J Supercomput
DOI 10.1007/s11227-014-1104-2

Unmixing-based content retrieval system for remotely sensed hyperspectral imagery on GPUs

Jorge Sevilla · Sergio Bernabe · Antonio Plaza

Abstract This paper presents a new unmixing-based retrieval system for remotely sensed hyperspectral imagery. The need for this kind of system is justified by the exponential growth in the volume and number of remotely sensed data sets from the surface of the Earth. This is particularly the case for hyperspectral images, which comprise hundreds of spectral bands at different (almost contiguous) wavelength channels. To deal with the high computational cost of extracting the spectral information needed to catalog new hyperspectral images in our system, we resort to efficient implementations of spectral unmixing algorithms on commodity graphics processing units (GPUs). Spectral unmixing is a very popular approach for interpreting hyperspectral data with sub-pixel precision. This paper particularly focuses on the design of the proposed framework as a web service, as well as on the efficient implementation of the system on GPUs. In addition, we present a comparison of spectral unmixing algorithms available in the system on both CPU and GPU architectures.

Keywords Hyperspectral imaging · Content-based image retrieval (CBIR) · Spectral unmixing · Endmember extraction · GPUs · Distributed resources

The system is available online at: http://www.hypercomp.es/repository.

J. Sevilla · S. Bernabe · A. Plaza (✉)
Hyperspectral Computing Laboratory, Department of Technology of Computers and Communications, Escuela Politecnica de Caceres, University of Extremadura, Cáceres, Spain
e-mail: aplaza@unex.es

J. Sevilla
e-mail: jorgesece@unex.es

S. Bernabe
e-mail: sergiobernabe@unex.es

Published online: 01 February 2014

1 Introduction

Content-based image retrieval (CBIR) intends to retrieve, from real data stored in a database, information that is relevant to a query [1]. This is particularly important in large data repositories, such as those available in remotely sensed hyperspectral imaging [2]. For instance, the NASA Jet Propulsion Laboratory's Airborne Visible Infra-Red Imaging Spectrometer (AVIRIS) [3] is able to record the visible and near infrared spectrum of the reflected light of an area several kilometers long (depending on the duration of the flight) using hundreds of spectral bands. The resulting 'image cube' is a stack of images, in which each pixel (vector) has an associated spectral signature (or 'fingerprint') that uniquely characterizes the underlying objects. The resulting data volume often comprises several Gigabytes per flight. As a result, the computational requirements needed to store, manage and process these images are enormous [4]. For this reason, the development of CBIR systems for hyperspectral data repositories is an emerging need in remote sensing applications, particularly after the development of several new missions [5].

In this paper, we take a necessary first step towards the development of an open-source hyperspectral repository for sharing and efficiently retrieving image data in the hyperspectral imaging community. The newly developed data repository allows uploading new hyperspectral data sets along with (automatically generated) meta-data, ground-truth and analysis results. The developed system includes a spectral unmixing-based CBIR functionality, which allows retrieving images from the database using the spectrally pure components associated with each scene, and which can be used as effective meta-data for CBIR purposes. This is because spectral unmixing [6] has become a standard technique for analyzing hyperspectral images with sub-pixel precision. It should also be noted that, due to limited spatial resolution, most hyperspectral images are dominated by mixed pixels, i.e., pixels which are composed of several underlying pure spectral responses (called *endmembers* in the spectral unmixing literature). Our main focus in this work is on describing the design of the system as a web service, as well as on the computational algorithms which are available in our digital repository. We also provide a general description of the system including spectral unmixing-based CBIR functionality for searching images using the information contained in mixed pixels, which dominate hyperspectral images. In addition, we present a comparison of unmixing algorithms available in the system on both CPU and GPU architectures.

The remainder of the paper is organized as follows. Section 2 describes the unmixing algorithms that are used for the CBIR part of the system and their efficient implementation on GPUs. Section 3 generally describes the proposed CBIR system, with particular emphasis on its implementation as a web service. Section 4 presents experimental results to validate the accuracy and computational efficiency of the newly developed implementations. Section 5 concludes with some remarks.

2 Spectral unmixing algorithms and their GPU implementation

One of the main problems involved in hyperspectral data exploitation is spectral unmixing [7], as many of the pixels collected by imaging spectrometers such as AVIRIS are

highly mixed in nature due to spatial resolution and other phenomena. Linear spectral unmixing follows an unsupervised approach which aims at inferring pure spectral signatures, called *endmembers*, and their material fractions at each pixel of the scene. To deal with the computational cost of extracting the information needed to catalog a new hyperspectral image in our system, we resort to efficient implementations of spectral unmixing algorithms on GPUs, which have been successfully used to accelerate hyperspectral-related computations [8,9]. The spectral unmixing chain considered in this work comprises two main steps: (1) estimation of number of pure spectral signatures (endmembers), and (2) extracting a collection of endmembers from each considered hyperspectral scene. In the following, we describe the GPU implementation of several spectral unmixing algorithms used by our system to catalog new scenes and generate their associated meta-data.

The GPU implementations have been carried out using the compute unified device architecture (CUDA) developed by *NVidia*$^{\text{TM}}$. The architecture of a GPU can be abstracted as a set of multiprocessors (MPs), in which each multiprocessor is characterized by a single instruction multiple data (SIMD) architecture, i.e., in each clock cycle, each processor executes the same instruction but operating on multiple data streams. Each processor accesses a local shared memory and also local cache memories in the multi-processor, while the multiprocessors have access to the global GPU (device) memory. GPUs can be, therefore, 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 and which are executed by a multiprocessor, 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. The kernels can perform a kind of batch processing arranged in the form of a grid of blocks, where each block is composed of a group of threads that share data efficiently through the shared local memory and synchronize their execution for coordinating accesses to memory.

2.1 Estimation of the number of endmembers on GPUs

2.1.1 GPU implementation of virtual dimensionality (VD)

Virtual dimensionality [10] algorithm defines the number of endmembers as signal sources which are determined based on their distinct spectral properties. Once we load the full hyperspectral image $\mathbf{Y}$ (in pixel-by-pixel fashion) from disk to the main memory of the GPU, the first step is to calculate the covariance matrix $\mathbf{K}_{L\times L}$, where L denotes the number of spectral bands in the original hyperspectral image. For this purpose, we need to calculate the mean value $\overline{\mathbf{Y}}$ of each band of the image and subtract this mean value to all the pixels in the same band. To perform this calculation in the GPU, we use a kernel called `MeanPixel` as indicated in [11]. The resulting mean values of each band $\overline{\mathbf{Y}}$ are stored in a structure as they will be needed for the calculation of the covariance matrix $\mathbf{K}_{L\times L}$ in the GPU by means of a matrix multiplication operation $((\mathbf{Y}-\overline{\mathbf{Y}})^{\mathrm{T}}(\mathbf{Y}-\overline{\mathbf{Y}})$. This operation is performed using the

cuBLAS library. Specifically, we use the cublasSgemm function of cuBLAS. The next step is to calculate the correlation matrix $\mathbf{K}_{L\times L}$ in the GPU. To achieve this, we use a kernel called Correlation [11]. The remaining steps in the VD calculation (i.e., extraction of correlation-eigenvalues, covariance-eigenvalues and Neyman–Pearson test for estimation of the number of endmembers [10]) can be computed very fast in the CPU.

2.1.2 GPU implementation of hyperspectral subspace identification by minimum error (HySime)

The HySime [12] algorithm is split into two steps. First, the algorithm removes the noise in the original hyperspectral image, while in the second step the algorithm estimates the subspace in which the hyperspectral data live. Once we load the hyperspectral image $\mathbf{Y}$ in GPU memory, we first implement the noise estimation algorithm. For this purpose, the first step is to compute a signal correlation matrix $\mathbf{R}^{s}_{L\times L}$ and its inverse [11]. The former is calculated in the GPU by means of standard cuBLAS matrix multiplication (using cublasSgemm), while the latter is implemented in the CPU. This is because we have experimentally checked that, in our particular application context, the required memory transfers for a parallel execution of the inverse on GPUs consume more time than a serial execution on the CPU. Then, we estimate a noise correlation matrix $\mathbf{R}^{n}_{L\times L}$, which could be performed using a standard matrix multiplication in cuBLAS [11]. However, in the next steps of the algorithm, we only consider the diagonal elements of this matrix, hence it is not necessary to compute the rest of the values. As a result, we use L threads to ensure coalesced accesses to perform the computation of an element of the diagonal of $\mathbf{R}^{n}_{L\times L}$ [11]. These two matrices are used, via a minimization function, to get the number of different endmember materials $\hat{p}$ in the CPU.

2.2 Endmember extraction on GPUs

2.2.1 GPU implementation of N-FINDR

N-FINDR [13] algorithm looks for the set of pixels with the largest possible volume by *inflating* a simplex inside the data (see Fig. 1, which shows a toy example with three dimensions). The idea of the algorithm is to go from an initial random estimation which may not enclose all the pixels in the scene (see Fig. 1a) to an optimal solution in which all pixels in the scene are enclosed by the estimated endmembers (see Fig. 1b). Prior to the implementation on GPU, a set of optimizations were performed (see [14] for additional details). In the following, we describe in step-by-step fashion how the algorithm has been implemented in GPU.

1. *Initialization*. First, we obtain a reduced version $\mathbf{M}$ of the original hyperspectral image with p components, obtained resulting from the principal component transform (PCT) transform which is also performed on GPU [9]. Then, we form a matrix of size $p \times p$ by initializing to ones the first row and setting in each column (from row two) a randomly selected pixel from $\mathbf{M}$. The determinant of

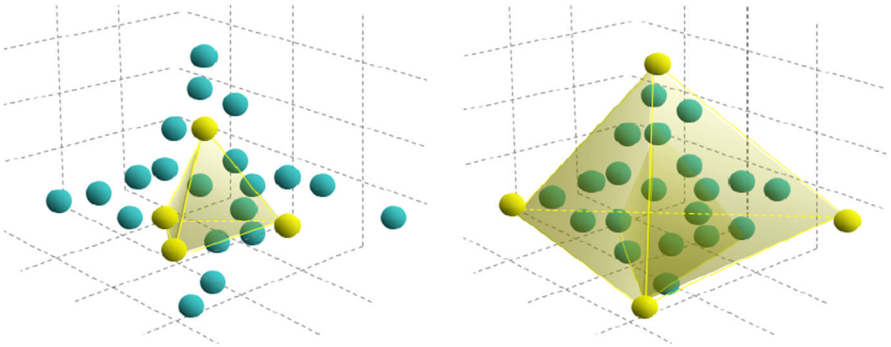

Fig. 1 Graphical interpretation of the N-FINDR algorithm in a three-dimensional space

the resulting matrix gives an initial volume $\bar{\mathbf{V}}_M^{(1)}$ that is now stored in the variable `currentVolume`. Since the dimensions of the aforementioned matrix are small, the determinant computation can be done in CPU.

2. *Volume calculation and replacement*. Next, we form a vector `Vvolumes` with as many components as the number of endmembers, p. The volume resulting from the replacement of each endmember with each pixel of the scene is now calculated using the GPU kernel `VolumeCalculations`, following the procedure described in [14]. For each endmember position, we need to find the pixel which generated the biggest volume, and check if this volume is bigger than `currentVolume`. For this task, we use the kernel `ReductionVol` which performs a reduction process in which each block works with a section of volume data and extracts the local maximum and the position of this local maximum. Since each block achieves a different value, at the end of the execution there will be as many values as blocks (each value is the local maximum of its section), and it will be necessary to store these values in a structure, together with their positions in global memory. Then, these values will be copied to the main memory of the CPU to reduce again and get the global maximum and its position.

2.3 GPU implementation of OSP-GS

We have carried out an efficient GPU implementation of the orthogonal subspace projection (OSP) [15] algorithm using the Gram–Schmidt orthogonalization (see [16]). Our implementation for GPUs can be summarized by the following steps [8]:

1. To optimize accesses, we first store the pixel vectors of the hyperspectral image $\mathbf{Y}$ by columns. Then, a structure is created in which the number of blocks equals the number of pixel vectors in the hyperspectral image divided by the number of threads per block. A kernel called `GetPixelMaxBright` is now used to calculate the brightest pixel $\mathbf{e}_1$ in $\mathbf{Y}$. This kernel computes (in parallel) the dot product between each pixel vector and its own transposed version, retaining the pixel that results in the maximum projection value.

2. Once the brightest pixel in $\mathbf{Y}$ has been identified, the pixel is allocated as the first column in matrix $\mathbf{M}$. The algorithm now calculates the orthogonal vectors through the Gram–Schmidt method [16]. This operation is performed in the CPU because this method operates on a small data structure and the results can be obtained very quickly. A new kernel is now created, in which the number of blocks equals the number of pixel vectors in the hyperspectral image divided by the number of supported threads. This kernel, called `PixelProjection`, is now used to project the orthogonal vector onto each pixel in the image. We use the shared memories in the GPU to store the most orthogonal vectors obtained at each iteration of OSP-GS (this is because these vectors will be accessed every time that the projection onto each pixel of the image is performed). The maximum of all projected pixels is calculated using a separate reduction kernel `ReductionProjection` which also uses the shared memory to store each of the projections and obtains the new endmember $\mathbf{e}_2$.
3. The algorithm now extends the endmember matrix as $\mathbf{M} = [\mathbf{e}_1, \mathbf{e}_2]$ and repeats from step 2) until the desired number of endmembers (specified by the input parameter p) has been extracted. The output of the algorithm is a set of endmembers $\mathbf{M} = [\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_p]$.

3 System design

The CBIR system that we have developed in this work is designed as a web service, where the users can access to the services through a user interface in transparent fashion, and the server is in charge of managing meta-data and executing unmixing algorithms. The architecture of the system (see Fig. 2) follows a modular design in which the communication between layers is defined using standard exchange data formats and transfer protocols. The system has been developed using free software tools such as `Symfony2`[1], a full-stack web framework written in `PHP`, `JavaScript` and `AJAX`. The communication between layers is carried out via Hypertext Transfer Protocol (`HTTP`)[2], Secure Shell (`SSH`)[3] and File Transport Protocol (`FTP`)[4], while the adopted format for data exchange is `JSON`[5]. In the following, we briefly describe the different layers that compose the system:

– The *client layer* provides easy access to external users through a web interface.
– The *server layer* manages the system and is composed of three parts. First, the *web server* communicates with the web interface and manages the system resources, such as meta-data and file data management, in addition to handling algorithm executions. The *database sever* is in charge of storing the image data. The *file storage sever* is in charge of uploading and downloading files (such as those associated with the image data and meta-data).

[1] http://symfony.com.
[2] http://www.w3.org/Protocols/rfc2616/rfc2616.html.
[3] http://www.ietf.org/rfc/rfc4251.txt.
[4] http://www.w3.org/Protocols/rfc959/.
[5] http://www.json.org/.

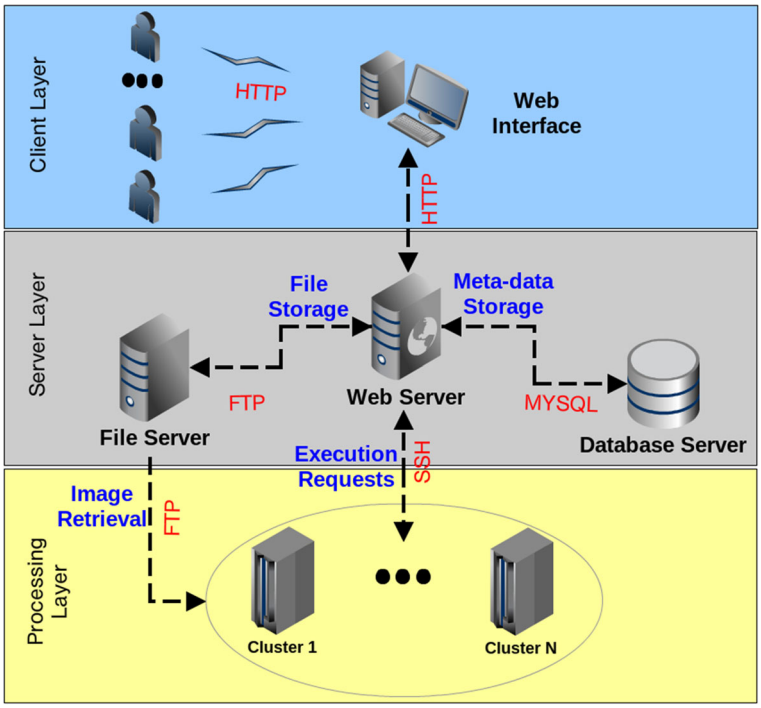

Fig. 2 Architecture of the proposed CBIR system

– The algorithms are executed by the *processing layer*, which relieves the web server load and provides high system availability.

The design of the system as a web service is achieved as follows. The client layer defines the interactions between the user (through an Internet browser) and our system, and is responsible for providing users with remote access to the system. The web interface has been designed using `jQuery`[6], `HTML5`[7] and `CSS3`[8]. The interaction between the user and the web interface is captured by the events handlers of the `jQuery` libraries. The web interface transmits the request to the server layer via `HTTP`, which can be considered the engine of the system since it is in charge of managing and connecting the different components of our system. Here, the database server stores image meta-data using `Mysql`[9]. A file storage server is also included in this layer for providing remote file access to any of the layers of the system. This module is in charge of uploading and downloading file data, such as images and meta-data, via `FTP`. The file server provides image data to the processing layer, which is in charge of executing algorithms with high computational cost, mostly related with the

[6] http://jquery.com.

[7] http://www.w3.org/TR/html51.

[8] http://www.w3.org/Style/CSS.

[9] http://www.mysql.com.

cataloguing of new hyperspectral images in the system and the execution of queries for image retrieval. In other words, the processing layer is in charge of efficiently executing requests coming from the web server. The execution requests from the web server are managed via `SSH`. As mentioned above, our processing layer supports efficient algorithm execution of unmixing algorithms using GPU architectures.

4 Experimental results

4.1 Hyperspectral data

In this section, we focus on evaluating the performance of the GPU algorithms which manage the cataloguing of new data in the system, as the content-based search based on meta-data is several orders of magnitude faster [17]. For this purpose, we have used two hyperspectral data sets to illustrate the capability of the system to efficiently catalog hyperspectral data for CBIR purposes. The data sets were collected by the Airborne Visible Infra-Red Imaging Spectrometer (AVIRIS):

- The first data set is the well-known AVIRIS Cuprite scene (see Fig. 3a), acquired in the summer of 1995. The portion used in experiments corresponds to a 350 × 350-pixels subset with 188 spectral bands in the range from 400 to 2,500 nm, and a total size of around 50 MB. The site is well understood mineralogically, and has several exposed minerals of interest, including *Alunite GDS83 Na63*, *Buddingtonite GDS85 D-206*, *Calcite WS272*, *Kaolinite CM9*, and *Muscovite GDS108*. Reference ground signatures of the above minerals, displayed in Fig. 3b and available in the USGS library[10], are used in this work for matching.
- The second data set was collected over World Trade Center (WTC) (see Fig. 4) area in New York City on September 16, 2001. The data set consists of 614 × 512 pixels, 224 spectral bands in the range from 400 to 2,500 nm and a total size of around 140 MB.

4.2 Analysis of matching accuracy

Our unmixing-based retrieval system retrieves the images based on the matching results obtained from reference endmembers extracted from those images. To illustrate the performance of the retrieval system, we specifically address a case study analyzing the accuracy of matching results with AVIRIS Cuprite scene using two extraction algorithms: N-FINDR and OSP-GS.

The spectral angle distance (SAD) [18] is used for matching the extracted endmember from the images with the reference endmembers, in which the range of values is [0°, 90°] where 0° is the best case. Table 1 presents the matching results for the Cuprite scene cataloged by the N-FINDR and OSP-GS algorithms. First, N-FINDR results present very low SAD scores, which indicate that the system provides high accuracy

[10] http://speclab.cr.usgs.gov.

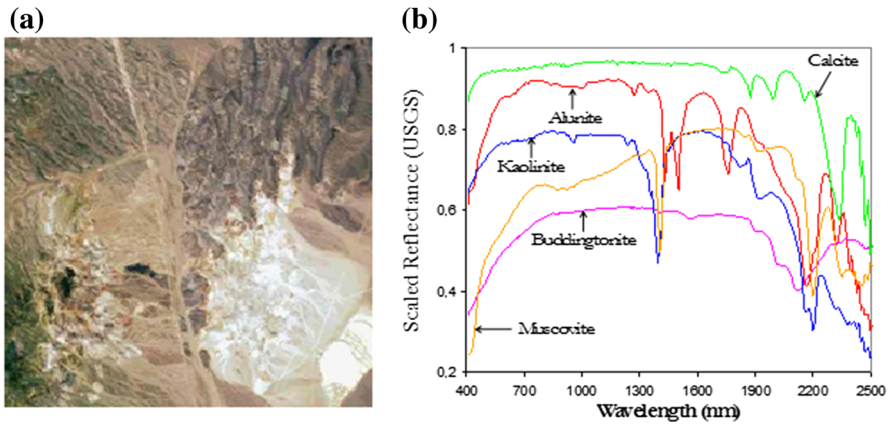

Fig. 3 **a** AVIRIS hyperspectral over cuprite mining district in Nevada and **b** US Geological Survey mineral spectral signatures used for validation purposes

Fig. 4 AVIRIS hyperspectral image collected by NASA Jet Propulsion Laboratory over WTC on September 16, 2001

in the matching process. On the other hand, the OSP-GS also provides results with very good accuracy, although the previous case provided slightly better results.

4.3 Performance analysis

The proposed unmixing chain has been tested on two different platforms (GPU and multicore-processor):

– The GPU platform is the *NVidia*™ TESLA C1060[11], which features 240 streaming processor cores with 1.3 GHz, total memory dedicated 4 GB and memory

[11] http://www.nvidia.es/object/tesla_c1060_es.html.

Table 1 Matching results (in degrees) obtained after comparing the AVIRIS Cuprite scene with the reference USGS mineral signatures available in for this scene

	Alunite	Buddingtonite	Calcite	Kaolinite	Muscovite	Average
N-FINDR	8.11°	4.26°	6.71°	8.73°	3.09°	6.18°
OSP-GS	9.05°	4.09°	5.13°	10.59°	3.31°	6.43°

Endmember extracted by N-FINDR and OSP-GS algorithms

Table 2 Processing times (in seconds) and speedups achieved for the GPU implementation of VD, HySime, OSP-GS and N-FINDR algorithms, tested with the AVIRIS WTC scene

	CPU time	GPU time	Speedup
VD			
Initialization	0.890	2.690	–
VD	61.668	0.617	99.950
Total	62.558	3.307	18.916
HySime			
Initialization	0.328	2.775	–
HySime	240.926	2.086	115.500
Total	241.254	4.861	49.631
OSP-GS			
Initialization	0.507	2.640	–
OSP	18.727	0.103	181.815
Total	19.234	2.743	7.012
N-FINDR			
Initialization	0.350	2.659	–
PCT	42.679	0.243	175.634
N-FINDR	18.054	0.736	24.530
Total	61.083	3.638	16.790

bandwidth of 102 GB/s. This GPU is connected to a Quad Core Intel Xeon at 2.26 GHz with 4 physical cores, of which only one is used, and 24 GB of DDR3 1,333 Mhz SRAM memory. It is mounted on a Bullx R422[12].

– The second platform is a multi-core system which is used also in our experiments. It is made up of a Intel Xeon CPU X7550 at 2.00 GHz with 8 cores, of which only one is used, and 1 TeraByte of DDR3 RAM. It is mounted on a Bullx s6030[13].

The serial algorithms (compiled using `gcc` with optimization flag –O3) were executed in one of the available cores of the multi-core system, and the GPU algorithms were executed in the used GPU architecture. Table 2 summarizes the timing results and speedups measured after processing the WTC data set by the C implementation and by the GPU implementation of VD, HySime, OSP-GS and N-FINDR algorithms. The results are broken down in initialization and processing time, and the speedups are calculated over the processing time and the total time (including initialization time).

[12] http://www.bull.com/catalogue/details.asp?tmp=bxs-rack-fr&opt=ns-r422e02&dt=ft&cat=bullx.

[13] http://www.bull.com/catalogue/details.asp?tmp=bxs-node&opt=bullx-s6030e00&cat=bullx&dt=ft.

All algorithms perform effectively on GPUs and provide significant speedups, even if the speedups are already quite high with the initialization time inclusion. From these results, we can conclude that GPUs provide a source of computational power that allows for the efficient retrieval and processing of hyperspectral images stored in our presented digital repository.

5 Conclusions and future lines

In this work, we have described the design as a web service and some computational aspects of a new unmixing-based image retrieval system for remotely hyperspectral imagery. The current version of our digital repository provides free online access through a web interface while a server manages the repository and algorithm executions. To deal with the computational cost of extracting the spectral information needed to catalog new hyperspectral images in our system, we resort to efficient implementations of spectral unmixing algorithms on GPU platforms. The system includes algorithms such as VD and HySyme for estimation of the number of endmembers in a given scene or N-FINDR and OSP-GS for the extraction of these endmembers for subsequent use (e.g., for cataloguing purposes). Our experimental results, conducted using data sets collected by the AVIRIS instrument, indicate that the proposed unmixing-based retrieval system can accurately extract hyperspectral image instances from a complex image database with sub-pixel precision. The performance analysis conducted in this work also reveals that the proposed techniques can achieve significant speedups.

As a future extension of the system, we will extend the presented implementations to multi-GPU systems to handle large collections of hyperspectral data sets, such as those that will be provided in upcoming Earth Observation missions such as EnMAP (http://www.enmap.de).

Acknowledgments This work was supported by the project AYA2011-29334-C02-02. And also, this work was partially supported by the computing facilities of Extremadura Research for Advanced Technologies (CETA-CIEMAT), funded by the European Regional Development Fund (ERDF). The CETA-CIEMAT belongs to the Spanish Ministry of Science and Innovation. Last but not least, the authors would like to take this opportunity to gratefully thank the editors and the two anonymous reviewers for their outstanding comments and suggestions, which greatly helped us to improve the technical quality and presentation of our manuscript.

References

1. Smeulders AWM, Worring M, Santini S, Gupta A, Jain R (2000) Fast dimensionality reduction and simple PCA. IEEE Trans Pattern Anal Mach Intell 22:1349–1380
2. Chang CI (2003) Hyperspectral imaging: techniques for spectral detection and classification. Kluwer Academic/Plenum Publishers, New York
3. Green RO, Eastwood ML, Sarture CM, Chrien TG, Aronsson M, Chippendale BJ, Faust JA, Pavri BE, Chovit CJ, Solis M et al (1998) Imaging spectroscopy and the airborne visible/infrared imaging spectrometer (AVIRIS). Remote Sens Environ 65(3):227–248
4. Plaza A, Chang CI (2007) High performance computing in remote sensing. Taylor & Francis, Boca Raton
5. Plaza A, Plaza J, Paz A, Sanchez S (2011) Parallel hyperspectral image and signal processing [applications corner]. Signal Process Mag IEEE 28(3):119–126

6. Bioucas-Dias J, Plaza A, Dobigeon N, Parente M, Du Q, Gader P, Chanussot J (2012) Hyperspectral unmixing overview: geometrical, statistical, and sparse regression-based approaches. IEEE J Sel Top Appl Earth Obs Remote Sens 5(2):354–379
7. Plaza A, Benediktsson JA, Boardman J, Brazile J, Bruzzone L, Camps-Valls G, Chanussot J, Fauvel M, Gamba P, Gualtieri J, Marconcini M, Tilton JC, Trianni G (2009) Recent advances in techniques for hyperspectral image processing. Remote Sens Environ 113:110–122
8. Bernabe S, Sanchez S, Plaza A, Lopez S, Benediktsson J, Sarmiento R (2013) Hyperspectral unmixing on GPUs and multi-core processors: a comparison. IEEE J Sel Top Appl Earth Obs 6(3):1386–1398
9. Sanchez S, Ramalho R, Sousa L, Plaza A (2012) Real-time implementation of remotely sensed hyperspectral image unmixing on GPUs. J Real Time Image Process 1–15. doi:10.1007/s11554-012-0269-2
10. Chang CI, Du Q (2004) Estimation of number of spectrally distinct signal sources in hyperspectral imagery. IEEE Trans Geosci Remote Sens 42(3):608–619
11. Sanchez S, Plaza A (2012) Fast determination of the number of endmembers for real-time hyperspectral unmixing on GPUs. J Real Time Image Process 1–9. doi:10.1007/s11554-012-0276-3
12. Bioucas-Dias JM, Nascimento JMP (2008) Hyperspectral subspace identification. IEEE Trans Geosci Remote Sens 46(8):2435–2445
13. Winter ME (1999) N-FINDR: an algorithm for fast autonomous spectral endmember determination in hyperspectral data. Proc SPIE 3753:266–277
14. Remon A, Sanchez S, Paz A, Quintana-Orti ES, Plaza A (2011) Real-time endmember extraction on multi-core processors. IEEE Geosci Remote Sens Lett 8:924–928
15. Harsanyi JC, Chang CI (1994) Hyperspectral image classification and dimensionality reduction: an orthogonal subspace projection. IEEE Trans Geosci Remote Sens 32(4):779–785
16. Lopez S, Horstrand P, Callico GM, Lopez JF, Sarmiento R (2012) A low-computational-complexity algorithm for hyperspectral endmember extraction: modified vertex component analysis. IEEE Geosci Remote Sens Lett 9(3):502–506
17. Sevilla J, Bernabe S, Plaza AJ, Garcia P (2012) A new digital repository for remotely sensed hyperspectral imagery with unmixing-based retrieval functionality. In: SPIE optics and photonics, satellite data compression, communication, and processing conference, vol 8514, San Diego, CA
18. Keshava N, Mustard JF (2002) Spectral unmixing. IEEE Signal Process Mag 19(1):44–57