Parallel Implementation of Spatial–Spectral Endmember Extraction on Graphic Processing Units

Luis Ignacio Jiménez, Sergio Sánchez, Gabriel Martín, Javier Plaza, *Senior Member, IEEE*, and Antonio J. Plaza, *Fellow, IEEE*

***Abstract*—The identification of pure spectral signatures (endmembers) in remotely sensed hyperspectral images has traditionally focused on the spectral information alone. Recently, techniques such as the spatial–spectral endmember extraction (SSEE) have incorporated both the spectral and the spatial information contained in the scene. Since hyperspectral images contain very detailed information in the spatial and spectral domain, the integration of these two sources of information generally comes with a significant increase in computational complexity. In this paper, we develop a new computationally efficient implementation of SSEE using commodity graphics processing units (GPUs). The relevance of GPUs comes from their very low cost, compact size, and the possibility to obtain significant acceleration factors by exploiting properly the GPU hardware architecture. Our experimental results, focused on evaluating the candidate endmembers produced by SSEE and also the computational performance of the GPU implementation, indicated significant acceleration factors that allow exploiting the SSEE method in computationally efficient fashion.**

***Index Terms*—Graphics processing units (GPUs), hyperspectral imaging, spatial–spectral endmember extraction (SSEE).**

I. Introduction

SPECTRAL unmixing [1] is an important technique for the exploitation of remotely sensed hyperspectral datasets. Over the last years, many techniques have been developed for the identification of pure spectral constituents (called endmembers in unmixing jargon) and their corresponding fractional abundances at a subpixel level [2]. The remaining problem is how to automatically identify endmembers, which are representative of both the spectral and the spatial information contained in the scene. For instance, it is generally difficult to obtain endmembers, which are representative in spatial sense, as endmember identification algorithms are often driven by the spectral information alone and are therefore sensitive to noise, outliers, and anomalous endmembers [3].

To address this issue, several strategies have been proposed in order to guide the endmember identification process to spatially homogeneous areas, expected to contain the purest signatures available in the scene [4]–[6]. For this purpose, several spectral–spatial techniques have been developed for the identification of endmembers in hyperspectral scenes.

One of the first algorithms in the literature designed to integrate the spatial and the spectral information was the automatic morphological endmember extraction (AMEE) [4], which used extended morphological operations of erosion and dilation to account for endmembers that are sufficiently pure (in spectral terms) and cover a large area (in spatial sense). The algorithm had several shortcomings, including the need to define a spatial search area around each pixel in the scene and its computational complexity. Another important method is the spatial–spectral endmember extraction (SSEE) [5], which uses spatial constraints to improve the relative spectral contrast of endmember spectra that have minimal unique spectral information, thus improving the potential for these subtle, yet potentially important endmembers to be selected. With the SSEE, the spatial characteristics of image pixels are used to increase the relative spectral contrast between spectrally similar, but spatially independent endmembers.

Finally, several spatial preprocessing (SPP) methods have been used prior to endmember identification [7]–[9]. These methods are intended to be combined with a spectral-based endmember extraction algorithm. The SPP in [7] introduces the spatial information in the endmember extraction process, so that the preprocessing can be combined with classic methods for endmember identification [10]. The main idea behind this preprocessing is to estimate, for each input pixel vector, a scalar factor, which is related to the spatial similarity between that pixel and its spatial neighbors, defined in a spatial window that defines a neighborhood around each pixel vector, and then use this scalar factor to spatially weigh the spectral information associated to the pixel. An extension of this concept was

Manuscript received April 9, 2016; revised September 17, 2016; accepted December 25, 2016. Date of publication January 8, 2017; date of current version March 22, 2017. This work was supported in part by the Junta de Extremadura (decreto 297/2014, ayudas para la realizacion de actividades de investigacion y desarrollo tecnologico, de divulgacion y de transferencia de conocimiento por los Grupos de Investigacion de Extremadura, Ref. GR15005) and in part by the computing facilities of Extremadura Research for Advanced Technologies (CETA-CIEMAT), funded by the European Regional Development Fund. The CETA-CIEMAT belongs to the Spanish Ministry of Science and Innovation. *(Corresponding author: Antonio J. Plaza.)*

L. I. Jiménez, S. Sánchez, J. Plaza, and A. J. Plaza are with the Hyperspectral Computing Laboratory, Department of Computer Technology and Communications, University of Extremadura, Cáceres E-10071, Spain (e-mail: luijimenez@unex.es; sersanmar@unex.es; jplaza@unex.es; aplaza@unex.es).

G. Martín is with the Instituto de Telecomunicações, Lisbon 1049-001 Portugal (e-mail: gabriel.hernandez@lx.it.pt).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSTARS.2016.2645718

presented in [8] in which the use of fixed spatial neighborhoods adopted by SPP was replaced by the incorporation of regions intended to better characterize the spatial context. However, the RBSPP strongly depends on a prior region growing algorithm that makes the procedure sensitive to the selected technique for region segmentation. Another more recent technique is the spatial and spectral preprocessing (SSPP) presented in [9], which integrates both spatial and spectral information at the preprocessing level. The process is divided in following four different steps.

1) Multiscale Gaussian filtering, in which the original hyperspectral image is filtered to characterize the spatial context.
2) Spatial homogeneity calculation, in which the filtered image is processed in order to obtain a spatial homogeneity index for each pixel in the original image.
3) Spectral purity index calculation, which first reduces the hyperspectral image using principal component analysis (PCA) [11] and then creates a spectral purity index using an approach similar to the one adopted by the pixel purity index algorithm in [12].
4) Fusion of spatial and spectral information, so that only pixels that are both spectrally pure and spatially homogeneous are selected, without giving priority to any of these sources of information as they are treated separately at the preprocessing stage.

All the aforementioned techniques for SSEE are characterized by their very high computational complexity, resulting from the need to account for the very rich spectral and spatial information contained in hyperspectral scenes. However, due to the regularity of their computations, most of these techniques have been efficiently implemented using latest-generation hardware accelerators such as commodity graphics processing units (GPUs). GPUs generally offer a good performance/cost ratio when compared against alternative high-performance architectures such as multicores or clusters. In addition, GPU accelerators offer the advantage that they are easy to program. In the future, they may be fully adapted to onboard processing for real-time processing (although currently their power consumption is still high for these missions, there are already some efforts toward using low-power GPU architectures in remote sensing missions). This is the main reason why we have targeted GPUs for our implementations. For instance, the SPP was implemented in GPUs in [13]. The SSPP was also implemented in GPUs in [14], whereas the RBSPP is also amenable for parallel implementation due to the regularity of its computations (with the main difficulty of the segmentation step that generally results on irregular load distribution in parallel versions). The AMEE was also implemented in GPUs in [15]. However, the SSEE (a highly representative and successful algorithm for SSEE) has not been implemented in parallel as of yet.

In this work, we develop a GPU implementation of the SSEE in [5]. In this algorithm, the hyperspectral image is processed in order to select a group of signatures that will be the endmember candidates from which the final set of endmembers are extracted. As a result, the SSEE can also be considered as an SPP algorithm. Reducing the number of candidate endmembers can significantly reduce the computational time of the subsequent endmember extraction algorithm and, therefore, the SSEE can also be used in combination with traditional endmember extraction methods as other preprocessing algorithms such as SPP, SSPP, and RBSPP. Although the SSEE is characterized by high computational complexity and processing times, we conduct several optimizations that allow us to efficiently implement this method on GPUs using NVidia's compute device unified architecture (CUDA).[1] The proposed parallel implementation has been tested on two different NVidia GPU architectures using real data. We conducted comparisons of our newly proposed SSEE implementation with the available GPU implementations of SPP and SSPP, revealing that these algorithms can be efficiently exploited in GPUs to include the spatial information into the endmember extraction process.

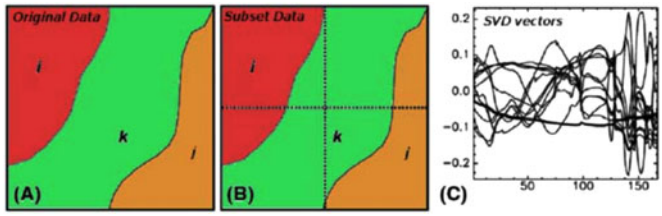

Fig. 1. First step of the SSEE algorithm (eigenvector calculation). An image region (a) containing three different components (i, j, and k) is divided in four subsets (b), which have the same size and do not overlap with each other. The SVD is used to obtain a set of eigenvectors (c).

The remainder of the paper is organized as follows. Section II describes the SSEE algorithm and its optimizations. Section III presents the GPU implementation of SSEE. The experiments conducted in order to evaluate the accuracy of the extracted signatures and the computational performance achieved by our parallel implementation are described in Section IV, which also describes a comparison with other methods. Finally, Section V concludes the paper with some remarks and hints at plausible future research lines.

II. SSEE and its Optimizations

The original SSEE algorithm can be summarized by the following steps [5].

1) First, the original hyperspectral image is partitioned into a set of subsets and singular value decomposition (SVD) is used to determine a set of eigenvectors that describe most of the spectral variance of each of the image subsets. This step is the most time-consuming. The original implementation of SSEE requires that the subsets of the image are square and do not overlap with each other (see Fig. 1).
2) Then, the entire image data are projected onto the aforementioned eigenvectors and the maxima and minima projection values are selected to determine a set of candidate endmember pixels (see Fig. 2).
3) In a third step (expansion and averaging of candidate pixels), the SSEE uses spatial constraints to combine and

[1] https://developer.nvidia.com/cuda-zone

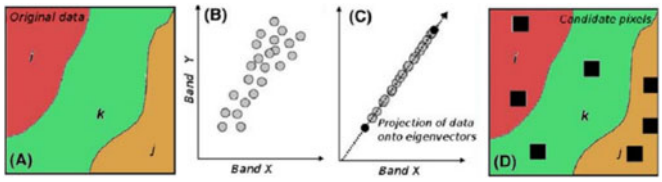

Fig. 2. Second step of the SSEE algorithm (data projection). The original image (a), represented in feature space as indicated in (b), is projected onto the eigenvectors derived in Fig. 1(c), identifying the maxima and minima projections to select a set of candidate pixels (d).

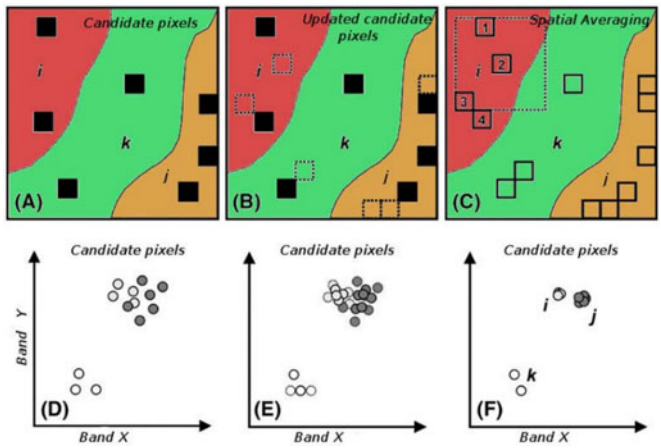

Fig. 3. Third step of the SSEE algorithm. After selecting the initial candidate pixels distributed spatially with respect to classes i, j, and k (a), the expansion begins around each candidate by selecting the pixels that are spatially close and spectrally similar to the ones already selected in (b). A spectral averaging process (using the same sliding window centered on each candidate endmember pixel) is performed in (c). In each case, the spectral distribution in feature space is shown. For instance, in (d) the candidate pixels are not averaged. The spatial–spectral averaging process starts in (e) for each class. This step continues during a number of iterations so that candidates are better grouped into spectrally separated classes (f).

average spectrally similar candidate endmember pixels. In other words, the set of candidate endmembers obtained after the previous step is now extended based on the spectral similarity between the candidates and the pixels located within a spatial window around them, followed by an averaging process also within a window of the same size the previous one. This process separates the endmember classes in spectral space, as illustrated in Fig. 3. At this point, the SSEE has obtained a set of candidate endmembers that contain the final set of endmembers. As a result, up to this point the SSEE produces a similar output to that provided by other spatial preprocessing algorithms such as SPP, RBSPP, or SSPP.

4) In the last step, the SSEE performs a ranking based on the distance from each candidate to the first of the set. After that, the endmembers can be extracted using a manual procedure, as discussed in the original SSEE contribution [5], or using spectral-based techniques for endmember extraction such as orthogonal subspace projection [16], vertex component analysis [17], or N-FINDR [18].

The SSEE method described above is characterized by its high computation complexity. Most of the processing time consumed by the algorithm (around a 99%) is spent in the candidate selection process (steps 1–3 above), due mainly to the computational cost of the SVD and the subsequent projections over the entire image. Prior to addressing our GPU implementation of SSEE, we describe two optimizations of the method aimed at reducing its computational complexity. The main modifications are focused on the method used to determine the eigenvectors, and the spatial subsampling process performed to select the initial candidate pixels. In the following, we describe the considered optimizations.

A. PCA Instead of SVD for Eigenvector Calculation

SVD is not the only projection technique that can be used to determine the eigenvectors needed for SSEE method. We studied the possibility of using a fast implementation of PCA [19] for this purpose. PCA performs a reduction while keeping most of the information contained in the scene. This has the potential to reduce the complexity of the projection calculations. In addition, PCA is highly parallelizable with many parallel implementations available.

B. Spatial Subsampling

An important optimization of the SSEE presented in [20] is to retain local endmembers together with the local eigenvectors, as these are calculated for the spatial partitions. The main difference with the original SSEE is that local endmembers are also retained and these are used to represent the spatial subsampling of the data. This approach was originally thought for larger images, in which computing the projections over the entire dataset was too expensive in computational terms. In this way, the amount of data involved in the eigenvector projections is reduced and the computational complexity is significantly improved. Here, we consider this spatial subsampling approach developed by the authors of SSEE in [20] to speed up the process. With this subsampling approach, the number of candidates is larger than in the original SSEE implementation because it tends to obtain similar candidates in different regions. However, the averaging process conducted in step 3 is expected to reduce (yet not completely remove) the redundant candidates.

To conclude this section, Fig. 4 provides a diagram with the four implementations of SSEE adopted in this work, resulting from the two considered modifications. In the figure, we have the original SSEE implementation using SVD eigenvectors projected onto the whole image (hereinafter called SSEE-SVD), the SSEE implementation using PCA eigenvectors projected onto the whole image (hereinafter called SSEE-PCA), the SSEE implementation using SVD eigenvectors projected into images obtained after spatial subsampling (hereinafter called SSEE SS-SVD), and the SSEE implementation using PCA eigenvectors projected onto the images obtained after spatial subsampling (hereinafter called SSEE SS-PCA). From these four options (which cover steps 1 and 2 of the original SSEE implementation), an extension of the candidate set and spatial averaging is conducted (step 3 in the original SSEE implementation)

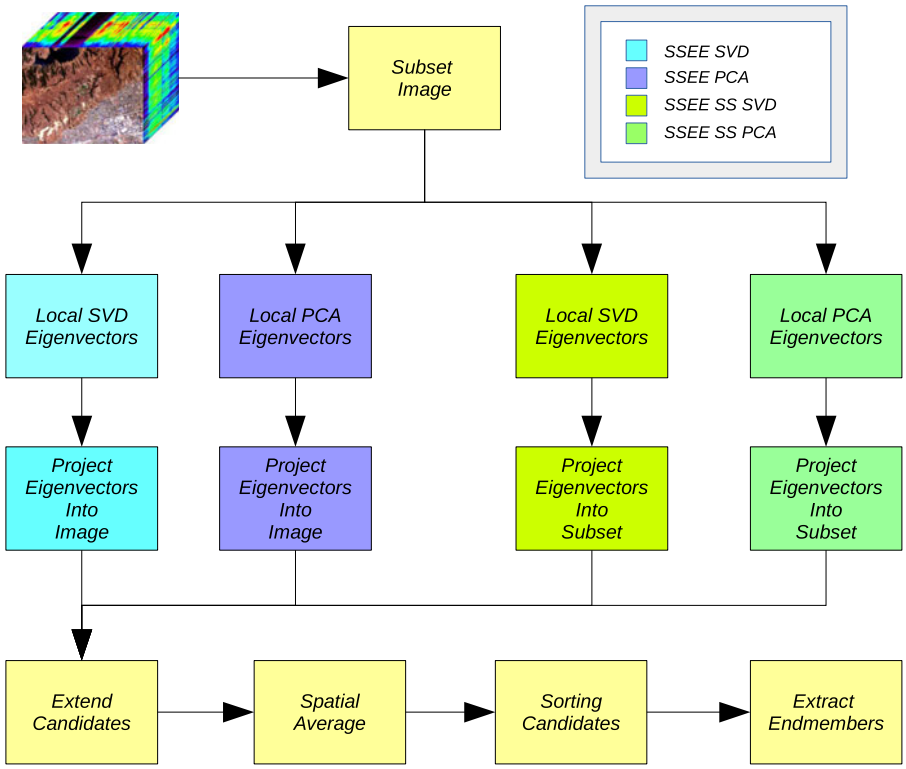

Fig. 4. Block diagram illustrating the different implementations of SSEE considered in this paper.

and finally the endmembers can be extracted after sorting the candidates and/or applying a spectral-based endmember extraction algorithm (step 4 in the original SSEE implementation).

III. GPU Implementation

The GPU implementation of SSEE has been developed using Nvidia CUDA 6.5.12 for Unix platforms, which uses the `gcc` compiler (version 4.8.2) with `-O3` as optimization level with some calls to predefined functions available in cuBLAS[2] and using Linear Algebra PACKage (LAPACK) routines.[3] Among these routines the `dgesvd` performs the SVD, meanwhile the PCA algorithm uses both the `dgesvd` and `dgemm` functions besides other functions. In the following, we describe the general GPU architecture and the individual implementation of each of the SSEE modules in Fig. 4.

A. GPU Architecture

GPUs can be abstracted in terms of a stream model, under which all datasets are represented as streams (i.e., ordered datasets). The architecture of a GPU can be seen as a set of multiprocessors (MPs). Each MP is characterized by a single instruction multiple data architecture, i.e., in each clock cycle, each processor executes the same instruction but operating on multiple data streams. Each processor has access to a local shared memory and also to local cache memories in the MP, while the MPs have access to the global GPU (device) memory. Algorithms are constructed by chaining so-called *kernels*, which operate on entire streams and are executed by an MP, 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 by several *threads*, which share data efficiently through the shared local memory and synchronize their execution for coordinating accesses to memory. As a result, there are different levels of memory in the GPU for the thread, block, and grid concepts. There is also a maximum number of threads that a block can contain but the number of threads that can be concurrently executed is much larger (several blocks executed by the same kernel can be managed concurrently at the expense of reducing the cooperation between threads since the threads in different blocks of the same grid cannot synchronize with the other threads).

B. GPU Implementation of SSEE

The GPU implementation of the different SSEE steps can be summarized as follows.

1) *Eigenvector Calculation*: A significant part of the computational time of SSEE is used to calculate the eigenvectors. This step can be performed using the SVD or the PCA. The SVD has been implemented by simply resorting to the *dgesvd* function available in the BLAS/LAPACK library, which has been experimentally evaluated to be

[2]https://developer.nvidia.com/cublas
[3]http://www.netlib.org/lapack/

computationally very fast for our purpose. For the GPU implementation of PCA, we conduct the following steps. First, a kernel that normalizes the subset data subtracting the average pixel from the subset image is developed. This kernel, called `avgXCUDA`, sets a one-dimensional (1-D) grid where the size is the number of bands of the image and the block size is equal to the maximum number of threads, which the GPU is capable of allocating. After that, a call to cuBLAS function `cublasDgemm` is performed in order to obtain the result of multiplying the data matrix by its transpose. Finally the SVD transformation is computed in the CPU, as we have experimentally tested that parallelization is not needed in order to conduct this final result in computationally efficient fashion.

2) *Data Projection*: Once the eigenvectors are calculated and have been transferred to the GPU, a call to cuBLAS function `cublasDgemm` is performed so that the full scene (or the subset image, depending on the considered method) is processed. This step is the most important in terms of computational time, specially if the subsampling is not used, and the size of the window used for the processing is very important as it can significantly increase the complexity of the computation. For this step, a kernel called `maxminProjection` has been designed to select the maxima and minima projection values of each band, using a reduction operation corresponding to the given number of bands that represents the percentage of the total defined by a threshold value svd_th. Here, the number of blocks is set to the considered number of bands and the number of threads per block is set to the maximum value allowed by the GPU device.
3) *Expansion and Averaging of Candidate Pixels*: In order to implement this step, we first compute the Euclidean norm of each pixel present in the image using a kernel called `euclideanNorm`. The number of threads for this kernel is set to the maximum capability of the device, and the number of blocks is established as the ratio between the number of pixels in the image and the size of the block plus one. A second kernel called `expandCandidatePixels` is designed to perform the expansion of the candidate set within the range of the window size, where parameter ws (window size) is set according to the similarity between candidates already established and other pixels in that range. Finally, a kernel called `averageStep` computes the averaging process between the candidate pixels located within the spatial window. Both kernels are defined by a 2-D_ grid set in which the number of rows and columns are, respectively, defined to the number of rows and columns of the original image, and the maximum number of threads is set as the block size.
4) *Ranking*: The last step just ranks the candidate pixels by computing the distance between them. In this paper, we use a kernel called `BitonicSort` to calculate the ranking by performing a bitonic sorting algorithm in the GPU. Here, the number of threads is empirically set to 256 and the number of blocks is calculated according to the following expression:

$$n\_\text{blocks} = \exp\left(\log_2 \left\lfloor \frac{n\_\text{pixels}}{n\_\text{threads}} \right\rfloor\right)$$

where $n_$blocks, $n_$pixels, and $n_$threads, respectively, denote the number of blocks, the number of pixels, and the number of threads. We emphasize that this last step is optional and can be replaced by an endmember extraction algorithm since steps 1–3 provide a SPP of the original image. We also emphasize that this step represents a negligible amount of the total computation time, even for large datasets and candidate pixel sets.

IV. Experimental Results

Our experimental results were conducted using a subset of the dataset collected by the airborne visible infrared imaging spectrometer (AVIRIS), operated by the NASA's Jet Propulsion Laboratory, over the Cuprite mining district in Nevada in the summer of 1997 (these data are available online from http://aviris.jpl.nasa.gov). The portion used in the experiments corresponds to a 350 $\times$ 350 pixel subset of the sector labeled as f970619t01p02r02 in the online data (available online: ftp://popo.jpl.nasa.gov/pub/free_data/f970619t01p02r02c_rfl.tar), which contains four 512 $\times$ 512 images comprising 224 spectral bands in the range from 400 to 2500 nm and a total size of around 50 MB. The 350 $\times$ 350 pixel subset was extracted at the upper rightmost corner of the fourth 512 $\times$ 512 pixel image available in the aforementioned online file. Water absorption bands as well as bands with low signal-to-noise ratio were removed, retaining 188 spectral bands (after specifically removing bands 1–4, 105–115, and 150–170. The main reason for selecting this specific subset in experiments is the fact that this particular area is well understood mineralogically, with many reference ground signatures of the main minerals of interest present in the scene available in the form of a United States Geological Survey (USGS) library (http://speclab.cr.usgs.gov/spectral-lib.html), which has been used in this paper to perform the accuracy evaluation.

Two kinds of experiments were conducted in order to evaluate the quality of the candidate pixels as well as the computational complexity of our GPU implementation of SSEE. First, we analyzed the quality of the candidates selected by different approaches (not only our SSEE implementations, but also the SPP and SSPP) as compared to a set of reference signatures available in the USGS library, using the spectral angle distance (SAD) as a quantitative metric. We emphasize that the SSEE can be seen as a spatial preprocessing method, much like SPP and SSPP. As a result, a comparison with these methods is established to properly analyze the quality of the candidate pixels provided by these different methods. For our purposes, we have selected a total of 25 reference mineral signatures from the USGS library, which are displayed in Fig. 5. Then, we analyzed the parallel performance of the GPU implementations, using two different GPU architectures. Since the SSEE algorithm requires to split the original image in square disjoints parts of the same size, the

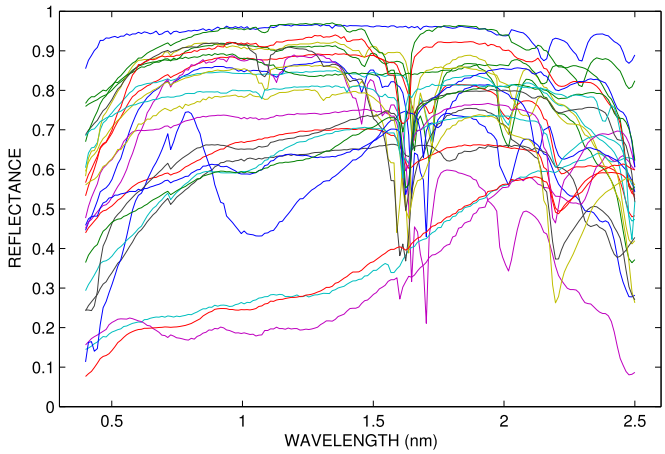

Fig. 5. Set of 25 reference USGS signatures selected for our experiments: Alunite GDS84 Na03, Alunite GDS83 Na63, Alunite GDS82 Na82, Alunite AL706 Na, Alunite HS295.3B, Alunite SUSTDA-20, Buddingtonite GDS85 D-206, Calcite WS272, Calcite HS48.3B, Chalcedony CU91-6A, Chlorite HS179.3B, Chlorite SMR-13.a 104-150, Dickite NMNH106242, Halloysite NMNH106236, Jarosite GDS99 K-y 200C, Kaolinite KGa-1 (wxyl), Kaolinite KGa-2 (pxyl), Kaolin/Smect KLF506 95%K, Montmorillonite SWy-1, Montmorillonite SAz-1, Muscovite GDS107, Muscovite HS146.3B, Muscovite HS24.3, Nontronite GDS41, and Pyrophyllite PYS1A fine g.

influence of this spatial subsetting has been evaluated using four different sizes (35×35, 50×50, 70×70, and 175×175)—which are multiples of the 350×350 pixel scene considered in experiments—to generate the spatial square disjoint subsets of the original image during all the experimental validation. Our reason to consider different sizes (from smaller to greater) is to analyze the difference obtained in the algorithm performance with spatially closer candidates. In the following, we summarize the conducted experiments.

A. Quality of Candidate Pixels

In order to evaluate the quality of the candidate pixels selected by the GPU implementation of SSEE as compared to the reference USGS signatures, we considered two different matching algorithms [9]. The first matching algorithm obtains the average matching results after 100 iterations, reordering both sets (i.e., candidate pixels and reference USGS signatures) randomly in each iteration and selecting the best match (in terms of SAD), from the first reference signature to the last, as shown in Table I. This table includes the results of the four considered implementations of SSEE in Fig. 4, as well as the results provided by SPP and SSPP. The second matching algorithm matches a pair of signatures (i.e., one from the set of candidate pixels and another one from the reference signatures) by taking the minima SAD distance of all possible combinations, as shown in Table II. In both cases, the 25 mineral signatures from the USGS library shown in Fig. 5 are used as reference signatures for the matching. In order to establish a fair comparison, we need to compare the number of candidate signatures that are matched to USGS reference signatures. However, a problem is how to decide if a candidate pixel was matched to a reference USGS signature. In our experiments, we consider that a USGS signature is found by the candidate when their SAD is below 10° (the worst case for the SAD between two signatures is 90°). Based on the aforementioned observations, we use the following metric to show the percentage of success S of a certain method:

$$S(\%) = \frac{\frac{m}{P} + \left[1 - \frac{n}{N}\right]}{2} \times 100$$

where n is the number of candidate pixels extracted by the method, N is the total number of pixels in the original image, m is the number of signatures, which have been successfully matched (i.e., they are below the SAD threshold of 10°), and P is the number of reference USGS signatures. Tables I and II both include the results of the aforementioned metric. In the case of SPP, the number of candidates pixels is equal to the size of the original image as the SPP does not perform a reduction in the number of candidates, but for the SSPP and all variants of SSEE the number of candidate pixels is smaller than the number of pixels in the original image.

As shown by Tables I and II, some of the 25 reference USGS signatures could not be matched by the candidate pixels provided by different methods, e.g., *Alunite HS295.3B* or *Chlorite SMR-13.a 104–150*. This is because it is difficult to guarantee that these signatures are present in the original image, as some minerals are represented by different alterations in the set of 25 USGS reference signatures used for validation. In any event, most of thealgorithms were able to provide candidate pixels that were similar, spectrally, to the USGS reference signatures, even when considering a very string similarity threshold of 10°. In general terms, we can conclude from Tables I and II that the SSEE provided good results in most cases, with the subsampling implementations (SSEE SS-SVD and SSEE SS-PCA) providing generally worse results than the implementations working with the full image (SSEE-SVD and SSEE-PCA). Interestingly, the SSEE SS-PCA provides better results than the SSEE SS-SVD, even if the number of candidate pixels is much smaller for every window size tested. Overall, the best results are obtained by the SPP method due to the fact that it does not reduce the number of candidate pixels, but it is remarkable that the SSEE-PCA method obtains very similar results using a much smaller set of candidate pixels, which will have a strong impact in the computational performance results discussed in the following section.

B. Computational Performance

Before describing the computational performance results obtained, we emphasize that all the GPU implementations of SSEE, SPP, and SSPP discussed in this section provide exactly the same results as their serial versions, reported in the previous section. In order to evaluate the computational performance, several experiments were conducted in the following two different architectures.

1) Architecture 1 is based on an Intel Core i7 920 CPU at 2.67 GHz and 6 GB of RAM with a GPU NVidia GTX 580, which features 512 processor cores operating at 1.54 GHz.

TABLE I
AVERAGE SPECTRAL ANGLE DISTANCE (IN DEGREES)

	SSEE-SVD				SSEE-PCA				SSEE SS-SVD				SSEE SS-PCA				SPP				SSPP
Window Size	35x35	50x50	70x70	175x175	35x35	50x50	70x70	175x175	35x35	50x50	70x70	175x175	35x35	50x50	70x70	175x175	3x3	5x5	7x7	9x9	
Number of Candidates	2416	1795	1310	403	204	140	77	28	19203	9917	5047	716	1284	604	292	42	122500	122500	122500	122500	3653
Alunite GDS84 Na03	7,467	7,088	7,472	8,756	**5,735**	5,965	6,361	9,241	8,295	8,566	7,528	10,526	8,036	8,632	7,572	10,734	5,819	5,942	6,041	6,041	6,863
Alunite GDS83 Na63	7,660	7,676	8,165	9,398	6,886	**6,660**	7,796	10,204	9,306	8,906	8,530	10,741	9,370	8,614	8,784	11,079	6,876	6,955	7,033	7,035	9,449
Alunite GDS82 Na82	5,089	5,077	5,572	6,871	4,669	**4,107**	5,223	8,165	6,790	6,391	6,041	8,674	6,856	6,102	6,271	8,945	4,302	4,395	4,491	4,474	6,958
Alunite AL706 Na	5,713	5,847	6,644	7,848	**4,268**	5,229	6,286	8,328	7,915	7,161	7,615	8,934	8,054	6,702	7,873	9,373	4,794	4,820	4,852	4,844	7,320
Alunite HS295.3B	15,044	14,865	15,465	17,039	11,398	11,202	11,724	17,443	16,006	16,086	15,485	18,640	15,481	16,242	15,428	18,869	12,196	12,486	12,725	12,737	**11,002**
Alunite SUSTDA-20	6,153	6,195	6,748	7,916	5,427	**5,352**	6,385	8,810	7,928	7,435	7,292	9,653	7,987	7,083	7,529	9,736	5,434	5,501	5,562	5,562	7,763
Buddingtonite GDS85 D-206	3,793	3,932	3,904	4,854	3,924	3,918	4,049	5,553	3,986	3,822	4,025	5,185	3,821	3,897	3,913	5,223	3,781	3,778	**3,777**	**3,777**	3,863
Calcite WS272	7,237	7,440	8,341	9,543	**5,052**	7,035	7,948	10,066	9,497	8,706	9,561	10,644	9,714	8,181	9,904	10,782	5,252	5,307	5,383	5,380	8,647
Calcite HS48.3B	7,761	7,976	8,863	10,082	**5,503**	7,644	8,563	10,787	9,633	9,247	9,903	11,099	9,918	8,686	10,442	11,262	5,559	5,624	5,699	5,700	8,703
Chalcedony CU91-6A	4,242	4,425	5,194	6,275	**2,918**	4,666	5,019	7,139	6,326	5,559	6,571	7,341	6,526	5,042	6,695	7,442	2,993	3,035	3,061	3,057	5,717
Chlorite HS179.3B	15,833	16,155	17,932	22,978	15,545	15,192	17,528	22,276	16,698	16,896	16,504	23,484	16,697	16,914	16,527	23,344	14,796	15,001	15,129	15,146	**13,868**
Chlorite SMR-13.a 104-150	20,676	21,056	22,675	27,574	20,506	20,018	22,767	27,151	21,647	21,833	21,400	27,890	21,629	21,852	21,359	27,954	19,695	19,893	20,021	20,018	**18,821**
Dickite NMNH106242	7,221	7,272	7,769	8,819	**6,563**	6,664	7,490	9,592	8,756	8,249	8,299	10,079	8,807	7,957	8,512	10,302	6,618	6,652	6,704	6,699	8,664
Halloysite NMNH106236	12,106	11,733	12,097	13,587	**9,376**	9,548	9,823	13,546	13,127	13,303	12,191	15,314	12,862	13,158	12,285	15,532	10,026	10,237	10,388	10,385	10,816
Jarosite GDS99 K-y 200C	8,637	8,607	8,827	11,015	8,531	8,845	9,838	10,630	8,763	8,673	8,951	11,421	8,768	8,659	8,981	11,487	8,357	8,380	8,392	8,392	**8,245**
Kaolinite KGa-1 (wxyl)	5,296	5,455	6,246	7,460	**3,755**	5,082	5,928	8,138	7,485	6,688	7,439	8,897	7,648	6,195	7,674	8,925	4,199	4,249	4,285	4,297	6,848
Kaolinite KGa-2 (pxyl)	4,393	4,573	5,360	6,489	**3,052**	4,466	5,110	7,005	6,481	5,750	6,603	7,875	6,712	5,252	6,859	8,000	3,386	3,435	3,455	3,461	5,611
Kaolin/Smect KLF506 95%K	3,835	3,619	4,249	6,946	4,137	5,363	5,885	6,984	3,456	3,545	3,803	7,194	3,505	3,520	3,771	6,990	**2,799**	2,853	2,867	2,885	3,080
Montmorillonite SWy-1	5,922	6,113	6,937	8,045	**4,234**	6,006	6,681	8,812	8,074	7,314	8,156	9,231	8,282	6,808	8,442	9,257	4,507	4,550	4,594	4,597	7,232
Montmorillonite SAz-1	8,733	8,688	9,016	10,159	8,348	**7,867**	8,800	11,163	10,018	9,808	9,239	11,542	9,947	9,611	9,425	11,772	7,987	8,064	8,142	8,150	9,807
Muscovite GDS107	5,191	5,084	5,682	8,118	5,571	6,314	6,628	8,312	4,922	4,859	5,299	8,033	4,992	5,037	5,282	8,360	4,092	4,055	4,133	4,142	**4,047**
Muscovite HS146.3B	8,205	8,003	8,953	12,232	8,090	9,335	10,274	12,119	8,164	8,035	8,251	12,331	8,230	8,190	8,412	12,387	7,716	7,843	7,902	7,896	**7,415**
Muscovite HS24.3	6,213	6,026	7,039	10,765	6,138	7,422	8,304	10,275	6,120	5,999	6,196	10,954	6,184	6,148	6,371	10,601	5,788	5,818	5,833	5,831	**5,365**
Nontronite GDS41	14,366	14,833	16,429	21,576	14,205	13,372	16,609	21,054	15,455	15,699	15,027	22,048	15,457	15,756	15,055	21,846	13,365	13,583	13,741	13,727	**12,362**
Pyrophyllite PYS1A fine g	6,674	6,869	7,675	8,862	**4,773**	6,616	7,338	9,178	8,827	8,080	8,949	9,841	8,919	7,530	9,238	10,036	4,892	4,984	5,031	5,041	7,905
Mean	8,138	8,184	8,930	10,928	7,144	7,756	8,734	11,279	9,347	9,064	9,154	11,903	9,376	8,871	9,304	12,010	**7,009**	7,098	7,170	7,171	8,255
S(%)	89,014	89,267	89,465	79,836	91,917	**91,943**	89,969	75,989	80,162	85,952	87,940	71,708	89,476	89,753	87,881	69,983	——	——	——	——	88,509

Between the Set of Candidate Pixels Obtained by Different Spatial Preprocessing Methods (SSEE, SPP, and SSPP) and the USGS Reference Signatures for the AVIRIS Cuprite Image

TABLE II
SPECTRAL ANGLE DISTANCES (IN DEGREES)

	SSEE-SVD				SSEE-PCA				SSEE SS-SVD				SSEE SS-PCA				SPP				SSPP
Window Size	35x35	50x50	70x70	175x175	35x35	50x50	70x70	175x175	35x35	50x50	70x70	175x175	35x35	50x50	70x70	175x175	3x3	5x5	7x7	9x9	
Number of Candidates	2416	1795	1310	403	204	140	77	28	19203	9917	5047	716	1284	604	292	42	122500	122500	122500	122500	3653
Alunite GDS84 Na03	7,467	7,088	7,472	8,761	5,735	5,965	**5,500**	6,444	8,169	8,545	7,506	10,477	7,987	8,632	7,481	10,648	5,819	5,942	6,041	6,041	6,804
Alunite GDS83 Na63	7,662	7,678	8,207	9,405	6,886	**6,663**	8,227	9,203	9,342	8,981	8,696	12,130	9,344	8,635	8,797	11,508	6,939	7,015	7,070	7,070	9,518
Alunite GDS82 Na82	5,087	5,075	5,454	6,861	4,669	**4,098**	5,014	6,545	6,692	6,310	5,707	7,415	6,836	6,052	6,195	7,891	4,224	4,285	4,365	4,365	6,905
Alunite AL706 Na	5,720	5,847	6,666	7,842	**4,268**	5,228	6,031	7,392	7,925	7,075	7,364	8,342	8,072	6,661	7,814	8,690	4,836	4,877	4,911	4,911	7,270
Alunite HS295.3B	15,044	14,865	15,465	17,049	11,398	11,202	11,084	21,917	16,076	16,115	15,634	19,799	15,518	16,242	15,563	20,054	12,196	12,660	12,834	12,834	**11,002**
Alunite SUSTDA-20	6,153	6,203	6,736	7,912	5,427	**5,359**	6,202	7,643	7,938	7,404	7,084	8,680	7,983	7,052	7,443	9,050	5,424	5,526	5,561	5,561	7,753
Buddingtonite GDS85 D-206	3,793	3,932	3,904	4,750	3,924	3,918	4,045	4,767	3,964	3,822	3,921	4,881	3,819	3,897	3,913	4,865	3,781	3,778	**3,777**	**3,777**	3,863
Calcite WS272	7,258	7,446	8,413	9,549	**5,053**	7,060	8,314	8,912	9,601	8,819	9,831	10,644	9,709	8,208	9,967	10,940	5,323	5,381	5,432	5,432	8,772
Calcite HS48.3B	7,785	7,983	8,970	10,090	**5,503**	7,665	9,074	9,569	10,067	9,352	10,542	12,469	9,971	8,802	10,442	11,810	5,678	5,753	5,827	5,827	8,900
Chalcedony CU91-6A	4,140	4,415	4,808	6,271	**2,918**	4,644	4,869	5,639	6,085	5,341	6,389	6,151	6,490	4,960	6,656	6,566	2,959	2,988	3,021	3,021	5,669
Chlorite HS179.3B	15,896	16,562	17,964	22,946	15,547	15,192	20,597	24,182	16,698	16,901	16,640	24,087	16,697	16,916	16,640	24,042	14,805	14,951	15,134	15,134	**14,126**
Chlorite SMR-13.a 104-150	20,841	21,470	22,762	27,658	20,653	20,018	27,473	28,595	21,647	21,839	21,545	28,741	21,629	21,854	21,545	28,693	19,755	20,029	20,126	20,126	**19,041**
Dickite NMNH106242	7,221	7,277	7,804	8,821	**6,563**	6,670	7,420	8,616	8,828	8,314	8,415	9,962	8,829	7,975	8,518	9,799	6,659	6,689	6,787	6,787	8,761
Halloysite NMNH106236	12,106	11,733	12,097	13,595	**9,376**	9,548	9,408	19,196	13,332	13,423	12,371	16,990	12,878	13,205	12,375	17,066	10,026	10,197	10,360	10,360	10,816
Jarosite GDS99 K-y 200C	8,589	8,607	8,796	10,992	8,959	8,797	9,806	10,405	8,763	8,673	8,951	11,741	8,768	8,659	8,981	11,803	8,357	8,380	8,392	8,392	**8,245**
Kaolinite KGa-1 (wxyl)	5,299	5,446	6,230	7,455	**3,755**	5,066	5,736	6,949	7,224	6,603	7,191	7,754	7,659	6,148	7,614	8,070	4,192	4,277	4,308	4,308	6,639
Kaolinite KGa-2 (pxyl)	4,390	4,562	5,307	6,483	**3,052**	4,440	4,950	6,079	6,101	5,556	6,434	6,688	6,727	5,192	6,781	7,007	3,335	3,346	3,363	3,363	5,461
Kaolin/Smect KLF506 95%K	3,821	3,619	4,200	6,701	4,137	5,363	5,509	5,192	3,454	3,545	3,774	4,673	3,502	3,489	3,771	4,673	**2,637**	2,721	2,763	2,763	3,080
Montmorillonite SWy-1	5,941	6,115	6,995	8,041	**4,234**	6,029	6,485	7,463	8,299	7,247	8,154	8,242	8,268	6,757	8,470	8,608	4,482	4,601	4,655	4,655	7,253
Montmorillonite SAz-1	8,733	8,688	9,045	10,164	8,348	**7,867**	9,251	14,627	10,161	9,933	9,476	13,232	9,965	9,673	9,433	13,173	8,114	8,237	8,178	8,178	9,911
Muscovite GDS107	5,214	5,084	5,782	8,258	5,571	6,314	6,623	7,026	4,925	4,859	5,315	8,419	4,998	5,049	5,282	8,417	**3,935**	4,024	4,076	4,076	4,047
Muscovite HS146.3B	8,206	7,984	8,919	12,373	8,073	10,066	10,412	11,964	8,164	8,042	8,251	13,013	8,230	8,206	8,434	12,990	7,867	7,901	7,946	7,946	**7,423**
Muscovite HS24.3	6,218	5,868	6,749	10,807	6,058	7,006	8,406	10,273	6,120	5,990	6,196	11,236	6,182	6,153	6,343	11,251	5,788	5,818	5,830	5,830	**5,354**
Nontronite GDS41	14,434	14,568	16,429	21,561	13,621	13,372	18,601	22,426	15,455	15,684	14,538	22,828	15,457	15,750	14,538	22,805	13,324	13,502	13,622	13,622	**12,093**
Pyrophyllite PYS1A fine g	6,685	6,871	7,781	8,865	**4,773**	6,640	7,315	8,294	9,104	8,108	9,170	9,542	8,905	7,551	9,301	9,541	4,872	4,942	5,021	5,021	7,987
Mean	8,148	8,199	8,918	10,928	7,140	7,768	9,054	11,173	9,365	9,059	9,164	11,925	9,377	8,869	9,292	11,998	**7,013**	7,113	7,176	7,176	8,268
S(%)	89,014	89,267	89,465	79,836	**91,917**	89,943	89,969	81,989	78,162	85,952	85,940	73,708	89,476	89,753	87,881	73,983	——	——	——	——	88,509

Between the Set of Candidate Pixels Obtained by Different Spatial Preprocessing Methods (SSEE, SPP, and SSPP) and the USGS Reference Signatures for the AVIRIS Cuprite Image

2) Architecture 2 is a cluster made up of Intel Xeon CPUs E5645 at 2.40 GHz and 24 GB DDR3, divided in 12 modules of 2 GB each, with an Nvidia TESLA S2070 GPU with 2 M2075 per node (we only use one of the nodes for experiments).

A reason for including Architecture 2 in experiments (in addition to the standard Architecture 1) is that the TESLA S2070 GPU performs error checking, making the execution a little slower in some cases but providing more reliable results than the GTX 580 GPU. We believe that the two considered GPU architectures, although not latest generation, are quite general and illustrative, so that they provide quite representative results. In both cases, the serial algorithms were executed in one of the available cores, and the parallel times were measured using all the resources from each GPU platform. The speedups are calculated between each CPU/GPU pair using the mean values after ten Monte Carlo runs. In our context, the Monte Carlo approach was used to obtain a representative processing time after several runs of the parallel version (the execution time slightly differs from one run to another due to threads scheduling and different initialization in an inner sorting step of the algorithm). In any event, we have experimentally tested that the standard deviation between different runs is very small so our Monte Carlo runs confirm that the parallel execution times remain very stable.

Table III shows the performance of the parallel implementations of SSEE in the two considered architectures (for a comparison with the parallel versions of SPP and SSPP, we refer to works [13] and[14], respectively, describing the GPU implementations of these algorithms). From Table III, we can conclude that the SSEE implementations, which involve the PCA instead of the SVD, exhibit a lower computational cost, mostly due to the data volume reduction in addition to a higher parallelization order. SVD approaches are limited by this operation because only parallelization of very large datasets can obtain a significant performance improvement [21] when executed on GPUs. If we take into account the accuracy results reported in Section IV-A,

TABLE III
EXECUTION TIMES (SECONDS) AND PARALLEL PERFORMANCE (SPEEDUP)

	SVD		PCA		SSE SS-SVD		SSEE SS-PCA	
Architecture 1: Intel core i7 920 CPU at 2.67 GHz and 6 GB of RAM with a GPU NVidia GTX 580								
Window size	Total time (s)	Speedup	Total time (s)	Speedup	Total time (s)	Speedup	Total time (s)	Speedup
35 × 35	7,2530	15,1616	8,7473	11,4655	10,2661	5,0818	4,0162	1,9642
50 × 50	5,2137	11,4076	4,3565	11,4519	7,4948	6,1597	2,1512	2,7603
70 × 70	4,0422	9,1843	2,3778	11,0915	5,8285	7,5405	1,3844	3,6789
175 × 175	4,4371	5,1112	0,6876	8,9735	5,2349	6,7613	0,6262	6,4053
Architecture 2: Intel Xeon CPU E5645 at 2.40 GHz and 24 GB DDR3, with an Nvidia TESLA S2070 GPU								
Window size	Total time (s)	Speedup	Total time (s)	Speedup	Total time (s)	Speedup	Total time (s)	Speedup
35 × 35	10,1880	61,2082	8,7600	70,4919	14,5560	3,9652	4,5600	3,5300
50 × 50	8,0400	38,6563	4,5230	67,6003	11,3480	4,4413	2,5920	5,4471
70 × 70	7,3590	22,8132	2,5300	62,8731	10,0100	4,9502	1,7550	7,5379
175 × 175	7,6640	5,9659	0,9030	35,3189	8,6188	4,8518	0,8830	16,1687

For GPU Implementation of Different SSEE Approaches, Considering Four Different Spatial Subset Sizes, After Processing the AVIRIS Cuprite Image using Architecture 1 (Top Row) and Architecture 2 (Bottom Row) Ten Monte-Carlo runs are conducted in each experiment, and the average values are reported.

we can conclude that PCA-based implementations clearly improve the results obtained by the SVD-based methods. The results reported on Architecture 2 also exhibit lower performance for subsampling-based approaches, although the overall computational times achieved are better. This is due to the significant improvement achieved by the projection operation in this particular architecture. Overall, the SSEE SS-PCA achieves better performance with larger window sizes, which is an important observation in terms of scalability.

V. CONCLUSION AND FUTURE LINES

In this paper, we have presented a new implementation of the SSEE algorithm, which can be considered as an SPP module able to select suitable candidate pixels for subsequent endmember extraction. Several different implementations have been considered, based on the way eigenvectors are extracted and the amount of data that are projected onto them. The proposed GPU implementations are compared with other well-established spatial preprocessing methods, such as SPP and SSPP. The results obtained suggest that the GPU implementation of SSEE provides competitive results with such algorithms and allows a very significant reduction of computational times while retaining high-quality candidate pixels for endmember extraction purposes. Our future work will focus on the addition of new spatial–spectral preprocessing algorithms to the comparison and on the development of strategies for parallelization of preprocessing algorithms on GPUs and other high-performance computing architectures such as field programmable gate arrays. In the future, we will also perform additional experiments with more recent GPU hardware.

ACKNOWLEDGMENT

The authors would like to thank the Editors and Reviewers for their outstanding comments and suggestions for improvement, which have greatly helped them to improve the technical quality and presentation of the manuscript.

REFERENCES

[1] N. Keshava and J. F. Mustard, "Spectral unmixing," *IEEE Signal Process. Mag.*, vol. 19, no. 1, pp. 44–57, Jan. 2002.
[2] J. M. Bioucas-Dias *et al.*, "Hyperspectral unmixing overview: Geometrical, statistical and sparse regression-based approaches," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 5, no. 2, pp. 354–379, Apr. 2012.
[3] A. Plaza, P. Martinez, R. Perez, and J. Plaza, "A quantitative and comparative analysis of endmember extraction algorithms from hyperspectral data," *IEEE Trans. Geosci. Remote Sens.*, vol. 42, no. 3, pp. 650–663, Mar. 2004.
[4] A. Plaza, P. Martinez, R. Pérez, and J. Plaza, "Spatial/spectral endmember extraction by multidimensional morphological operations," *IEEE Trans. Geosci. Remote Sens.*, vol. 40, no. 9, pp. 2025–2041, Sep. 2002.
[5] D. M. Rogge, B. Rivard, J. Zhang, A. Sanchez, J. Harris, and J. Feng, "Integration of spatial–spectral information for the improved extraction of endmembers," *Remote Sens. Environ.*, vol. 110, no. 3, pp. 287–303, 2007.
[6] S. Lopez, J. F. Moure, A. Plaza, G. M. Callico, J. F. Lopez, and R. Sarmiento, "A new preprocessing technique for fast hyperspectral endmember extraction," *IEEE Geosci. Remote Sens. Lett.*, vol. 10, no. 5, pp. 1070–1074, Sep. 2013.
[7] M. Zortea and A. Plaza, "Spatial preprocessing for endmember extraction," *IEEE Trans. Geosci. Remote Sens.*, vol. 47, no. 8, pp. 2679–2693, Aug. 2009.
[8] G. Martin and A. Plaza, "Region-based spatial preprocessing for endmember extraction and spectral unmixing," *IEEE Geosci. Remote Sens. Lett.*, vol. 8, no. 4, pp. 745–749, Jul. 2011.
[9] G. Martin and A. Plaza, "Spatial-spectral preprocessing prior to endmember identification and unmixing of remotely sensed hyperspectral data," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 5, no. 2, pp. 380–395, Apr. 2012.
[10] J. Plaza, E. M. T. Hendrix, I. Garcia, G. Martin, and A. Plaza, "On endmember identification in hyperspectral images without pure pixels: A comparison of algorithms," *J. Math. Imaging Vis.*, vol. 42, no. 2–3, pp. 163–175, 2012.
[11] J. A. Richards and X. Jia, *Remote Sensing Digital Image Analysis*. Berlin, Germany: Springer-Verlag, 1999.
[12] J. W. Boardman, F. A. Kruse, and R. O. Green, "Mapping target signatures via partial unmixing of AVIRIS data," in *Proc. Jet Propulsion Laboratory Airborne Earth Sci. Workshop*, 1995, pp. 23–26.

[13] J. Delgado, G. Martin, J. Plaza, L. I. Jimenez, and A. Plaza, "Fast spatial preprocessing for spectral unmixing of hyperspectral data on graphics processing units," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 9, no. 2, pp. 952–961, Feb. 2016.

[14] L. I. Jimenez *et al.*, "GPU implementation of spatial-spectral preprocessing for hyperspectral unmixing," *IEEE Geosci. Remote Sens. Lett.*, vol. 13, no. 11, pp. 1671–1675, Nov. 2016. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7563405 &isnum ber=4357975

[15] J. Setoain, M. Prieto, C. Tenllado, A. Plaza, and F. Tirado, "Parallel morphological endmember extraction using commodity graphics hardware," *IEEE Geosci. Remote Sens. Lett.*, vol. 4, no. 3, pp. 441–445, Jul. 2007.

[16] J. C. Harsanyi and C.-I. Chang, "Hyperspectral image classification and dimensionality reduction: An orthogonal subspace projection approach," *IEEE Trans. Geosci. Remote Sens.*, vol. 32, no. 4, pp. 779–785, Jul. 1994.

[17] J. M. P. Nascimento and J. M. Bioucas-Dias, "Vertex component analysis: A fast algorithm to unmix hyperspectral data," *IEEE Trans. Geosci. Remote Sens.*, vol. 43, no. 4, pp. 898–910, Apr. 2005.

[18] M. Winter, "N-FINDR: An algorithm for fast autonomous spectral endmember determination in hyperspectral data," *Proc. SPIE*, vol. 3753, pp. 266–270, 1999.

[19] I. Jolliffe, *Principal Component Analysis*. Hoboken, NJ, USA: Wiley, 2002.

[20] D. Rogge, M. Bachmann, B. Rivard, and J. Feng, "Spatial sub-sampling using local endmembers for adapting OSP and SSEE for large-scale hyperspectral surveys," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 5, no. 1, pp. 183–195, Feb. 2012.

[21] S. Lahabar and P. J. Narayanan, "Singular value decomposition on GPU using CUDA," in *Proc. IEEE Int. Symp. Parallel Distrib. Process.*, May 2009, pp. 1–10. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5161058 &isnum ber=5160846

Author's photographs and biographies are not available at the time of publication.