc. IEEE International Geoscience and Remote Sensing Symposium 2002* **3**, 1327–1330 (2002).

41. Staenz, K. and D. J. Williams, "Retrieval of surface reflectance from hyperspectral data using a look-up table approach," *Canadian J. Remote Sens.* **23**, 354–368 (1997).

42. Boardman, J. W., "Analysis, understanding and visualization of hyperspectral data as convex sets in n-space," *Proc. International SPIE Symposium on Imaging Spectrometry* **2480**, 23–36 (1995).
43. Shimabukuru, Y. E. and J. A. Smith, "The least squares mixing models to generate fraction images derived from remote sensing on multispectral data," *IEEE Trans. Geosci. Remote Sens.* **29**, 16–20 (1991).
44. Gao, B.-C., K. B. Heidebrecht, and A. F. H. Goetz, "Derivation of scaled surface reflectance from AVIRIS data," *Remote Sens. Environ.* **44**, 165–178 (1993).
45. Anger, C. D., S. Mah, T. A. Ivanco, S. B. Achal, R. Price, and J. Busler, "Extended operational capabilities of the CASI," *Proc. 2nd International Airborne Remote Sensing Conference and Exhibition* **1**, 124–133 (1996).
46. Vane, G., R. O. Green, T. Chrien, H. T. Enmark, E. G. Hansen, and W. M. Porter, "The Airborne Visible/Infrared Imaging Spectrometer (AVIRIS)," *Remote Sens. Environ.* **44**, 127–143 (1993).
47. Miller, J. R., J. R. Freemantle, P. R. Shepherd, L. Gray, N. T. O'Neill, A. Royer, and E. Senese, "Deployment of CASI to meet the needs of BOREAS science," *Proc. 17th Canadian Remote Sensing Symposium*, Saskatoon, Canada (1995).
48. Vogelmann, J. E., B. N. Rock, and D. M. Moss, "Red-edge spectral measurements from sugar maple leaves," *Int. J. Remote Sens.* **14**(8), 1563–1575 (1993).
49. Bonham-Carter, G. F., "Numerical procedures and computer program for fitting an inverted Gaussian model to vegetation reflectance data," *Computers & GeoScience* **14**(3), 339–356 (1988).
50. Miller, J. R, J. Wu, M. G Boyer, M. Belanger, and E. W. Hare, "Seasonal patterns in leaf reflectance red-edge characteristic," *Int. J. Remote Sens.* **12**(17), 1509–1523 (1991).
51. Miller, J. R, E. W Hare, and J. Wu "Quantitative characterization of the vegetation red edge reflectance 1. An inverted-Gaussian reflectance model," *Int. J. Remote Sens.* **11**(110), 1755–1773 (1990).
52. Belanger, M., J. R Miller, and M. G. Boyer, "Comparative relationships between some red edge parameters and seasonal leaf chlorophyll concentrations," *Canadian J. Remote Sens.* **21**(1), 16–21 (1995).
53. Gray, R. M, "Vector quantization," *IEEE ASSP Mag.* **1**, 4–29 (1984).
54. Gitelson, A. A and M. N Merzlak, "Remote estimation of chlorophyll content in higher plant leaves," *Int. J. Remote Sens.* **18**(12), 2691–2697 (1997).
55. Lichtenthaler, H. K, A. Gitelson, and M. Lang, "Non-destructive determination of chlorophyll content of leaves of a green and an aurea mutant of tobacco by reflectance measurements," *J. Plant Physiol.* **148**, 483–493 (1996).

Chapter 9
Hyperspectral Image Browser for Online Satellite Data Analysis and Distribution

9.1 Motivation for Web-Based Hyperspectral Image Analysis

Hyperspectral image data encapsulates a wealth of information. Remote sensing using hyperspectral instruments allows collection of the spectral signature of ground objects over large regions. These spectral signatures can be used to determine the chemical composition of the objects and thus allow applications such as mineral detection and vegetation health monitoring from airborne or spaceborne platforms. One of the greatest impediments to widespread use of hyperspectral remote sensing data is that there is no quick and easy method for potential users of hyperspectral data to locate and access suitable datasets. The datasets are large, and data providers do not generally advertise their products. As such, the user community has remained static, consisting of a small set of knowledgeable users.

The traditional approach for managing and advertising large holdings of remote sensing image data has been to use a cataloging system that maintains text metadata about each image in the archive together with a "quick-look" browse image. The browse image is typically a heavily subsampled and compressed version of the original image (in black and white or color). By performing a spatial or temporal search of the metadata within the catalog, potential users of the data can effectively discover potential datasets of interest. By then visualizing the corresponding browse images, users can evaluate each of the potential datasets to determine whether the imagery is of use in their application. The key to the success of this paradigm is that the metadata supports discovery of potential datasets, whereas the spatially subsampled browse images support quick evaluation of the spatial quality of the imagery. It is argued here that, until now, no such discovery and evaluation paradigm has existed for hyperspectral data archives.

Unlike traditional panchromatic or multispectral images, whose "information context" is contained primarily in the spatial domain, hyperspectral images are rich in spectral information. A typical hyperspectral dataset, often called a datacube, has a spatial extent, for example, of 640 lines by 512 pixels with 224 spectral bands. In other words, the spectral dimension is of the same order of magnitude as the spatial dimension. Cataloging systems that support only spatial or temporal queries of a metadata database miss the whole spectral domain, and thus users are unable to discover potential datasets of interest. By providing only spatially subsampled black-and-white or color browse images, users are unable to assess image quality or attributes in the spectral domain, and so are unable to evaluate the potential usefulness of the datacubes without ordering full copies of them. In other words, although the browse image adequately captures the spatial information in the full-resolution datacube, the spectral information is too heavily subsampled.

This chapter describes an innovative hyperspectral image browser (HIBR) system that overcomes the limitations of traditional archive catalogs. The HIBR via internet provides users of hyperspectral imagery with an effective data-discovery tool for exploring hyperspectral data archives, and for effectively evaluating the quality of potential datacubes before deciding whether or not to order the complete datasets.

The HIBR system has, at its core, a novel VQ compression scheme that packs the information of a hyperspectral datacube into VQ-compressed data (referred to as "VQube") that is a small fraction of the size of the original datacube. A unique property of the VQube is that any remote sensing algorithms can be applied to it directly to derive the image products without returning to the original format by decompressing it. Because a VQube is much smaller in size, the processing is extremely fast. The net gain is that the VQube allows both a reduction in data storage requirements and very fast processing of hyperspectral data using a wide selection of spectral processing algorithms. The HIBR system has the potential to revolutionize the user-data catalogs for future hyperspectral Earth-observation missions.

Although the VQ compression is necessarily a "lossy" process, one of its strengths is that the loss of spectral information is small and distributed across the spectrum. It maintains high spectral integrity even at a high CR. Results show that the image products obtained from spectral algorithm analysis applied to the VQube are within 2% or less than those derived from the original datacubes.

9.2 Web-Based Hyperspectral Image Browser and Analysis

The HIBR concept was initially invented by the author of this book together with colleagues at the Canadian Space Agency (CSA).[1] Under contract to CSA, MacDonald Dettwiler and Associates (MDA) subsequently developed

and extended its functionality to demonstrate the usefulness of the HIBR for rapidly browsing, selecting, and extracting geophysical product information from hyperspectral data archives. As part of this work, MDA expanded the functionality of the search methodology by linking the system to existing metadata browsers, narrowing the number of datacubes to be searched, and expanding the functionality of the browser implementation. The latter includes downloading compressed VQubes from a remote server via internet, viewing the decompressed data, performing spectral searches, and computing, analyzing, and displaying remote sensing products derived directly from the VQubes.

The fundamental concept behind the HIBR is the realization that hyperspectral data users require specialized tools to enable them to quickly and effectively discover and evaluate data holdings within a large hyperspectral image archive. These tools must allow users to evaluate the spectral content of individual datacubes over remote links, much like how today's users of multispectral satellite datasets use the Internet to search existing catalogs and preview browse images. The HIBR is such a system for hyperspectral data.

As mentioned in the previous section, the HIBR system is built on a novel VQ compression technique that compresses datacubes and packs the datacube information into VQ-compressed "VQubes" a fraction of the size of the original datacubes. Because compression ratios from 20:1 to 100+:1 are routinely achievable, VQubes are well suited to online storage and dissemination over the Internet. In essence, VQubes are to datacubes what quick-look browse images are to multispectral images.

The HIBR consists of two parts: a Java client running on a user's desktop, and a catalog server located at the image archiving facility. The server is responsible for searching through online VQubes and serving them up to the desktop client software. The client software provides a rich set of hyperspectral visualization and analysis tools that can be used to assess the suitability of a particular datacube. Figure 9.1 illustrates the overall HIBR concept.

The process begins with the initial data acquisition. Hyperspectral satellite and aircraft sensors collect large quantities of data, which have 2D spatial dimensions and one spectral dimension, producing a datacube. The raw datacubes are inserted into an archive typically consisting of storage libraries. The resulting hyperspectral data archives are huge, and it is difficult and time-consuming to check data quality and utility. Furthermore, the size of the datacubes means that there is no practical way to let potential users browse the data and thus make informed decisions about whether or not the data is suitable for their application.

To address the problem of data volume, the HIBR compressor reduces the data volumes to small manageable VQubes with minimal loss to the data

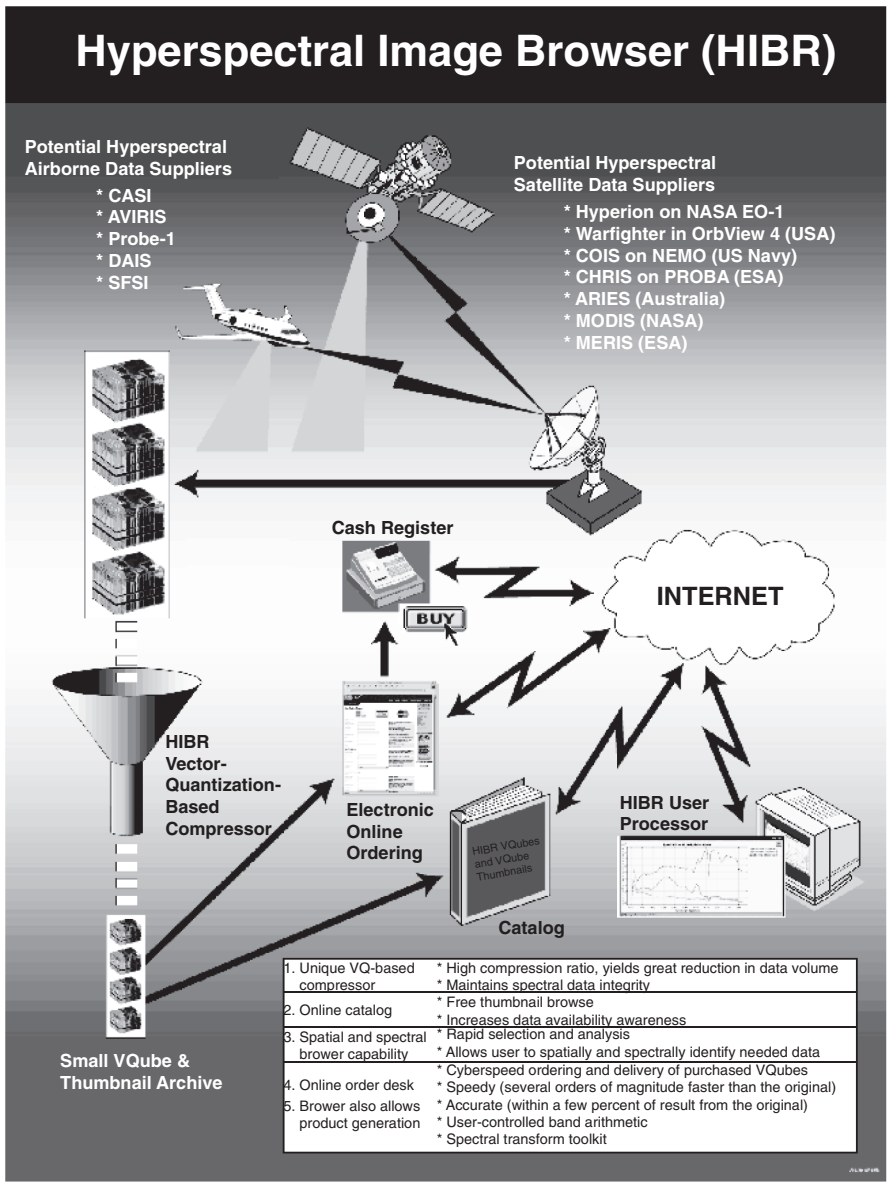


Figure 9.1 HIBR allows hyperspectral data users to remotely browse a hyperspectral data archive, helping them find datasets of potential interest and evaluate their applicability to the user's application. For a color version of this figure, see Plate 10 in the color plate section of this book.

information content. The HIBR-compressed VQubes are smaller and thus lend themselves well to online storage. Furthermore, the VQubes are faster to process and electronically transfer. The compressor can be applied to radiance data (for data quality checking by the data distributor) and to reflectance data

(for data dissemination). The conversion from radiance to reflectance can be performed on the compressed radiance data, as it has been shown that the reflectance data thus produced is of the same quality as would be achieved by performing atmospheric correction on the original noncompressed data and then compressing. This allows the atmospheric correction to be done much more rapidly than would otherwise be possible.

Once compressed, the VQubes are inserted into an online catalog. The catalogue provides much the same functionality as a traditional Earth-observation catalog with the ability to perform spatial and temporal searches of the test image metadata. There are two key differences, however: first, the HIBR supports a spectral search of the archive, thus allowing a user to identify datasets that contain spectral signatures of potential interest to the user. Second, instead of (or in addition to) offering traditional browse images, the HIBR provides compressed VQubes to the users' desktop browser. The user can then run the local HIBR applet within a browser to perform a wide range of hyperspectral image analysis and visualization tasks. In this manner, the user can evaluate whether the spectral content of the selected datacube is sufficient for the application's needs.

The user can order the full resolution dataset to perform detailed analysis after an appropriate datacube has been identified. The exact mechanism for ordering image data typically depends on the policies of individual data provider. However, the HIBR has been designed to, and fully supports, the notion of online ordering for compressed or noncompressed datasets.

9.3 HIBR Functions and Data Flow

Figure 9.2 shows the block diagram of the HIBR functions. The HIBR system contains three libraries: a local hyperspectral library, a benchmark hyperspectral library, and a compressed data library. The compression module provides compression functions including lossy compression (described earlier), lossless compression, browser communications, and applet server. The compression module exchanges data with the three libraries via the internal data bus and communicates with the browser module and service control module via transmission bus. The browser module provides functions of remote communications, decompression, visualization, remote processing, and a set of algorithms for application products. The service control module is the interface between the HIBR system, remote users, and the third-party data sources through the Internet. From the previous discussion, it is clear that the hyperspectral data compressor, the central web server, and the desktop hyperspectral image browser are the three key technologies that underpin the development of the HIBR.

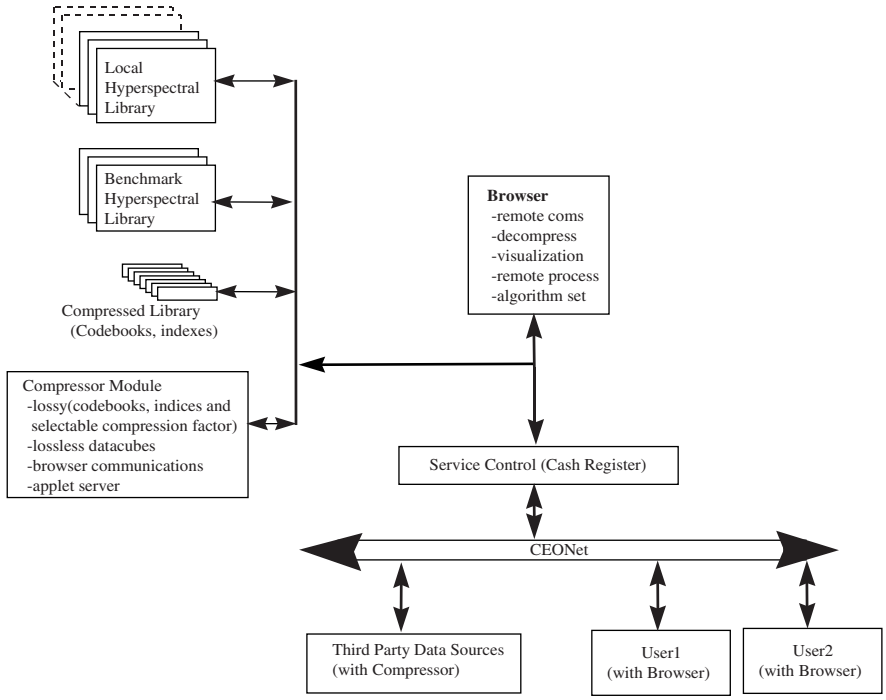


Figure 9.2 Block diagram of HIBR functions.

9.3.1 Hyperspectral data compressor

The HIBR is based on a high-data-compression-ratio software algorithm and is achieved through the use of a lossy compression technique that uses VQ. This algorithm forms the core of the HIBR system and is in fact the enabling technology that makes HIBR possible. The VQ compression algorithm has a number of attributes that are of particular importance to the HIBR application:

1. The technique supports large compression ratios while maintaining much of the spatial and spectral content of the original image. Because of this, it is practical to maintain an online archive of VQubes for all datacubes within a mission data holding.
2. While the compression is lossy, the algorithm distributes the compression errors across the spectrum, thus minimizing the local loss of spectral information. Tests on compressed VQubes show that the image products obtained from spectral algorithm analysis applied to the VQ-compressed datacubes are within 2% of those derived from the original noncompressed datacubes.
3. Decompression of the image is extremely fast, meaning VQubes can be rendered in real time on even low-end workstations and PCs.

4. The compressed data format is such that most traditional hyperspectral-image-processing algorithms (such as the SAM, spectral indices, etc.) can be applied directly to the compressed data. This results in an extraordinary shortening of image processing time because VQubes are extremely small and require no preliminary decompression time. Furthermore, because the complete dataset easily fits in memory, there is no need to process hundreds of megabytes of image data on disk (as is typically the case with datacubes). The result is that many traditionally computationally expensive algorithms run in real time.

As described in Chapter 4, VQ compression is composed of two steps: codebook training and codevector matching (also called encoding). The actual compression performance depends on the selected codebook training algorithm and the selected hyperspectral image data training set. Three codebook training algorithms are used in the HIBR:

1. 3DVQ, a conventional VQ data compression for 3D hyperspectral datacubes using the Linde–Buzo–Gray (LBG) algorithm,[2]
2. 3DVQ using a spectral-feature-based binary coding (SFBBC) fast algorithm,[3] and
3. Multiple-subcodebook algorithm (MSCA).[4]

The LBG algorithm generates codevectors by iteratively reclustering the data, using the Euclidean distance to determine to which cluster each image vector should be assigned and then taking the centroid of each cluster as a codevector. This is a very computationally intensive procedure.

The SFBBC algorithm first computes a spectral "feature" vector for each spectral vector in a datacube. Codevectors are again taken as cluster centroid, except that the use of feature vectors allows the use of the Hamming distance as the distance measure instead of the Euclidean distance. The former involves a bitwise exclusive-or operation as opposed to the squared-error computation required for the Euclidean distance, and thus it is much faster than the conventional LBG algorithm.

The MSCA algorithm requires that the scene of the datacube be presegmented using a remote sensing technique such as spectral index, for example, the normalized difference vegetation index (NDVI). Subcodebooks are then generated independently (using either LBG or SFBBC methods) for each segment (training subset). Working on small segments at a time significantly speeds up codebook generation. The segmentation "class map" is saved for use in encoding.

Five VQ codevector-matching algorithms are used in the HIBR:

1. 3D vector quantization (3DVQ),[3]
2. Fast 3DVQ (F3DVQ),[6]
3. Correlation vector quantization (CVQ),[5]

4. Fast CVQ (FCVQ),[6] and
5. Spectral-index-based multiple-subcodebook algorithm (MSCA).[4]

The 3DVQ algorithm encodes each spectral vector in a datacube with the nearest codevector in the codebook, using the Euclidean distance as the distance measure. The index of each matched codevector is recorded in an index map, which is transmitted with the codebook in the compressed VQ product.

The F3DVQ algorithm encodes each spectral vector in a datacube with the nearest codevector in the codebook, using the Hamming distance between their spectral feature vectors (i.e., SFBBC vectors) as the distance measure. As above, the match indices are recorded in an index map.

The CVQ algorithm uses a 2×2 moveable window to check the Euclidean distance between a vector to be encoded and three encoded vectors in the window. If the smallest distance is less than a given threshold, the codevector index of the closest encoded vector is assigned to the vector instead of searching through the entire codebook; otherwise, the entire codebook is searched to find the best match.

The FCVQ algorithm operates similar to the CVQ algorithm except that the Hamming distance between spectral feature vectors is used in place of the Euclidean distance between spectral vectors.

In the case where the MSCA was used in training, the encoding step uses the presegmentation class map to ensure that only the subcodebook relevant to the segment in which the current vector lies is searched for the best-matching codevector. The best match can be found using any one of the previous techniques.

The decompression stage is a simple lookup procedure, where the index map dictates which codevector to apply at each spatial location in the decompressed image.

9.3.2 Hyperspectral catalog web server

The HIBR exploits a central web server to which the user can connect to query the hyperspectral data holdings within an archive. To facilitate the traditional temporal and spatial searches of data holdings, HIBR can be integrated into an existing image catalog system. This allows a data-archiving facility to seamlessly integrate hyperspectral data with their existing Earth-observation data holdings. The primary difference with the catalog is that instead of presenting the user with a color browse image of the hypercube, the catalog will download the VQube to the user's desktop computer to facilitate hyperspectral image browsing.

A unique aspect of HIBR is that it also supports spectral querying of hyperspectral datacubes within the data archive. This is an extraordinarily powerful tool whereby the user can select a reference spectral signature (either

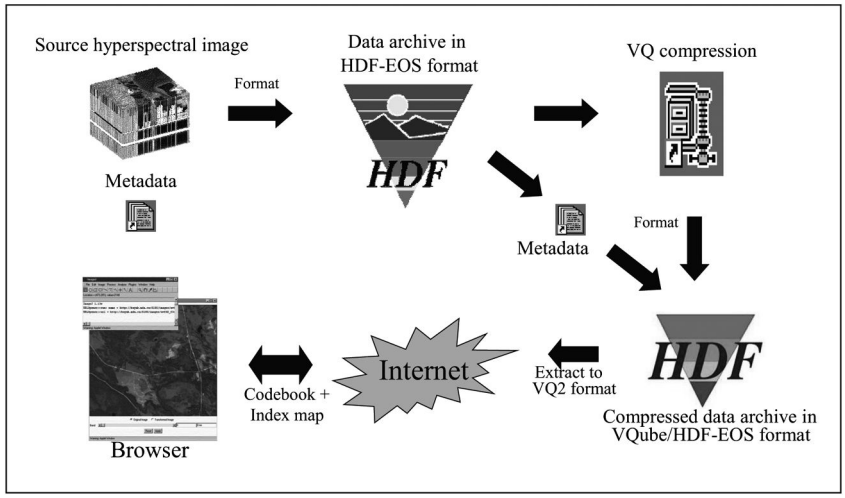


Figure 9.3 The flow of hyperspectral data from the sensor to the user's desktop.

from the integrated USGS spectral signature library or supplied by the user), and run a SAM algorithm on all of the VQubes of interest. Using this technique, the user can quickly identify potential datasets that contain a specific spectrum (e.g., of a mineral of interest). The feature is made possible by the fact that the VQubes retain much of the spectral information contained in the original datacubes but with a much lower data volume. All functionality within the catalog server is accessed through a standard web browser.

9.3.3 Overall data flow

Figure 9.3 illustrates the overall flow of hyperspectral data through the HIBR system. Data received from a hyperspectral satellite is first archived in the HDF-EOS format. These files are then compressed using the VQ compressor. The output compressed data (VQubes) also use the HDF format; these files can be browsed over the Internet using the HIBR browser.

9.4 User Scenarios

The most-visible part of the HIBR system is the hyperspectral image browser itself, a Java applet that runs within a standard web browser on the users' desktop. The browser is actually a relatively complex hyperspectral-image analysis tool that has been tuned to operate on compressed VQubes. When users identify a potential dataset of interest in the catalog, they click on an icon that downloads the corresponding VQube and initiates the browser applet. Users are then free to browse through the dataset, including applying a number of standard analysis algorithms.

The HIBR browser allows the user to preview available hyperspectral datacubes. The user may perform preliminary visualization and simple spectral searches (user product search) during this preview. All operations are performed on the lossy representation of the data. Once the user has identified the datacube(s) of interest, the full datacube may be ordered from the appropriate source. Figure 9.4 shows scenarios under which the browser can be operated.

There are two main branches of operation that a user of the HIBR may take to process the datacube. Operations include data visualization using one of several visualization tools and user product searches.

9.5 Hyperspectral Image Browser Operations

In order to meet the design requirements, Java has been selected as the development language. The advantages of Java include its readiness for Internet applications and its platform independence. Furthermore, the existing prototype was developed in this language. As a result, the entire HIBR system is built using the Java Development Kit. Four groups of software have been combined to create the current HIBR system.

The HIBR is a functional hyperspectral image browser that allows users to browse VQ-compressed hyperspectral image cubes. A suite of applications and functions was implemented that allows the user to examine and visualize the data. Users can perform many operations by combining these functions and applications.

9.5.1 HIBR visualization

The visualization module contains core functionality that allows the user to visualize and examine the datacube:

1. Compose RGB,
2. Display image by band and wavelength,
3. Zoom into a region,
4. Plot spectrum at a specified spatial location,
5. Plot spectrum with confidence measure display,
6. Band arithmetic (addition, subtraction, multiplication, and division),
7. Contrast stretching,
8. Apply different color maps,
9. Display transform image, and
10. Provide interactive cursor location and value display.

The HIBR provides operation of visualizing any image band or band combination in black and white or color, including full roam and zoom

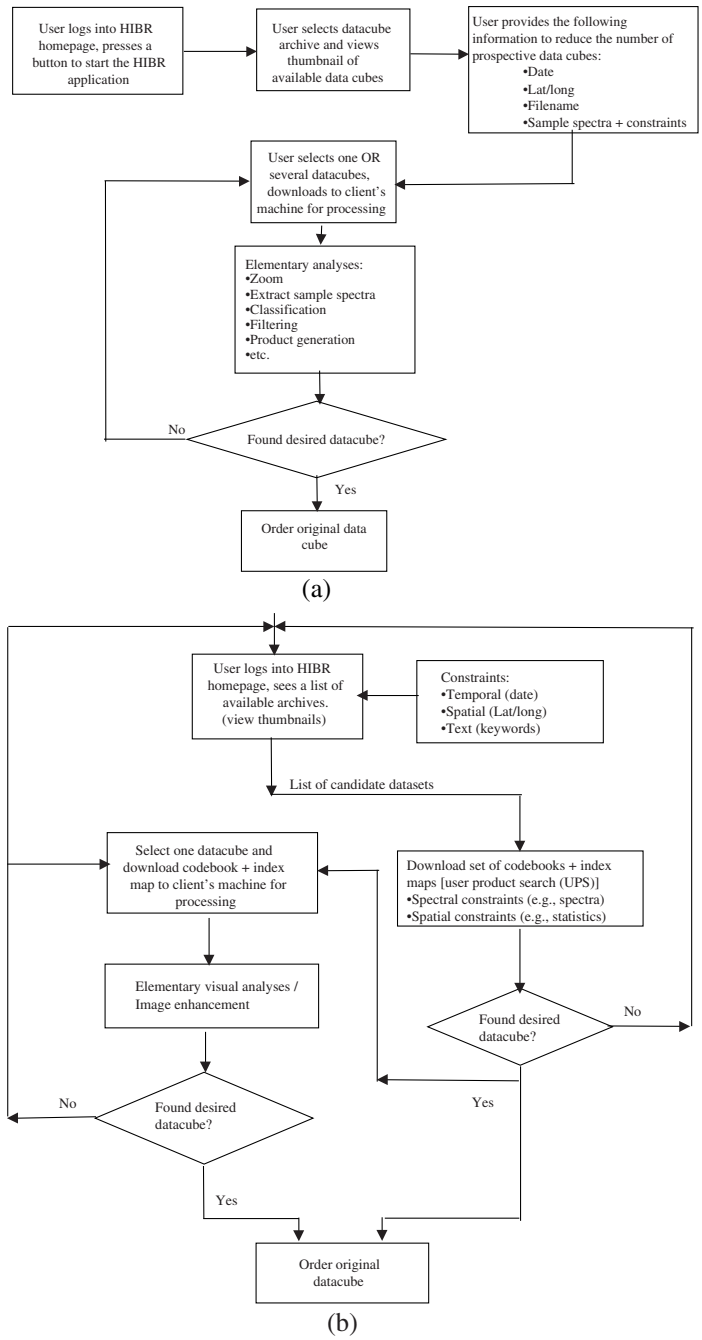


Figure 9.4 (a) HIBR user data visualization and (b) user product-search scenarios.

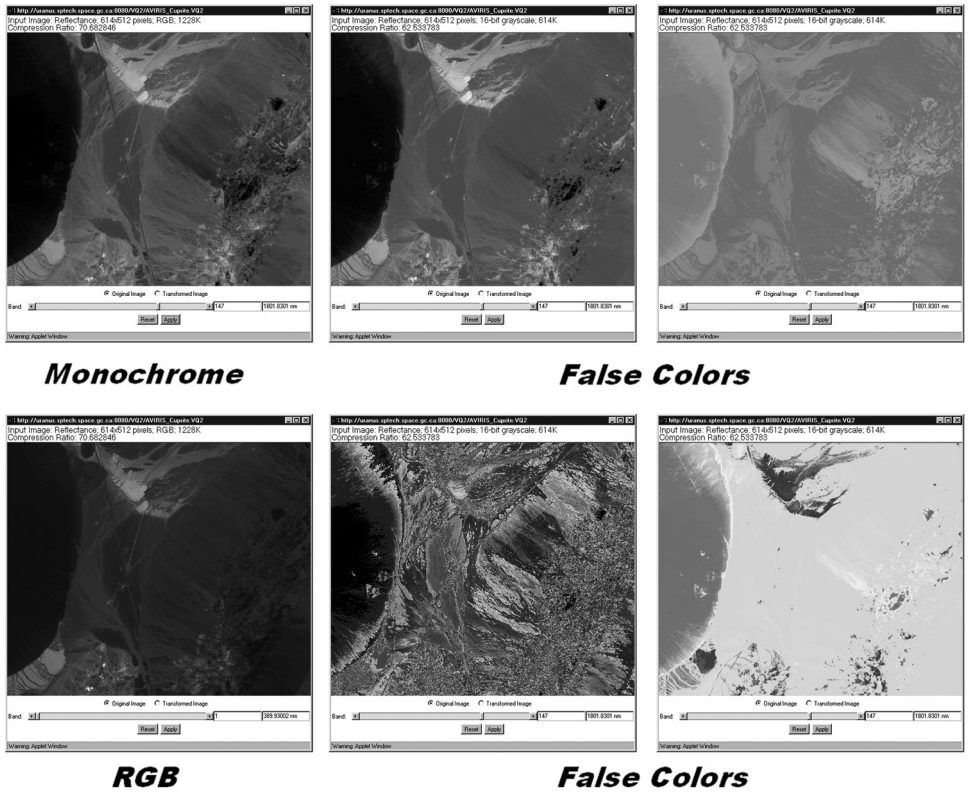


Figure 9.5 HIBR function for visualizing monochrome, RGB, and false color images. For a color version of this figure, see Plate 11 in the color plate section of this book.

capabilities, contrast stretch, and pseudo-color editing. Figures 9.5 and 9.6 illustrate examples of visualizing monochrome, RGB, and false color, as well as processed images of hyperspectral datacubes.

9.5.2 User product search

The user product-search module contains core functionality that allows the user to search the list of available datacubes by providing a reference spectrum:

1. The user can select a spectrum from a given library,
2. The user can search all available datacubes for the selected spectrum,
3. The search results are displayed as a list of the percentage of matching pixels per image, and
4. The user can display the search results for each image in image format, where pixel value equals the computed value of the cosine of the spectral angle between the image spectrum at the given pixel and the selected library spectrum. (The results are in the range [0.0, 1.0], where a value of 1.0 indicates a perfect match).

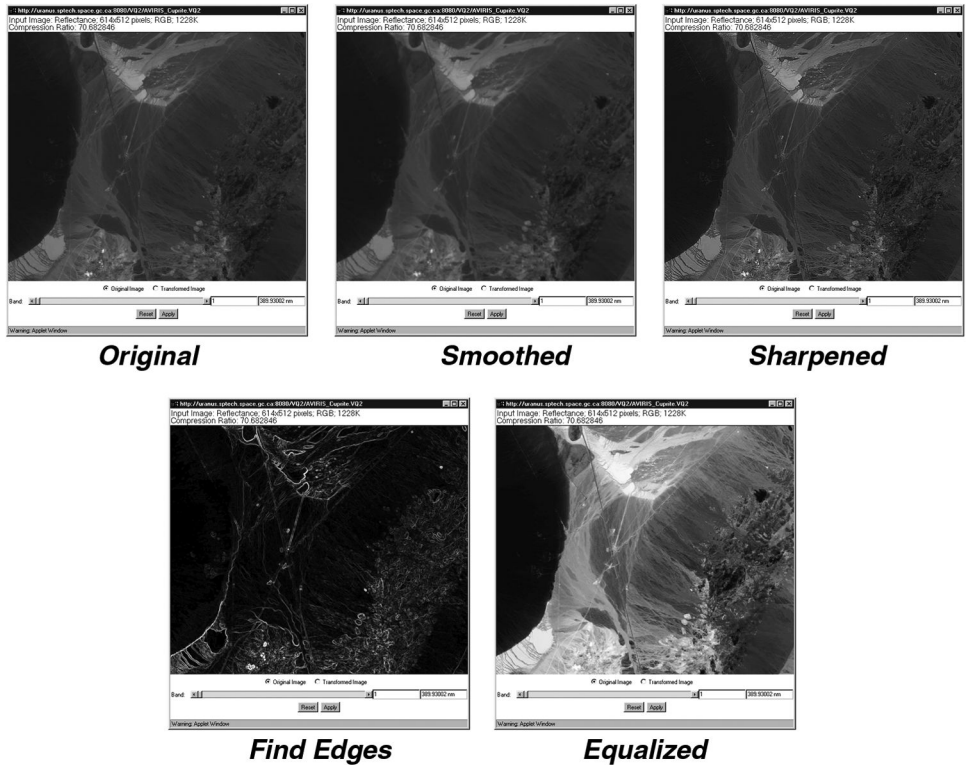


Figure 9.6 HIBR function for visualizing processed images. For a color version of this figure, see Plate 12 in the color plate section of this book.

Figure 9.7 illustrates a typical application with the HIBR browser to search for interested hyperspectral datacubes using the SAM as a means. A reference spectrum (sometimes referred to as a "seed spectrum") is used to match the hyperspectral datacubes in the archive. The search is extremely fast because it is carried out in the VQubes.

9.5.3 User product generation

The user product generation module contains core functionality that allows the user to generate products from the list of available datacubes:

1. NDVI,
2. SAM (by using a spectrum from a library or by using a spectrum from the datacube),
3. Vogelmann red-edge bio-indicator, and
4. Inverse Gaussian red-edge biophysical indicator parameters.

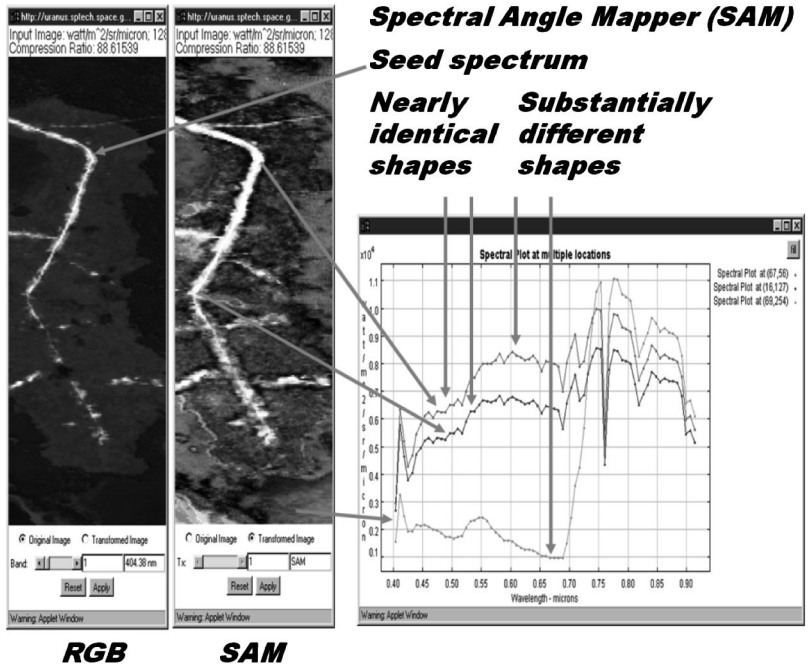


Figure 9.7 A seed spectrum is used to search for interested hyperspectral datacubes using the SAM as a means.

9.5.4 HIBR graphical user interface

The main display of the HIBR system is shown in Fig. 9.8, which consists of a hypertext markup language (HTML) page from which the main Java applet program of HIBR may be launched.

The Java applet program is the main controller of the HIBR system. Two pull-down menus are also provided on the HTML page: the first pull-down menu lists the compressed hyperspectral image cubes available for display, and the second menu lists available reference spectra for the user product search. Below the second menu is a message area for displaying messages such as the search results from the user product search.

Figure 9.9 shows the image viewer of the HIBR system, divided into six areas:

1. Pull-down-menu bar: These menus allow the user to select from the available functions.
2. Main image display: This is where the image or transformed image is displayed.
3. Position information display and selection: The values of the cursor position and the pixel intensity at this position are displayed in this area. A toggle button is provided for the user to select between having the position

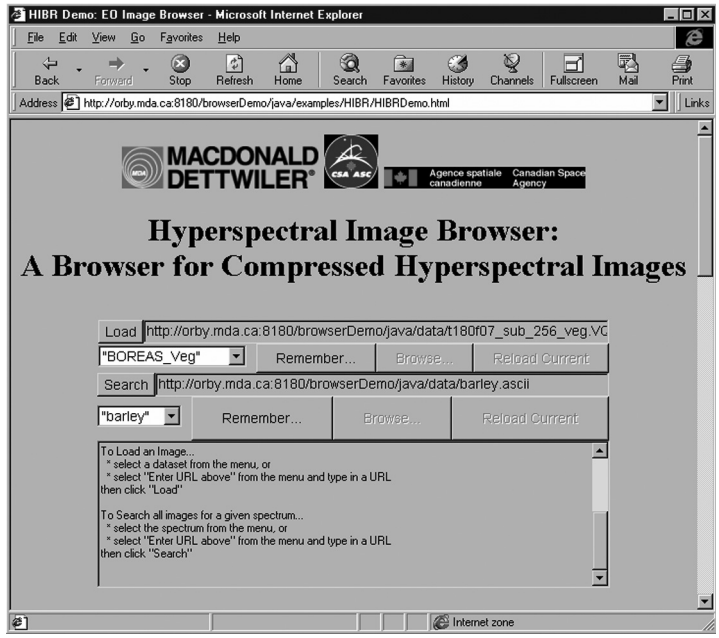


Figure 9.8 Main page of the HIBR.

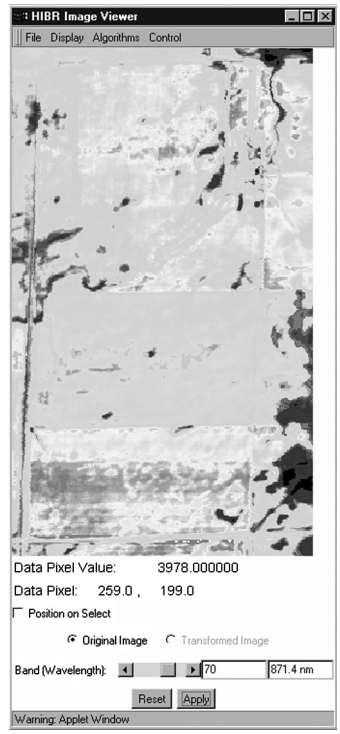


Figure 9.9 HIBR image viewer. For a color version of this figure, see Plate 13 in the color plate section of this book.

of the cursor constantly updated when the user uses the mouse to scroll around the image display, or only updating the information when the user clicks the left mouse button at a selected position.
4. Toggle buttons to display transformed and original images: These are provided to allow the user to switch between displaying the original image and the transformed (product) image.
5. Band-selection slider: The user can select which band to display by moving the band slider. The current band number and wavelength are listed alongside the slider. The display is not updated until the "Apply" button is clicked to avoid the user having to wait for the display to be updated until the choice is validated.
6. Application buttons: Two buttons are provided. The "Reset" button resets the current band selection to "1," and the "Apply" button updates the image display area to show the currently selected band.

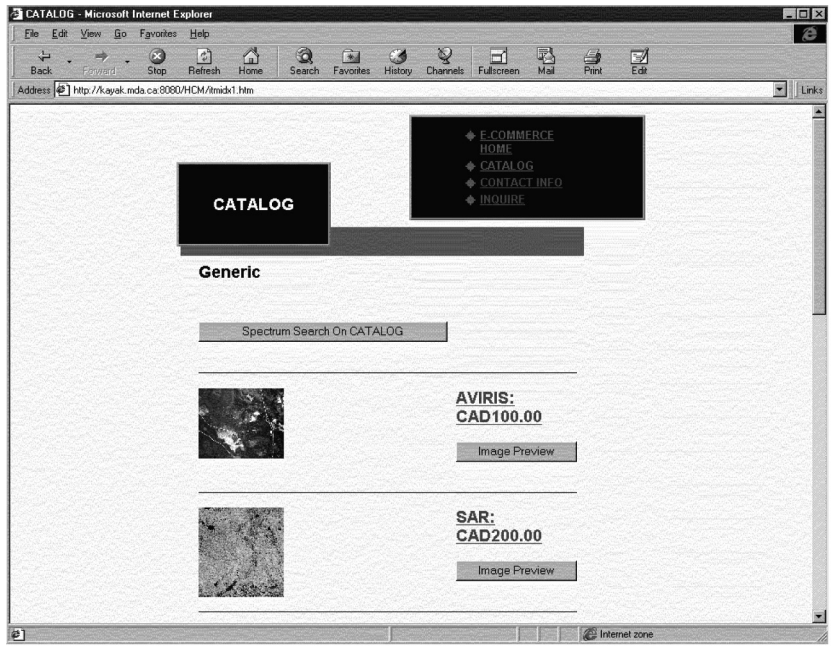
Figure 9.10 illustrates a typical set of screens a user would encounter when browsing a hyperspectral archive. The left side depicts the traditional "results" screen that shows the available images in a data holding. The screenshot on the right illustrates the use of the spectral search interface in the catalog. Figure 9.11 illustrates a typical session with the HIBR browser. As the figure shows, the browser provides a wide range of data visualization tools.

9.6 Summary

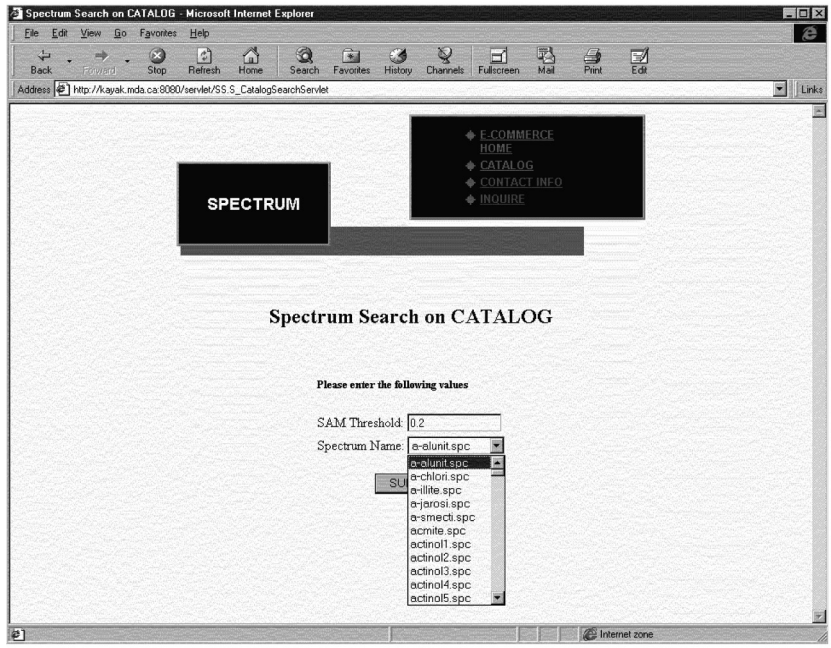
This chapter presents an overview of the HIBR, a novel web-based hyperspectral-image browsing system. The HIBR provides hyperspectral image users with effective tools to aid them in the discovery and evaluation of potentially useful hyperspectral datacubes within large data archives, thus eliminating one of the major impediments to the widespread exploitation of hyperspectral image data.

Key to the success of the HIBR is an innovative hyperspectral compression algorithm that enables compact VQubes to be generated from large datacubes. VQubes significantly reduce data storage requirements, thus enabling online archives and fast network transfers, and allowing for the fast processing of remote sensing products directly in the compressed data format. Furthermore, VQubes retain most of the spectral information contained in the original datacube, allowing effective evaluation of the suitability of the dataset.

It is anticipated the HIBR technology will be of great interest to hyperspectral data users and principal investigators to facilitate their access to datasets within existing image archives. The HIBR will also be of interest to data providers and managers of large data archives, allowing them to effectively advertise their hyperspectral data holdings. Finally, the HIBR will



(a)



(b)

Figure 9.10 The HIBR supports traditional catalog queries that result in sets of candidate datasets, as shown in (a), as well as spectral queries that allow the user to compare all datasets of interest to a particular reference spectral signature, as shown in (b).

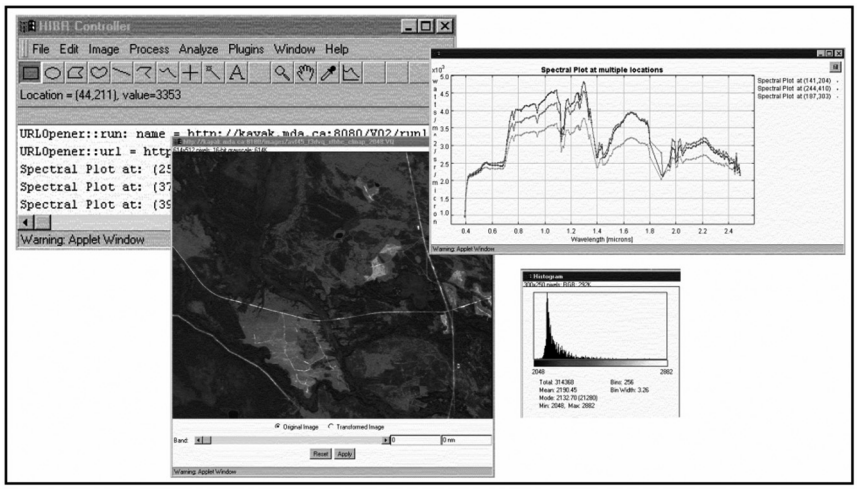


Figure 9.11 The HIBR includes a rich set of hyperspectral image analysis tools that run as a Java applet within a standard Internet browser. These tools allow users to quickly evaluate the suitability of a given dataset to a particular application.

be of interest to end users who wish to perform fast, quantitative analysis of datacubes to identify promising areas where full-resolution analysis is justified.

The HIBR has the potential to revolutionize the user data catalogs for future hyperspectral Earth-observation missions, including the Hyperion instrument on the EO-1 satellite, the MODIS instrument on the EOS-AM satellite, and future hyperspectral instruments.

References

1. Hollinger, A., S.-E. Qian, and D. Williams, "System for Interactive Visualization and Analysis of Imaging Spectrometer Datasets Over a Wide-Area Network," U.S. Patent No. 6,546,146 B1, issued on April 8, 2003.
2. Gray, R. M., "Vector quantization," *IEEE ASSP Mag.* **1**, 4–29 (1984).
3. Qian, S.-E., A. B. Hollinger, D. Williams, and D. Manak, "Fast 3D data compression of hyperspectral imagery using vector quantization with spectral-feature-based binary coding," *Opt. Eng.* **35**(11), 3242–3249 (1996) [doi: 10.1117/1.601062].
4. Qian, S.-E., A. B. Hollinger, D. Williams, and D. Manak, "Vector quantization using spectral index based multiple sub-codebooks for hyperspectral data compression," *IEEE Trans. Geosci. Remote Sens.* **38**, 1183–1190 (2000).

5. Qian, S.-E., A. B. Hollinger, D. Williams, and D. Manak, "A near lossless 3-dimensional data compression system for hyperspectral imagery using correlation vector quantization," *Proc. 47th International Astronautical Congress*, Beijing (1996).
6. Qian, S.-E., A. B. Hollinger, D. Williams, and D. Manak, "3D data compression systems based on vector quantization for reducing the data rate of hyperspectral imagery," *Appl. Photonic Technol*. **2**, 641–654, Plenum Press, New York (1997).

Index

G

S

Dr. Shen-En Qian is a senior scientist and technical authority at the Canadian Space Agency. He is an internationally recognized expert in optical spacecraft payloads, space technologies for satellite missions and deep space exploration, remote sensing, satellite image processing and analysis, onboard satellite data compression, data handling, and the enhancement and international spacecraft data standards. He has been working in these areas for 30 years. He holds nine U. S. patents, three European patents, and several pending patents. He is the author of two reference books and chapters in three others. He has published over 100 scientific papers and produced 100 unpublished proprietary technical reports. He is a fellow of SPIE and a senior member of the Institute of Electrical and Electronics Engineers (IEEE).

Dr. Qian received his B. Eng. in industrial electrical automation in 1982, his M. S. in optoelectronics in 1985, and his Ph. D. in telecommunication and electronic systems in 1990. He was the recipient of the Marie Curie Award (European Community International Scientific Cooperation Program). He received the Canadian Government Invention Award for his multiple patents for satellite missions, and he was twice the recipient of the Director Award from the Federal Government of Canada for his outstanding contributions to R&D in space technology and satellite missions in the government laboratory.

OPTICAL SATELLITE

Data Compression and Implementation

Shen-En Qian

This book provides a global review of optical satellite image and data compression theories, algorithms, and system implementations. Consisting of nine chapters, it describes a variety of lossless and near-lossless data-compression techniques and three international satellite-data-compression standards. The author shares his firsthand experience and research results in developing novel satellite-data-compression techniques for both onboard and on-ground use, user assessments of the impact that data compression has on satellite data applications, building hardware compression systems, and optimizing and deploying systems. Written with both postgraduate students and advanced professionals in mind, this handbook addresses important issues of satellite data compression and implementation, and it presents an end-to-end treatment of data compression technology.

Want to learn more about the calibration and enhancement of spaceborne optical sensors and methods for image enhancement and fusion? SPIE Press presents this book's companion text, *Optical Satellite Signal Processing and Enhancement,* also written by **Shen-En Qian**.

ISBN 9780819497871
90000
9 780819 497871

SPIE.

P.O. Box 10
Bellingham, WA 98227-0010

ISBN: 9780819497871
SPIE Vol. No.: PM241