Lossy Hyperspectral image compression on NVidia™ Graphics Processing Units

Antonio Plaza[a], Javier Plaza[a], Sergio Sánchez[a]

[a]Department of Technology of Computers and Communications
University of Extremadura, Avda. de la Universidad s/n
E-10071 Cáceres, Spain

ABSTRACT

Hyperspectral imaging instruments are capable of collecting hundreds of images, corresponding to different wavelength channels, for the same area on the surface of the Earth. For instance, NASA is continuously gathering imagery data with instruments such as the Jet Propulsion Laboratory's Airborne Visible-Infrared Imaging Spectrometer (AVIRIS), able to record the visible and near-infrared spectrum (wavelength region from 0.4 to 2.5 micrometers) of the reflected light of an area 2 to 12 kilometers wide and several kilometers long, using 224 spectral bands. The resulting multidimensional data volume typically comprises several GBs per flight. We have developed a computationally efficient approach for lossy compression of remotely sensed hyperspectral images that retains the relevant information for analyzing the hyperspectral data with sub-pixel precision. The proposed methodology has been implemented, using the compute unified device architecture (CUDA), on an NVidia™ GeForce 8800 GTX GPU, achieving speedups in the order of 26x when compared to an optimized implementation of the same code in a dual-core CPU.

Keywords: Hyperspectral data lossy compression, spectral mixture analysis, endmember extraction, linear spectral unmixing, graphics processing units (GPUs).

1. INTRODUCTION

Hyperspectral imaging instruments are capable of collecting hundreds of images, corresponding to different wavelength channels, for the same area on the surface of the Earth.[1] For instance, NASA is continuously gathering imagery data with instruments such as the Jet Propulsion Laboratory's Airborne Visible-Infrared Imaging Spectrometer (AVIRIS),[2] able to record the visible and near-infrared spectrum (wavelength region from 0.4 to 2.5 micrometers) of the reflected light of an area 2 to 12 kilometers wide and several kilometers long using 224 spectral bands. The resulting hyperspectral data cube[3] is a stack of images (see Fig. 1) in which each pixel (vector) is represented by a spectral signature or 'fingerprint' that uniquely characterizes the underlying objects, and the resulting multidimensional data volume typically comprises several GBs per flight.[4] This introduces the need for hyperspectral data compression.[5]

Spectral mixture analysis (SMA) is a popular tool for characterizing mixed pixels in remotely sensed hyperspectral data sets.[6] The standard processing chain for linear SMA comprises two stages: 1) extraction of pure spectral signatures (endmembers), and 2) estimation of the fractional abundance of each endmember in each pixel of the scene. The use of spectral endmembers allows one to deal with the problem of mixed pixels. For instance, it is likely that the pixel labeled as 'vegetation' in Fig. 1 actually comprises a mixture of vegetation and soil. In this case, the measured spectrum can be decomposed into a linear combination of pure spectral endmembers of soil and vegetation, weighted by abundance fractions that indicate the proportion of each endmember in the mixed pixel. Let us assume that a remotely sensed hyperspectral scene with n bands is denoted by $\mathbf{F}$, in which the pixel at the discrete spatial coordinates (i,j) of the scene is represented by a vector $\mathbf{X}(i,j) = [x_1(i,j), x_2(i,j), \cdots, x_n(i,j)] \in \Re^n$, where $\Re$ denotes the set of real numbers in which the pixel's spectral response $x_k(i,j)$ at sensor channels $k = 1,\ldots,n$ is included. Under the linear mixture model assumption, each pixel vector in the original scene can be modeled using the following expression:

Send correspondence to Antonio J. Plaza:
E-mail: aplaza@unex.es; Telephone: +34 927 257000 (Ext. 51662); URL: http://www.umbc.edu/rssipl/people/aplaza

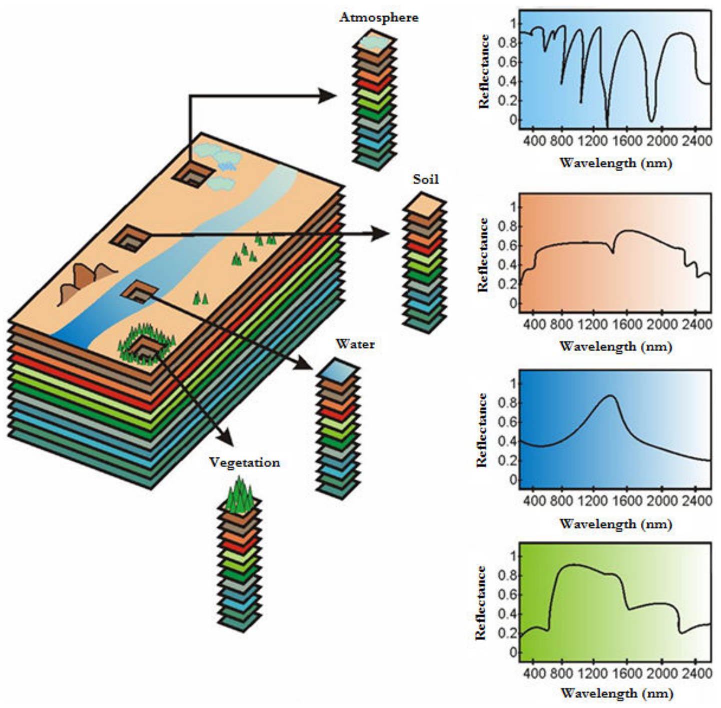

Figure 1. Concept of hyperspectral imaging.

$$\mathbf{X}(i,j) = \sum_{z=1}^{p} \Phi_z(i,j) \cdot \mathbf{E}_z + \mathbf{n}(i,j), \quad (1)$$

where $\mathbf{E}_z$ denotes the spectral response of endmember z, $\Phi_z(i,j)$ is a scalar value designating the fractional abundance of the endmember z at the pixel $\mathbf{X}(i,j)$, p is the total number of endmembers, and $\mathbf{n}(i,j)$ is a noise vector. The solution of the linear spectral mixture problem described in (1) relies on a successful estimation of how many endmembers, p, are present in the input hyperspectral scene $\mathbf{F}$, and also on the correct determination of a set $\{\mathbf{E}_z\}_{z=1}^{p}$ of endmembers and their correspondent abundance fractions $\{\Phi_z(i,j)\}_{z=1}^{p}$ at each pixel $\mathbf{X}(i,j)$. Two physical constrains are generally imposed into the model described in (1), these are the abundance non-negativity constraint (ANC), i.e., $\Phi_z(i,j) \geq 0$, and the abundance sum-to-one constraint (ASC), i.e., $\sum_{z=1}^{p} \Phi_z(i,j) = 1$.[3]

Over the last decade, several algorithms have been developed for automatic or semi-automatic extraction of spectral endmembers directly from an input hyperspectral data set. Among the classic techniques for this purpose[7] is the pixel purity index (PPI),[8] which makes use of convex geometry concepts to identify the endmembers as the most extreme pixels in the n-dimensional data cloud. However, one of the distinguishing properties of hyperspectral data is the multivariate information coupled with a two-dimensional (pictorial) representation amenable to image interpretation. In order to accommodate spatial information into the endmember searching process, several methods have been proposed.[9] One of such methods is the automatic morphological endmember extraction (AMEE) algorithm,[10] which uses extended mathematical morphology[11] concepts to integrate the spatial and the spectral information.

A processing chain made up of endmember extraction followed by spectral unmixing can be computationally expensive due to several reasons, such as the extremely large size and spectral resolution of the hyperspectral images, which calls for appropriate compression frameworks that can significantly reduce data volume while retaining the relevant information needed for data interpretation. In this work, we develop a computationally efficient approach for lossy compression of hyperspectral scenes which has been specifically tuned to preserve the relevant information required in SMA applications. The proposed method adopts the standard SMA processing chain as the lossy compression framework itself, and has been implemented in graphics processing units (GPUs),[12] an exciting development in the field of commodity computing.

The speed of graphics hardware doubles approximately every six months, which is much faster than the improving rate of the CPU. Currently, state-of-the-art GPUs deliver peak performances in the order of 300 Gflops, which is more than one order of magnitude over high-end micro-processors.[13] The ever-growing computational requirements introduced by hyperspectral imaging applications can fully benefit from this type of specialized hardware and take advantage of the compact size and relatively low cost of these units, which make them appealing for onboard data processing at much lower costs than those introduced by other hardware devices.[14] Specifically, our proposed lossy compression approach based on SMA can achieve very high compression ratios when applied to standard hyperspectral data sets, and can also retain the relevant information required for spectral unmixing in a computationally efficient way, achieving speedups in the order of 26 on a NVidia^{TM} GeForce 8800 GTX graphic card when compared to an optimized implementation of the same code in a dual-core CPU, as will be shown by experimental results in this paper.

The remainder of the paper is organized as follows. Section 2 describes the endmember extraction (PPI, AMEE) and spectral unmixing (FCLSU) methods used in this work to devise a lossy compression framework, described in Section 3. Section 4 describes a GPU-based implementation of the proposed lossy compression framework. Section 5 provides an experimental validation in the context of spectral unmixing applications, by drawing comparisons between the proposed method and other state-of-the-art compression techniques. This section also provides an experimental assessment of the parallel efficiency of the proposed GPU-based implementation. Finally, section 5 concludes with some remarks and hints at plausible future research.

2. ENDMEMBER EXTRACTION AND SPECTRAL UNMIXING METHODS

This section describes two endmember extraction algorithms (PPI, AMEE) and a fully constrained linear spectral unmixing (FCLSU) algorithm that will be used in the following section to develop a lossy compression strategy for hyperspectral image data.

2.1 Pixel purity index

One of the most successful algorithms for automatic endmember extraction in the literature has been the pixel purity index (PPI) algorithm.[8] The algorithm proceeds by generating a large number of random, n-dimensional unit vectors called "skewers" through the dataset. Every data point is projected onto each skewer, and the data points that correspond to extrema in the direction of a skewer are identified and placed on a list (see Fig. 2). As more skewers are generated, the list grows, and the number of times a given pixel is placed on this list is also tallied. The pixels with the highest tallies are considered the final endmembers.

The inputs to the algorithm are a hyperspectral data cube $\mathbf{F}$ with n dimensions; a maximum number of endmembers to be extracted, E; the total number of random skewers to be generated during the process, T; a cut-off threshold value, t_v, used to select as final endmembers only those pixels that have been selected as extreme pixels at least t_v times throughout the PPI process; and a threshold angle, t_a, used to discard redundant endmembers during the process. The output of the algorithm is a set of p final endmembers $\{\mathbf{E}_z\}_{z=1}^{p}$. The algorithm can be summarized as follows:

1. *Skewer generation.* Produce a set of T randomly generated unit vectors $\{\mathbf{skewer}_k\}_{k=1}^{T}$.

2. *Extreme projections.* For each $\mathbf{skewer}_k$, all sample pixel vectors $\mathbf{X}(i,j)$ in the original data set $\mathbf{F}$ are projected onto $\mathbf{skewer}_k$ via dot products to find sample vectors at its extreme (maximum and minimum) projections, thus forming an extrema set for $\mathbf{skewer}_k$ which is denoted by $S_{extrema}(\mathbf{skewer}_k)$. The dot products are calculated as follows:

$$|\mathbf{X}(i,j)\cdot\mathbf{skewer}_k| = \frac{\mathbf{X}(i,j)\cdot\mathbf{skewer}_k}{\|\mathbf{X}(i,j)\|\cdot\|\mathbf{skewer}_k\|} \tag{2}$$

Despite the fact that a different $\mathbf{skewer}_k$ would generate a different extrema set $S_{extrema}(\mathbf{skewer}_k)$, it is very likely that some sample vectors may appear in more than one extrema set. In order to deal with this

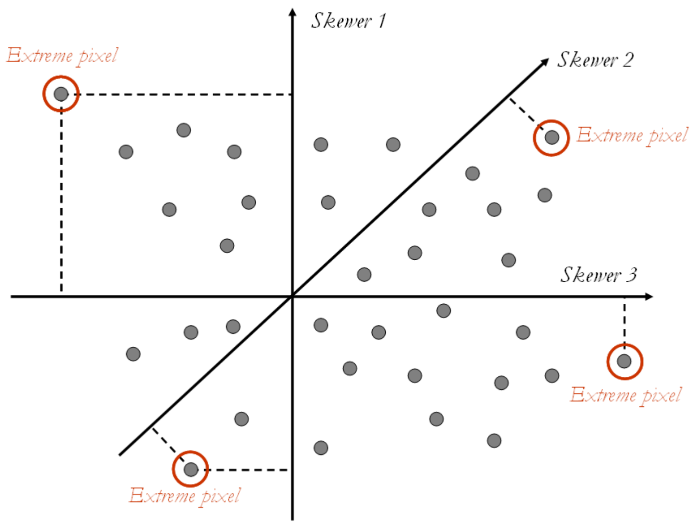

Figure 2. Toy example illustrating the performance of the PPI endmember extraction algorithm in a 2-dimensional space.

situation, we define an indicator function of a set S, denoted by $I_S(\mathbf{X}(i,j))$, to denote membership of an element $\mathbf{X}$(i,j) to that particular set as follows:

$$I_S(\mathbf{X}(i,j) = \begin{Bmatrix} 1 \text{ if } \mathbf{X}(i,j) \in S \\ 0 \text{ if } \mathbf{X}(i,j) \notin S \end{Bmatrix} \tag{3}$$

3. *Calculation of PPI scores.* Using the indicator function above, we calculate the PPI score associated to the sample pixel vector $\mathbf{X}(i,j)$ (i.e., the number of times that given pixel has been selected as extreme in step 2) using the following equation:

$$N_{PPI}(\mathbf{X}(i,j)) = \sum_{k=1}^{T} I_{S_{extrema}(\mathbf{skewer}_k)}(\mathbf{X}(i,j)) \tag{4}$$

4. *Endmember selection.* Find the pixels with value of $N_{PPI}(\mathbf{X}(i,j))$ above t_v, and form a unique set of endmembers $\{\mathbf{E}_z\}_{z=1}^{p}$ by calculating the spectral angle distance (SAD)[3] for all possible vector pairs and discarding those pixels which result in an angle value below t_a. It should be noted that the SAD between a pixel vector $\mathbf{X}(i,j)$ and a different pixel vector $\mathbf{X}(i',j')$ is defined as follows:

$$\text{SAD}(\mathbf{X}(i,j), \mathbf{X}(i',j')) = \cos^{-1} \frac{\mathbf{X}(i,j) \cdot \mathbf{X}(i',j')}{\|\mathbf{X}(i,j)\| \cdot \|\mathbf{X}(i',j')\|} \tag{5}$$

It should be noted that, although other spectral similarity metrics have been used in hyperspectral imaging research,[3] the SAD is widely regarded as a standard spectral similarity metric in remote sensing operations, mainly because it is invariant under the multiplication of the input vectors by constants and, consequently, is invariant to unknown multiplicative scalings that may arise due to differences in illumination and sensor observation angle.

2.2 Automatic morphological endmember extraction

The automatic morphological endmember extraction (AMEE) algorithm[10] is based on extended mathematical morphology concepts. Mathematical morphology is a theory for spatial structure analysis that was established by introducing fundamental operators applied to two sets. A set is processed by another one having a carefully selected shape and size, known as the structuring element (SE).[11] The two basic operations of mathematical morphology are erosion and dilation, which have been extended to hyperspectral image data by adopting a

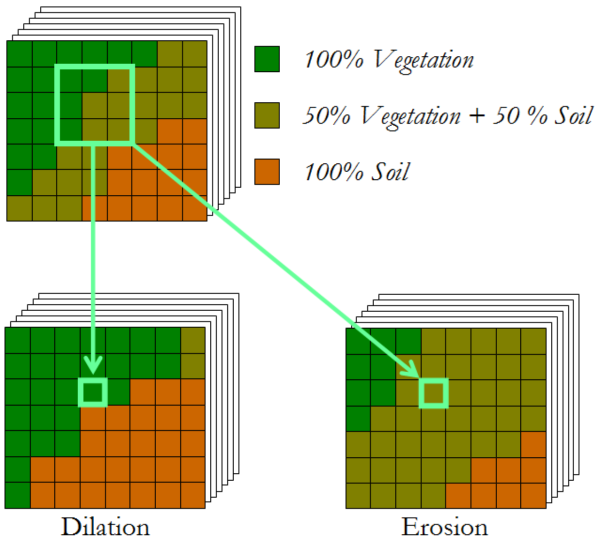

Figure 3. Toy example illustrating extended morphological operations.

distance-based technique which utilizes a cumulative distance between one particular pixel vector $\mathbf{X}(i,j)$, and all the pixel vectors in the spatial neighborhood given by a SE denoted by K as follows:

$$C_K(\mathbf{X}(i,j)) = \sum_{(i',j')\in K} \mathrm{SAD}(\mathbf{X}(i,j),\mathbf{X}(i',j')), \tag{6}$$

where SAD is the spectral angle distance.[3] As a result,$C_K(\mathbf{X}(i,j))$ is given by the sum of SAD scores between $\mathbf{X}(i,j)$ and every other pixel vector in the K-neighborhood. At this point, we need to be able to define a maximum and an minimum given an arbitrary set of vectors $\mathbf{S} = \{\mathbf{v}_1,\mathbf{v}_2,\cdots,\mathbf{v}_p\}$, where k is the number of vectors in the set. This can be done by computing $C_K(\mathbf{S}) = \{C_K(\mathbf{v}_1), C_K(\mathbf{v}_2),\cdots,C_K(\mathbf{v}_k)\}$ and selecting $\mathbf{v}_i$ such that $C_K(\mathbf{v}_i)$ is the minimum of $C_K(\mathbf{S})$, with $1 \leq i \leq k$. In similar fashion, we can select $\mathbf{v}_j$ such that $C_K(\mathbf{v}_j)$ is the maximum of $C_K(\mathbf{S})$, with $1 \leq j \leq p$. Based on the definitions above, the extended erosion $\boldsymbol{f} \ominus K$ consists of selecting the K-neighborhood pixel vector that produces the minimum C_K value as follows:[15]

$$(\mathbf{X} \ominus K)(i,j) = \mathrm{argmin}_{(i',j')\in K}\{C_K(\mathbf{X}(i+i',j+j')\}. \tag{7}$$

On the other hand, the extended dilation $\mathbf{X} \oplus K$ selects the K-neighborhood pixel that produces the maximum value for C_K as follows:[15]

$$(\mathbf{X} \oplus K)(i,j) = \mathrm{argmax}_{(i',j')\in K}\{C_K(\mathbf{X}(i-i',j-j')\}. \tag{8}$$

For illustrative purposes, Fig. 3 shows a graphical representation of the performance of these two basic operators using a toy example in which a synthetic hyperspectral image is used for demonstration. As can be seen in Fig. 3, morphological dilation expands the spatial regions made up of pure pixel vectors in accordance with the spatial neighborhood defined by a 3×3 SE, while morphological erosion expands the regions made up of highly mixed pixel vectors in accordance with the same spatial neighborhood. In order to avoid changing the size and shape of the features in the image, a desirable feature for spatial-spectral filtering, extended morphological opening and closing operations have also been defined, respectively, as follows: $(\mathbf{X} \circ K)(i,j) = [(\mathbf{X} \ominus K) \oplus K](i,j)$, i.e., erosion followed by dilation, and $(\mathbf{X} \bullet K)(i,j) = [(\mathbf{X} \oplus K) \ominus K](i,j)$, i.e., dilation followed by erosion.

With the above definitions in mind, the inputs to the AMEE algorithm are the full hyperspectral data cube $\boldsymbol{F}$, a structuring element K (used to define the spatial context around each image pixel), a maximum number of algorithm iterations I_{max}, and a maximum number of endmembers to be extracted, p. The output is an endmember set, $\{\boldsymbol{e}_i\}_{i=1}^{q}$, with $q \leq p$. A step-by-step description of the algorithm follows:

1. Set $i = 1$ and initialize a morphological eccentricity index $\text{MEI}(i, j) = 0$ for each pixel $\mathbf{X}(i, j)$.

2. Move K through all the pixels of the input data, defining a local spatial search area around each pixel $\mathbf{X}(i, j)$, and calculate the maximum and minimum pixel vectors at each K-neighborhood using extended opening and closing operations, respectively, and update the MEI at each spatial location (i, j) using:

$$\text{MEI}(i, j) = \text{SAD}[(\mathbf{X} \circ K)(i, j), (\mathbf{X} \bullet K)(i, j)]. \tag{9}$$

3. Set $i = i + 1$. If $i = I_{max}$, then go to step 4. Otherwise, replace the original image with its dilation using K as follows: $\mathbf{F} = \mathbf{F} \oplus K$. This propagates only the purest pixels at the local neighborhood to the following algorithm iteration. Then, go to step 2.

4. Select the set of p pixel vectors with higher associated MEI scores (called endmember candidates) and form a unique spectral set of $\{\mathbf{E}_z\}_{z=1}^{p}$ pixels, with $p \leq q$, by calculating the SAD for all pixel vector pairs.

2.3 Fully constrained linear spectral unmixing

Once a set of $\{\mathbf{E}_z\}_{z=1}^{p}$ endmembers has been extracted from the original scene $\mathbf{F}$, for each pixel vector $\mathbf{X}(i, j)$ in the scene a set of abundance fractions specified by $\Phi(i, j) = \{\Phi_1(i, j), \Phi_2(i, j), \cdots, \Phi_p(i, j)\}$ can be obtained[3] using the set of endmembers $\{\mathbf{E}_z\}_{z=1}^{p}$ by minimizing the noise term $\mathbf{n}(i, j)$ in the following expression: $\mathbf{X}(i, j) = \mathbf{E}_1 \cdot \Phi_1(i, j) + \mathbf{E}_2 \cdot \Phi_2(i, j) + \cdots + \mathbf{E}_p \cdot \Phi_p(i, j) + \mathbf{n}(i, j)$, thus solving the mixture problem described in 1. In order to achieve the decomposition above, we multiply each pixel $\mathbf{X}(i, j)$ by $(\mathbf{M}^T\mathbf{M})^{-1}\mathbf{M}^T$, where $\mathbf{M} = \{\mathbf{E}_z\}_{z=1}^{p}$ and the superscript "T" denotes the matrix transpose operation. In the expression above, the ANC and ASC constraints are imposed as described by Chang.[16]

3. LOSSY COMPRESSION FRAMEWORK

The idea of the proposed data compression algorithm is to represent a hyperspectral image cube $\mathbf{F}$ by a set of p fractional abundance images (one per extracted endmember), where $p \leq n$. This idea has been suggested in previous work.[17] More precisely, for each n-dimensional pixel vector $\mathbf{X}(i, j)$, its associated p-dimensional abundance vector $\Phi(i, j)$ is used as a fingerprint of $\mathbf{X}(i, j)$ with regards to p endmembers obtained by the PPI or AMEE algorithm. The implementation of the proposed data compression algorithm can be simply summarized by the following steps:

1. Use the PPI or AMEE algorithms to generate a set of p endmembers $\mathbf{E}_z\}_{z=1}^{p}$.

2. For each pixel vector $\mathbf{X}(i, j)$ in the input hyperspectral scene $\mathbf{F}$, use the FCLSU algorithm to estimate the endmember abundance fractions $\Phi(i, j) = \{\Phi_1(i, j), \Phi_2(i, j), \cdots, \Phi_p(i, j)\}$ and approximate $\mathbf{X}(i, j) = \mathbf{E}_1 \cdot \Phi_1(i, j) + \mathbf{E}_2 \cdot \Phi_2(i, j) + \cdots + \mathbf{E}_p \cdot \Phi_p(i, j) + \mathbf{n}(i, j)$. Note that this is a reconstruction of $\mathbf{X}(i, j)$, where the quality of the reconstruction is determined by the number of endmembers p.

3. Construct p fractional abundance images, one for each endmember, and represent the original n-dimensional hyperspectral image $\mathbf{F}$ as a collection of p fractional abundance planes, where the value of p can be increased or decreased to control the compression ratio, given by n/p.

4. GPU-BASED IMPLEMENTATION

GPUs can be abstracted in terms of a *stream* model, under which all data sets are represented as streams (i.e., ordered data sets). Algorithms are constructed by chaining so-called *kernels*, which operate on entire streams, taking one or more streams as inputs and producing one or more streams as outputs. Thereby, data-level parallelism is exposed to hardware, and kernels can be concurrently applied without any sort of synchronization. Modern GPU architectures adopt this model and implement a generalization of the traditional rendering pipeline, which consists of two main stages:

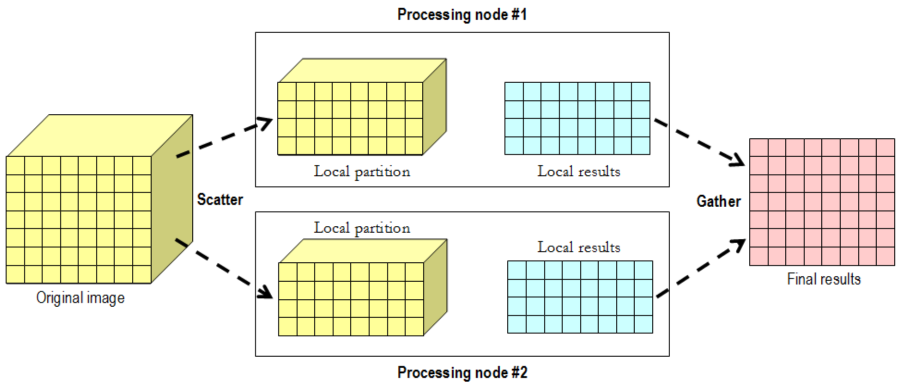

Figure 4. Spatial-domain decomposition of a hyperspectral data set.

1. *Vertex processing*. The input to this stage is a stream of vertices from a 3-D polygonal mesh. Vertex processors transform the 3-D coordinates of each vertex of the mesh into a 2-D screen position, and apply lighting to determine their colors (this stage is fully programmable).

2. *Fragment processing*. In this stage, the transformed vertices are first grouped into rendering primitives, such as triangles, and scan-converted into a stream of pixel fragments. These fragments are discrete portions of the triangle surface that corresponds to the pixels of the rendered image. Apart from identifying constituent fragments, this stage also interpolates attributes stored at the vertices, such as texture coordinates, and stores the interpolated values at each fragment. Arithmetical operations and texture lookups are then performed by fragment processors to determine the ultimate color for the fragment. For this purpose, texture memories can be indexed with different texture coordinates, and texture values can be retrieved from multiple textures.

It should be noted that fragment processors currently support instructions that operate on vectors of four RGBA components (Red/Green/Blue/Alpha channels) and include dedicated texture units that operate with a deeply pipelined texture cache. As a result, an essential requirement for mapping non-graphics algorithms onto GPUs is that the data structure can be arranged according to a stream-flow model, in which kernels are expressed as *fragment* programs and data streams are expressed as *textures*. Using C-like, high-level languages such as the compute unified device architecture (CUDA), programmers can write fragment programs to implement general-purpose operations. Our proposed lossy compression framework was implemented using NVidia^{TM} CUDA, a collection of C extensions and a runtime library. CUDAs functionality primarily allows a developer to write C functions to be executed on the GPU. CUDA also includes memory management and execution configuration, so that a developer can control the number of GPU processors and threads that are to be invoked during a function's execution.

The first issue that needs to be addressed is how to map a hyperspectral image onto the memory of the GPU. Since the size of hyperspectral images usually exceeds the capacity of such memory, we split them into multiple spatial-domain partitions[18] made up of entire pixel vectors (see Fig. 4), i.e., each spatial-domain partition incorporates all the spectral information on a localized spatial region and is composed of spatially adjacent pixel vectors. Each spatial-domain partition is further divided into 4-band tiles (called spatial-domain tiles), which are arranged in different areas of a 2-D texture. Such partitioning allows us to map four consecutive spectral bands onto the RGBA color channels of a texture element.

Apart from the tiles, we also allocate additional memory to hold other information, such as the skewers which are generated at the host (CPU) and then transmitted to the GPU, and also intermediate results such as dot products, norms, and point-wise distances. Figure 5 shows a flowchart describing the kernels involved in our GPU-based implementations of PPI, AMEE and FCLSU algorithms. There are two kernels which are common to the two considered endmember extraction algorithms: the *data partitioning* stage performs the spatial-domain decomposition of the original hyperspectral image. In the *stream uploading* stage, the spatial-domain partitions

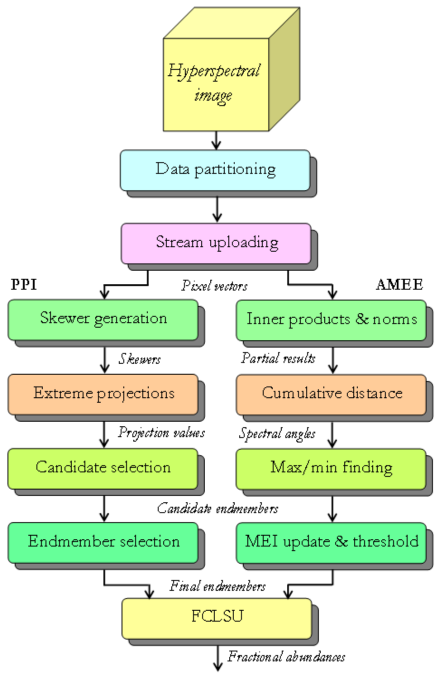

Figure 5. Flowchart of the stream-based GPU implementations.

are uploaded as a set of tiles onto the GPU memory. The kernels which are specific to the PPI implementation can be described as follows:

- *Skewer generation.* This kernel provides the skewers, using NVidia^{TM} parallel implementation of the Mersenne twister pseudo-random number generator on the GPU.[19]
- *Extreme projections.* The tiles are input streams to this stage, which obtains all the dot products necessary to compute the required projections. The implementation of this stage is based on a multi-pass kernel that implements an element-wise multiply and add operation, thus producing four partial inner products stored in the RGBA channels of a texture element.
- *Candidate selection.* This kernel uses as inputs the projection values generated in the previous stage, and produces a stream for each pixel vector, containing the relative coordinates of the pixels with maximum and minimum distance after the projection onto each skewer. A complementary kernel is then used to find pixels which have been selected repeatedly during the process.
- *Endmember selection.* For each endmember candidate, this kernel computes the cumulative SAD with all the other candidates. It is based on a single-pass kernel that computes the SAD between two pixel vectors using the dot products and norms produced by the previous stage. A complementary kernel is then used to discard those candidates with cumulative SAD scores below a threshold angle.

On the other hand, the GPU implementation of AMEE algorithm requires the following specific kernels:

- *Inner products & norms.* The tiles are input streams to this kernel, which obtains all the inner products and norms necessary to compute the point-wise spectral angle distances involved in the calculations.
- *Cumulative distance.* For each pixel vector, this kernel accumulates the spectral angle with all the neighboring pixels. It is based on a single-pass kernel that computes the spectral angle distance between two pixel vectors using the inner products and norms produced by the previous kernel. Finally, the kernel calculates, for each pixel vector, the cumulative spectral angle between the pixel and all its neighbors.

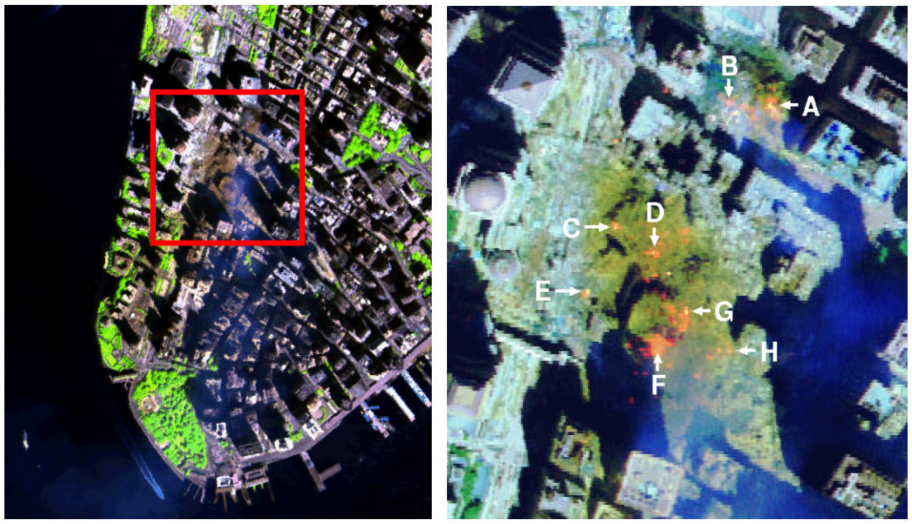

Figure 6. False color composition of an AVIRIS hyperspectral image collected by NASA's Jet Propulsion Laboratory over lower Manhattan on Sept. 16, 2001 (left). Location of thermal hot spots in the fires observed in World Trade Center area, available online: http://pubs.usgs.gov/of/2001/ofr-01-0429/hotspot.key.tgif.gif (right).

- *Max/min finding.* Morphological erosion and dilation are finalized at this stage through a kernel that applies minimum and maximum reductions. This kernel uses as inputs the cumulative values generated in the previous stage and produces a stream containing (for each pixel) the relative coordinates of the neighboring pixels with maximum and minimum cumulative distance.
- *Eccentricity update.* This kernel updates the morphological eccentricity scores using the maximum/minimum and point-wise distance streams. A complementary kernel applies a threshold to select a set of final endmembers at the end of the process.

Finally, the FCLSU kernel (common to both PPI and AMEE endmember extraction algorithms) uses as inputs the final endmembers selected in the previous stage and produces the endmember fractional abundances for each pixel by means of an inversion process.

5. EXPERIMENTAL RESULTS

This section evaluates the effectiveness of the proposed lossy hyperspectral compression framework. First, we describe the hyperspectral data sets used for evaluation purposes. A detailed survey on algorithm performance in a real application domain is then provided, along with a discussion on the advantages and disadvantages of the proposed lossy compression technique in terms of hyperspectral data fidelity after compression. The section concludes with an evaluation of parallel performance of the proposed GPU-based implementation.

5.1 Hyperspectral data

The image scene used for experiments in this work was collected by the AVIRIS instrument, which was flown by NASA's Jet Propulsion Laboratory over the World Trade Center (WTC) area in New York City on September 16, 2001, just five days after the terrorist attacks that collapsed the two main towers and other buildings in the WTC complex. The data set selected for experiments was geometrically and atmospherically corrected prior to data processing, and consists of 614×512 pixels, 224 spectral bands and a total size of 140 MB. The spatial resolution is 1.7 meters per pixel. The leftmost part of Fig. 6 shows false color composition of the data set selected for experiments, in which a detail of the WTC area is shown in a rectangle. The rightmost part of Fig. 6 shows a thermal map generated by U.S. Geological Survey and centered at the region where the buildings collapsed. The map displays the locations of the thermal hot spots with temperatures ranging from 700^{o}F (marked as "F") to 1300^{o}F (marked as "G"). This thermal map will be used in this work as ground-truth to substantiate the

Table 1. SAD-based spectral similarity scores between the original hyperspectral image pixels labeled as hot spots in the AVIRIS scene over the World Trade Center and the corresponding pixels in the hyperspectral images obtained after compression using different techniques, including JPEG2000, 3D-SPIHT and the proposed method (with PPI and AMEE).

Hot spot	**Lat (North)**	**Long (West)**	**Temp (K)**	**JPEG2000**			**3D-SPIHT**			**Proposed (PPI)**			**Proposed (AMEE)**		
				20:1	40:1	80:1	20:1	40:1	80:1	20:1	40:1	80:1	20:1	40:1	80:1
‘A’	40^o42‘47.18“	74^o00‘41.43“	1000	0.114	0.125	0.136	0.107	0.112	0.124	0.065	0.082	0.092	0.055	0.073	0.088
‘B’	40^o42‘47.14“	74^o00‘43.53“	830	0.124	0.131	0.140	0.109	0.113	0.129	0.058	0.077	0.094	0.054	0.069	0.083
‘C’	40^o42‘42.89“	74^o00‘48.88“	900	0.098	0.117	0.129	0.106	0.111	0.126	0.068	0.079	0.093	0.061	0.072	0.087
‘D’	40^o42‘41.99“	74^o00‘46.94“	790	0.135	0.144	0.151	0.115	0.123	0.138	0.073	0.082	0.099	0.065	0.073	0.084
‘E’	40^o42‘40.58“	74^o00‘50.15“	710	0.123	0.139	0.150	0.110	0.119	0.141	0.071	0.080	0.090	0.070	0.069	0.085
‘F’	40^o42‘38.74“	74^o00‘46.70“	700	0.116	0.128	0.146	0.099	0.110	0.123	0.055	0.069	0.083	0.057	0.062	0.079
‘G’	40^o42‘39.94“	74^o00‘45.37“	1020	0.119	0.133	0.149	0.103	0.120	0.128	0.062	0.071	0.088	0.060	0.067	0.082
‘H’	40^o42‘38.60“	74^o00‘43.51“	820	0.130	0.144	0.152	0.119	0.131	0.145	0.069	0.078	0.091	0.062	0.074	0.089

capacity of the proposed compression algorithm to retain the sub-pixel spectral information required to detect the hot spot fires in the WTC complex.

5.2 Experimental assessment of data fidelity after compression

Prior to a full examination and discussion of compression results, it is important to outline parameter values used for the different stages of the proposed hyperspectral lossy compression technique. Specifically, the number of endmembers to be extracted in the WTC data was set to $p = 20$ after estimating the intrinsic dimensionality of the data using the *virtual dimensionality* (VD) concept.[3] For the PPI algorithm, the number of skewers was set to $T = 10^4$ (although values of $T = 10^3$ and $T = 10^5$ were also tested, we experimentally observed that the use of $T = 10^3$ resulted in the loss of important endmembers, while the endmembers obtained using $T = 10^5$ were essentially the same as those found using $T = 10^4$). The threshold angle parameter was set to $t_a = 0.1$, which is a reasonable limit of tolerance for this metric, while the threshold value parameter t_v was set to the mean of PPI scores obtained after $T = 10^4$ iterations. Finally, the size of the kernel K used for the AMEE algorithm was empirically set to 5×5 pixels after testing different window sizes. These parameter values are in agreement with those used before in the literature.[7]

In order to explore the degree of spectral fidelity of the compressed images obtained after applying the proposed compression method, Table 1 reports the SAD-based spectral similarity scores among the pixels labeled as hot spots in the original image (see rightmost part of Fig. 6) and the pixels at the same spatial locations in the images obtained after applying the proposed lossy hyperspectral data compression technique using both PPI and AMEE (in the table, the lowest the scores, the highest the spectral similarity). In experiments, compression ratios of 20:1, 40:1 and 80:1 (given by different tested values of input parameter p) were used. For illustrative purposes, we have also included the results provided by two standard methods in our comparison, i.e., the wavelet-based JPEG2000 multicomponent[20] and the 3D-SPIHT.[21] The JPEG2000 implementation used for our experiments was the one available in kakadu software*. Both techniques are 3D compression algorithms that treat the hyperspectral data as a 3D volume, where the spectral information is the third dimension. As expected, as the compression ratio was increased, the quality of extracted endmembers was decreased. Results in Table 1 show that, for the same compression ratio, a 3D lossy compression algorithm may result in significant loss of spectral information which can be preserved better, in turn, by an application-oriented algorithm such as the proposed lossy compression approach (using either PPI or AMEE for the endmember extraction stage, with AMEE providing slightly better results likely due to the incorporation of spatial information).

5.3 Experimental assessment of parallel performance

The GPU-based experiments were performed on a 2006-model HP xw8400 workstation based on dual Quad-Core Intel Xeon processor E5345 running at 2.33 GHz with 1.333 MHz bus speed and 3 GB RAM. The computer was equipped with an NVidia$^{\text{TM}}$ GeForce 8800 GTX with 16 multiprocessors, each composed of 8 SIMD processors operating at 1350 Mhz. Each multiprocessor has 8192 registers, a 16 KB parallel data cache of fast shared memory, and access to 768 MB of global memory. The GPU architecture is graphically illustrated in Fig. 7.

Table 2 shows the execution times measured for different image sizes by the CPU and GPU-based implementations, respectively, where the largest image size in the table (140 MB) corresponds to the full hyperspectral scene (614×512 pixels and 224 spectral bands) whereas the others correspond to cropped portions of the same

*http://www.kakadusoftware.com

Table 2. Processing time (seconds) for the dual-core CPU and GPU implementations.

Size	**Proposed framework (using PPI)**		**Proposed framework (using AMEE)**	
(MB)	Processing time (CPU)	Processing time (GPU)	Processing time (CPU)	Processing time (GPU)
18	66.58	2.55	172.94	6.72
36	126.13	4.87	341.06	13.88
70	249.35	9.57	675.21	26.12
140	493.78	18.93	1345.18	52.60

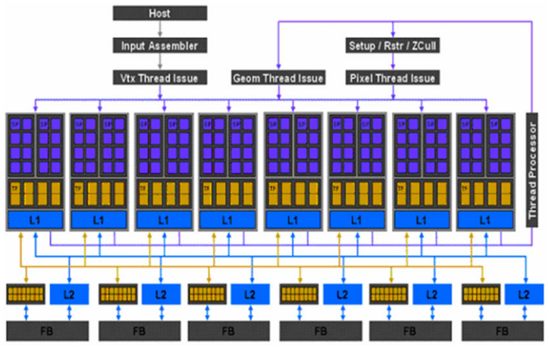

Figure 7. NVidia^{TM} GeForce 8800 GTX architecture.

image. In all cases, the number of endmembers was fixed to $p = 20$ (VD original estimate). The C function *clock()* was used for timing the CPU implementation and the CUDA timer was used for the GPU implementation. The time measurement was started right after the hyperspectral image file was read to the CPU memory and stopped right after the results of the processing chain were obtained and stored in the CPU memory. From Table 2, it can be seen that the full AVIRIS data cube was compressed in 18.93 seconds when PPI was used to extract the endmembers, and on 52.60 seconds when the AMEE was used. These response times are not strictly in real-time since the cross-track line scan time in AVIRIS, a push-broom instrument,[2] is quite fast (8.3 msec). This introduces the need to process the full image cube (614 lines) in about 5 seconds to achieve fully achieve real-time performance. Although the proposed implementation can still be optimized, Table 2 indicates that the complexity of the implementation scales linearly with the problem size, i.e. doubling the image size doubles the execution time. The speedups achieved by the GPU implementations over the corresponding CPU ones remained close to 26 for all considered image sizes. This is achieved using only one GPU device, with very few on-board restrictions in terms of cost, power consumption and size, which are important when defining mission payload.

6. CONCLUSIONS

The emergence of low-weight and low-power specialized hardware devices such as graphics processing units (GPUs), whose evolution in terms of performance and cost is mainly due to the advent of video-game industry, is now progressively bridging the gap towards real-time analysis of high dimensional data in many application domains, including remote sensing. This kind of specialized, on-board processing devices are essential to reduce mission payload and obtain analysis results quickly enough for practical use. In this work, we have presented our experience in computationally efficient implementation of lossy compression algorithms for remotely sensed hyperspectral data using commodity graphics hardware. Our implementations (tested on the NVidia^{TM} GeForce 8800 GTX GPU) resulted in speedups on the order of 26, which allows fast and efficient compression of hyperspectral data. Our future work will aim at achieving quality compression in real-time, on-board the sensor.

7. ACKNOWLEDGEMENT

This work has been supported by the European Community's Marie Curie Research Training Networks Programme under reference MRTN-CT-2006-035927 (HYPER-I-NET). Funding from the Spanish Ministry of Science and Innovation (HYPERCOMP/EODIX project, reference AYA2008-05965-C04-02) is gratefully acknowledged.

REFERENCES

1. A. F. H. Goetz, G. Vane, J. E. Solomon, and B. N. Rock, "Imaging spectrometry for earth remote sensing," *Science* **228**, pp. 1147–1153, 1985.
2. R. O. Green, "Imaging spectroscopy and the airborne visible-infrared imaging spectrometer (AVIRIS)," *Remote Sensing of Environment* **65**, pp. 227–248, 1998.
3. C.-I. Chang, *Hyperspectral imaging: techniques for spectral detection and classification*, Kluwer Academic and Plenum Publishers, New York, 2003.
4. A. Plaza and C.-I. Chang, *High performance computing in remote sensing*, CRC Press, Boca Raton, 2006.
5. G. Motta, F. Rizzo, and J. A. Storer, *Hyperspectral data compression*, Springer-Verlag, New York, 2005.
6. J. B. Adams, M. O. Smith, and P. E. Johnson, "Spectral mixture modeling: a new analysis of rock and soil types at the viking lander 1 site," *Journal of Geophysical Research* **91**, pp. 8098–8112, 1986.
7. A. Plaza, P. Martinez, R. Perez, and J. Plaza, "A quantitative and comparative analysis of endmember extraction algorithms from hyperspectral data," *IEEE Transactions on Geoscience and Remote Sensing* **42**, pp. 650–663, 2004.
8. J. W. Boardman, "Automating spectral unmixing of aviris data using convex geometry concepts," in *Summaries of Airborne Earth Science Workshop*, R. O. Green, ed., *JPL Publication* **93-26**, pp. 111–114, 1993.
9. M. Zortea and A. Plaza, "Spatial preprocessing for endmember extraction," *IEEE Transactions on Geoscience and Remote Sensing* **47**, 2009.
10. A. Plaza, P. Martinez, R. Perez, and J. Plaza, "Spatial/spectral endmember extraction by multidimensional morphological operations," *IEEE Transactions on Geoscience and Remote Sensing* **40**, pp. 2025–2041, 2002.
11. P. Soille, *Morphological image analysis: principles and applications*, Springer-Verlag, Berlin, 2003.
12. Y. Tarabalka, T. V. Haavardsholm, I. Kasen, and T. Skauli, "Real-time anomaly detection in hyperspectral images using multivariate normal mixture models and gpu processing," *Journal of Real-Time Image Processing* **4**, pp. 1–14, 2009.
13. J. Setoain, M. Prieto, C. Tenllado, A. Plaza, and F. Tirado, "Parallel morphological endmember extraction using commodity graphics hardware," *IEEE Geoscience and Remote Sensing Letters* **43**, pp. 441–445, 2007.
14. J. Setoain, M. Prieto, C. Tenllado, and F. Tirado, "GPU for parallel on-board hyperspectral image processing," *International Journal of High Performance Computing Applications* **22**(4), pp. 424–437, 2008.
15. A. Plaza, P. Martinez, J. Plaza, and R. Perez, "Dimensionality reduction and classification of hyperspectral image data using sequences of extended morphological transformations," *IEEE Transactions on Geoscience and Remote Sensing* **43**, pp. 466–479, 2005.
16. D. Heinz and C.-I. Chang, "Fully constrained least squares linear mixture analysis for material quantification in hyperspectral imagery," *IEEE Transactions on Geoscience and Remote Sensing* **39**, pp. 529–545, 2001.
17. Q. Du and C.-I. Chang, "Linear mixture analysis-based compression for hyperspectral image analysis," *IEEE Transactions on Geoscience and Remote Sensing* **42**, pp. 875–891, 2004.
18. A. Plaza, D. Valencia, J. Plaza, and P. Martinez, "Commodity cluster-based parallel processing of hyperspectral Imagery," *Journal of Parallel and Distributed Computing* **66**(3), pp. 345–358, 2006.
19. M. Matsumoto and T. Nishimura, "Mersenne twister: 623-dimensionally equidistributed uniform random number generator," *ACM Transactions on Modeling and Computer Simulation* **8**(1), pp. 3–30, 1998.
20. D. S. Taubman and M. W. Marcellin, *JPEG2000: Image compression fundamentals, standard and practice*, Kluwer, Boston, 2002.
21. B.-J. Kim, Z. Xiong, and W. A. Pearlman, "Low bit-rate scalable video coding with 3-D set partitioning in hierarchical trees (3-D SPIHT)," *IEEE Transactions on Circuits and Systems for Video Technology* **10**, pp. 1374–1387, 2000.