GPU IMPLEMENTATION OF A CONSTRAINED HYPERSPECTRAL CODED APERTURE ALGORITHM FOR COMPRESSIVE SENSING

Sergio Bernabé[1], *Gabriel Martín*[1], *José M. P. Nascimento*[1,2], *José M. Bioucas-Dias*[1], Antonio Plaza[3] and Vítor Silva[4]

[1]Instituto de Telecomunicações, Instituto Superior Técnico, Universidade de Lisboa, 1049-001 Lisbon, Portugal
[2]ISEL - Instituto Superior de Engenharia de Lisboa, Instituto Politécnico de Lisboa 1959-007 Lisbon, Portugal
[3]Hyperspectral Computing Laboratory, Department of Technology of Computers and Communications, University of Extremadura, E-10003 Cáceres, Spain
[4]Instituto de Telecomunicações, Departamento de Engenharia Electrotécnica e de Computadores, Universidade de Coimbra, 3030-290 Coimbra, Portugal

ABSTRACT

In this paper, a parallel implementation of a previously constrained hyperspectral coded aperture (CHYCA) algorithm for compressive sensing on graphics processing units (GPUs) is proposed. CHYCA method combines the ideas of spectral unmixing and compressive sensing exploiting the high spatial correlation that can be observed in the data and the generally low number of endmembers needed in order to explain the data. The performance of CHYCA relies which does not depend on the tuning of a regularization parameter, which is a time consuming task offering good performance compared with a previously hyperspectral coded aperture (HYCA) method. The proposed implementation exploits the GPU architecture at low level, thus taking full advantage of the computational power of GPUs using shared memory and coalesced accesses to memory. Experimental results using simulated data reveals speedups up to 56 times, with regards to serial implementation.

Index Terms— Hyperspectral imaging, compressive sensing (CS), coded aperture, graphics processing units (GPUS).

1. INTRODUCTION

Hyperspectral imaging instruments allow data collection in hundreds or even thousands of spectral bands (at different wavelength channels) for the same area on the Earth surface [1]. For instance, NASA is continuously gathering imagery data with instruments such as the Jet Propulsion Laboratory's Airborne Visible Infra-Red Imaging Spectrometer (AVIRIS), which is able to record the visible and near-infrared spectrum (wavelength region from 0.4 to 2.5 micrometers) of reflected light in an area of 2 to 12 kilometers wide and several kilometers long, using 224 spectral bands. The resulting multidimensional data cube typically comprises several GBs per flight. As a result, the computational requirements needed to store, manage and process these images are enormous [2].

In addition to extremely large dimensionality, another problem in the analysis of hyperspectral data is the presence of mixed pixels [3], which arise when the low spatial resolution of the sensor is not enough to separate spectrally distinct materials. Mixed pixels can also result when distinct materials are combined into a homogeneous or intimate mixture [4]. The spectra of the individual materials which forms the mixed pixel are often called endmembers in the hyperspectral imaging literature. A challenging task in hyperspectral imagery, called hyperspectral unmixing [3], aims at determining the endmember signatures present in the data and estimating their abundance fractions within each pixel. In recent years, several approaches have been proposed to solve the aforementioned problem using high performance computing systems [5, 6]. Usually the bandwidth connection between the satellite/airborne platform and the ground station is reduced, which limits the amount of data that can be transmitted. As a result, there is a clear need for (either lossless or lossy) hyperspectral data compression techniques that can be applied onboard the imaging instrument.

In contrast, the application of compressive sensing (CS) to hyperspectral images is an active area of research over the past few years, both in terms of the hardware and the signal processing algorithms [7], where the acquisition of the data in an already compressed form is involved. In [8], a constrained hyperspectral coded aperture (CHYCA) method is proposed,

This work was supported by the Portuguese Science and Technology Foundation under Projects UID/EEA/50008/2013, PTDC/EEI-PRO/1470/2012, and SFRH/BPD/94160/2013.

which combines the ideas of spectral unmixing and compressive sensing. It takes advantage of two main properties of hyperspectral data, namely the high correlation existing among the spectral bands of the hyperspectral data sets and the generally low number of endmembers needed to explain the data, which largely reduces the number of measurements necessary to correctly reconstruct the original data. However, these methods are computational expensive and time consuming. Until now, there has been no effort to accelerate coded aperture algorithms for hyperspectral images using parallel techniques in the open literature.

In this paper, a parallel implementation of a previously introduced CHYCA [8] algorithm for compressive sensing on GPUs using the compute unified device architecture (CUDA) is proposed, in order to reduce the processing time. The parallel constrained hyperspectral coded aperture (P-CHYCA) implementation exploits the GPU architecture at low level, using shared memory and coalesced accesses to memory. NVidia GeForce GTX 590 and GTX TITAN platforms have been used to test the proposed implementation with simulated data. Our study reveals that the NVidia GeForce GTX TITAN GPU can provide significant compressive sensing speedup factor.

The paper is organized as follows. Section 2 describes the original CHYCA method. Section 3 describes the proposed GPU implementation. Section 4 presents an experimental evaluation of the proposed implementation in terms of both accuracy and parallel performance using simulated data on two GPU platforms. Finally, Section 5 presents as few concluding remarks and pointers to future work.

2. METHOD DESCRIPTION

The original CHYCA method for CS was developed in [8]. This approach compresses the data on the acquisition process, then the compressed signal is sent to Earth and stored in compressed form. Later the original signal can be recovered by taking advantage of the fact that the hyperspectral data can be explained using a reduced set of spectral endmembers due to the mixing phenomenon and also exploits the high spatial correlation of the fractional abundances associated to the spectral endmembers. The algorithm can be briefly summarized as follows.

Let $\mathbf{x} \in \mathbb{R}^{n_b \times n_p}$ represent, in vector format, a hyperspectral image with n_b spectral bands and $n_p := n_r \times n_c$ pixels where n_r and n_c denote, respectively, the number of rows and columns of the hyperspectral image in the spatial domain. The ordering of $\mathbf{x}$ correspond to all image pixels for each spectral band. In order to perform the compression of the original signal $\mathbf{x}$, a set of q inner products between random vectors and the image pixels is performed. Thus the size of the compressed signal will be n_b/q times smaller than the original. This operation can be represented as a matrix multiplication $\mathbf{y} = \mathbf{A}\mathbf{x}$ where $\mathbf{A}$ is a block diagonal matrix containing the matrices $\mathbf{A}_i$ corresponding to the pixel $\mathbf{x}_i$. The CHYCA compression scheme splits the original datacube into spatial subsets for two reasons: i) the number of pixels in a real application is large, so the size of matrix $\mathbf{A}$ is huge; the algorithm splits the data set into spatial windows of size $ws \times ws = m$ and uses a small subset of matrices $\mathbf{H}_i$ so that $\mathbf{A}_i \in \{\mathbf{H}_1, \mathbf{H}_2, \ldots, \mathbf{H}_m\}$; ii) in order to exploit the spatial correlation of hyperspectral images the algorithm measures the neighbouring pixels in a spatial window using different matrices $\mathbf{H}_i$.

Let us now define the linear operator $\mathbf{x} = (\mathbf{I} \otimes \mathbf{E})\mathbf{z}$, where the matrix $\mathbf{E}$ represents the basis of the subspace where the data lives [9], $\mathbf{I}$ is the identity matrix and the vector $\mathbf{z}$ contains the coefficients. In this work, the $\mathbf{E}$ matrix contains the p endmembers of the data set by columns obtained in a very fast way through the vertex component analysis (VCA) algorithm [10], thus $\mathbf{z}$ contains the fractional abundances associated to each pixel.

Let us now assume that $\mathbf{K} = \mathbf{H}(\mathbf{I} \otimes \mathbf{E})$. If matrices $\mathbf{E}$ and $\mathbf{H}$ are available, one can estimate $\mathbf{z}$ with the q-dimensional transmitted signal $\mathbf{y}$. Since the fractional abundances in hyperspectral images exhibit a high spatial correlation, we exploit this feature for estimating $\mathbf{z}$ using the following optimization problem:

$$\min_{\mathbf{z} \geq \mathbf{0}} (1/2)\|\mathbf{y} - \mathbf{K}\mathbf{z}\|^2 + \lambda \mathrm{TV}(\mathbf{z}). \tag{1}$$

In order to avoid the need to tune the λ parameter, the reconstruction error term is a constraint $\|\mathbf{y} - \mathbf{K}\mathbf{z}\|^2 \leq \delta$ instead of being part of the objective function. Consequently, a set of new variables per term of the objective function were used and the ADMM methodology [11] has been used to solve the CHYCA minimization problem:

$$\min_{\mathbf{z} \geq 0} \mathrm{TV}(\mathbf{z}) \quad \text{subject to:} \quad \|\mathbf{y} - \mathbf{K}\mathbf{z}\|^2 \leq \delta, \tag{2}$$

where δ is a scalar value linked to the noise statistics.

By a careful choice of the new variables, the problem is converted into a sequence of much simpler problems. With this in mind, $\iota_{B(\epsilon)}$ is defined as the indicator on a ball of radius ϵ, *i.e.*, $\iota_{B(\epsilon)}(\mathbf{v}) = 0$ if $\|\mathbf{v}\| \leq \varepsilon$ and $+\infty$ otherwise.

$$\begin{aligned} \min_{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4, \mathbf{v}_5} \quad & \boldsymbol{\phi}(\mathbf{D}\mathbf{z}) + \iota_{B(\delta)}(\mathbf{v}_5) + \iota_{R+}(\mathbf{v}_2) \\ \text{subject to:} \quad & \mathbf{v}_1 = \mathbf{z} \\ & \mathbf{v}_2 = \mathbf{z} \\ & (\mathbf{v}_3, \mathbf{v}_4) = \mathbf{D}\mathbf{z} \\ & \mathbf{v}_5 = \mathbf{y} - \mathbf{K}\mathbf{v}_1, \end{aligned} \tag{3}$$

where $\iota_{R+}(\mathbf{v}) = \sum_{i=1}^{pn_p} \iota_{R+}(\mathbf{v}_i)$ is the indicator function ($\mathbf{v}_i$ represents the ith element of $\mathbf{v}$ and $\iota_{R+}(\mathbf{v}_i)$ is zero if $\mathbf{v}_i$ belongs to the nonnegative orthant and $+\infty$ otherwise).

Algorithm 1 shows the pseudo-code of the CHYCA algorithm to solve the problem in (3) and how to reconstruct the data using the linear mixture model.

Algorithm 1 Pseudocode of CHYCA algorithm.

1. Initialization: set $k = 0$, choose $\mu \geq 0$, $\mathbf{E}$, $\mathbf{z}^{(\mathbf{0})}$, $\mathbf{v}_1^{(0)}, \mathbf{v}_2^{(0)}, \mathbf{v}_3^{(0)}$, $\mathbf{v}_4^{(0)}, \mathbf{v}_5^{(0)}, \mathbf{d}_1^{(0)}, \mathbf{d}_2^{(0)}, \mathbf{d}_3^{(0)}, \mathbf{d}_4^{(0)}, \mathbf{d}_5^{(0)}$

2. repeat:

3. $\mathbf{A} \leftarrow [\mathbf{D}_h^T(\mathbf{v}_3^{(k)} + \mathbf{d}_3^{(k)}) + \mathbf{D}_v^T(\mathbf{v}_4^{(k)} + \mathbf{d}_4^{(k)})]$

4. $\mathbf{B} \leftarrow \mathbf{K}^T(-\mathbf{v}_5^{(k)} + \mathbf{y} - \mathbf{d}_5^{(k)})$

5. $\mathbf{z}^{(k+1)} \leftarrow (\mathbf{D}^T\mathbf{D} + 2\mathbf{I})^{-1}(\mathbf{v}_1^{(k)} + \mathbf{d}_1^{(k)} + \mathbf{v}_2^{(k)} + \mathbf{d}_2^{(k)} + \mathbf{A})$

6. $\mathbf{v}_1^{(k+1)} \leftarrow (\mathbf{K}^T\mathbf{K} + \mathbf{I})^{-1}[(\mathbf{z}^{(k+1)} - \mathbf{d}_1^{(k)}) + \mathbf{B}]$

7. $\mathbf{v}_2^{(k+1)} \leftarrow \max\left(0, \mathbf{z}^{(k+1)} - \mathbf{d}_2^{(k)}\right)$

8. $\mathbf{v}_3^{(k+1)} \leftarrow \text{soft}\left(\mathbf{D}_h(\mathbf{z}^{(k+1)}) - \mathbf{d}_3^{(k)}, 1/\mu\right)$

9. $\mathbf{v}_4^{(k+1)} \leftarrow \text{soft}\left(\mathbf{D}_v(\mathbf{z}^{(k+1)}) - \mathbf{d}_4^{(k)}, 1/\mu\right)$

10. $\xi \leftarrow \mathbf{y} - \mathbf{K}\mathbf{v}_1^{(k+1)}$

11. $$\mathbf{v}_5^{(k+1)} \leftarrow \begin{cases} \xi - \mathbf{d}_5^{(k)} & \|\xi - \mathbf{d}_5^{(k)}\| \leq \delta \\ \frac{\delta(\xi - \mathbf{d}_5^{(k)})}{\|\xi - \mathbf{d}_5^{(k)}\|} & \text{otherwise,} \end{cases}$$

12. Update Lagrange multipliers:

$\mathbf{d}_1^{(k+1)} \leftarrow \mathbf{d}_1^{(k)} - \mathbf{z}^{(k+1)} + \mathbf{v}_1^{(k+1)}$

$\mathbf{d}_2^{(k+1)} \leftarrow \mathbf{d}_2^{(k)} - \mathbf{z}^{(k+1)} + \mathbf{v}_2^{(k+1)}$

$\mathbf{d}_3^{(k+1)} \leftarrow \mathbf{d}_3^{(k)} - \mathbf{D}_h\mathbf{z}^{(k+1)} + \mathbf{v}_3^{(k+1)}$

$\mathbf{d}_4^{(k+1)} \leftarrow \mathbf{d}_4^{(k)} - \mathbf{D}_v\mathbf{z}^{(k+1)} + \mathbf{v}_4^{(k+1)}$

$\mathbf{d}_5^{(k+1)} \leftarrow \mathbf{d}_5^{(k)} + \mathbf{v}_5^{(k+1)} - \xi$

13. Update iteration: $k \leftarrow k + 1$

14. until $k = MAX\_ITERATIONS$

15. Reconstruction $\hat{\mathbf{x}} = (\mathbf{I} \otimes \mathbf{E})\mathbf{z}^k$

3. GPU IMPLEMENTATION

CHYCA is highly parallelizable, since all calculations can be performed in a pixel-by-pixel basis. The implementation of P-CHYCA algorithm starts with an initialization step. The `cuBLAS` and `cuFFT` libraries are first initialized. After that the compressed hyperspectral image is loaded band by band from the hard disk to the main memory of the GPU. This arrangement intends to access consecutive pixels in the same wavelength in parallel by the processing kernels (coalesced accesses to memory). The next step pre-computes the fixed term $(\mathbf{K}^T\mathbf{K} + \mathbf{I})^{-1}$ in order to avoid repeated computations inside the main loop from lines 2-14. Therefore, this step is computed in the CPU using the LAPACK[1] package due to the fact that the size of these matrices is small and it is not worth to perform this computation in the GPU.

The optimization of $\mathbf{z}$ in line 5 of Algorithm 1 is carried out with a kernel which computes the right side of the equation in this line; here each thread computes one element and the grid configuration will contain $Num_b = \lceil \frac{n_p}{1024} \rceil$ blocks with the maximum number of threads supported by the architecture (1024 for the GPU considered). Later the $\mathbf{z}$ optimization is performed using the `cuFFT` library. Herein, two Fourier transform types were used: `real-to-complex` (R2C) and `complex-to-real`(C2R), finally the result is stored in global memory. Once the above operation is completed, the optimization of $\mathbf{v}_1$ in line 6 is carried out in two kernels. The first calculates the operation: $\mathbf{K}^T(-\mathbf{v}_5^{(k)} + \mathbf{y} - \mathbf{d}_5^{(k)})$, where the matrix $\mathbf{K}$ is stored in shared memory to optimize the memory access to the elements of this matrix. In first place, each thread in this kernel computes one element of the operation: $(-\mathbf{v}_5^{(k)} + \mathbf{y} - \mathbf{d}_5^{(k)})$, storing the resulting $\mathbf{VYD}$ structure in global memory. Then, each thread computes one element in the matrix multiplication between $\mathbf{K}^T$ and $\mathbf{VYD}$. Once the above operation is completed, the second kernel is executed to compute the matrix multiplication between $(\mathbf{K}^T\mathbf{K} + \mathbf{I})^{-1}$ stored in shared memory and the right side of the equation in Line 6, where the same grid configuration as the previous one is used. Finally, the optimization of $\mathbf{v}_1$ is stored in global memory.

Line 7 in Algorithm 1 is carried out with a kernel called `Optimize_v2` which computes the maximum between 0 and $\mathbf{z}^{(k+1)} - \mathbf{d}_2^{(k)}$. This kernel uses as many threads as the number of elements of the vector (n_p), with the same grid configuration as the previous kernels. The optimization of $\mathbf{v}_3$ is carried out jointly by a single kernel called `Optimize_v3_v4` which computes the lines 8 and 9 in Algorithm 1. This kernel uses the same configuration as the previous kernels with as many threads as the image pixels. In this kernel each thread computes the horizontal and vertical differences of one element and performs the `soft` function for the corresponding element, where $\text{soft}(\cdot, \tau)$ denotes the application of the soft-threshold function $\mathbf{b} \mapsto \mathbf{b}\frac{\max\{\|\mathbf{b}\|_2 - \tau, 0\}}{\max\{\|\mathbf{b}\|_2 - \tau, 0\} + \tau}$.

On the other hand, the optimization of $\mathbf{v}_5$ is divided into two kernels in Line 11. The first one, each thread computes one element of $\mathbf{y} - \mathbf{d}_5$ and subtracts the matrix multiplication $\mathbf{K}$ by its corresponding element in $\mathbf{v}_1^{(k+1)}$ leading to the $\mathbf{Y\_KV1\_D5}$ structure in global memory. Due to the fact that this kernel does not use shared memory, the memory accesses can be optimized with a larger L1 cache memory in order to increase the kernel performance. Later the second kernel performs a reduction process which uses shared memory and coalesced memory accesses to add each element in $\mathbf{Y\_KV1\_D5}$ structure. Note that in this case, the number of threads per block is reduced to 512 in order to get a better performance. Finally, the update Lagrange multipliers is computed with two kernels which respectively compute the update of the variables $\mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_5$ and $\mathbf{d}_3, \mathbf{d}_4$.

The algorithm repeats this process until a number of iterations k is reached.

4. EXPERIMENTAL RESULTS

The simulated data set used in this experiments was generated from spectral signatures randomly selected from the United States Geological Survey (USGS)[2]. The data set consists of a set of 30 signatures from the USGS library and it is generated using the procedure described in [12] to simulate natural spatial patterns, composed by a total size of 600×512 pixels (275 MB).

[1] http://www.netlib.org/lapack/

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

In order to evaluate the accuracy of the proposed implementation in terms of reconstruction error, the normalized mean squared error (NMSE) between the original image and the reconstructed image (after data compression and decompression) has been adopted as performance indicator. In this experiments, we bring the dimensionality of the original hyperspectral image from $n_b = 224$ to $q = 15$, thus achieving a compression ratio of $n_b/q = 14.93$. Note that the NMSE values obtained for the proposed parallel implementation present the same result as the serial version.

Table 1. Processing times (in seconds) and speedups achieved for the P-CHYCA in two different platforms and tested with simulated data set.

	SYNTHETIC data set		
	CPU	**GPU1**	**GPU2**
RAM $\rightarrow GlobalMem.$	–	0.0112	0.0064
$(\mathbf{K}^{\mathbf{T}}\mathbf{K} + \mu\mathbf{I})^{-1}$	0.0018	0.0019	0.0020
RAM $\rightarrow GlobalMem.$	–	≈ 0.0000	≈ 0.0000
Compute $\mathbf{z}$ (line 3)	43.5500	2.3433	1.4437
Compute $\mathbf{v}_1$ (line 4)	173.7161	4.9505	3.0786
Compute $\mathbf{v}_2$ (line 5)	2.4494	0.2315	0.1435
Compute $\mathbf{v}_3$, $\mathbf{v}_4$ (lines 6 - 7)	29.2135	0.6049	0.4927
Compute $\mathbf{v}_5$ (line 8)	93.7765	1.7767	1.1494
Compute $\mathbf{d}$ (line 9)	73.8639	1.5798	1.0084
Reconstruction (line 12)	1.6416	0.0250	0.0057
RAM $\leftarrow GlobalMem.$	–	0.0805	0.0648
Total Time	418.2128	11.6051	7.3952
Speedup (CPU time / GPU time)	–	36.04x	56.55x

The proposed P-CHYCA algorithm has been tested on two different GPUs: NVidia GeForce GTX 590[3] (GPU1) and GTX TITAN[4] (GPU2). The CPU platform is equipped with a Intel i7-2600 CPU at 3.40 GHz with 4 physical cores and 16 GB of DDR3 RAM memory. It is important to emphasize that our GPU version provide exactly the same results as the serial version of the implemented algorithm. The gcc-4.8.2 (gnu compiler default) with optimization flags `-O3` (for the single-core version) and the 3.3.4 FFTW library[5] version have been used. Hence, the only difference between the serial and parallel versions is the time they need to complete their calculations. The serial algorithms were executed in one of the available CPU cores, while the parallel version developed for the GPU GTX 590 has been executed using one of the two GPUs available in the system. For each experiment, ten runs were performed and the mean values were reported (these times were always very similar, with differences on the order of a few milliseconds). Table 1 summarizes the obtained results after processing simulated data on the considered GPU platforms during 175 iterations. As shown in Table 1, the scene could be processed with significant speedup factor using the GPU2, up to 56 times, including both the processing time and the time spent on host/device memory transfers.

[3]http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-590/specifications

[4]http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-titan/specifications

[5]http://www.fftw.org/#documentation

5. CONCLUSIONS AND FUTURE WORK

A parallel implementation of a constrained hyperspectral coded aperture algorithm for compressive sensing has been developed. The method optimized for GPU platforms has a significant speedup when compared with regards to the sequential version. The performance of the implementation has been evaluated using two different GPU architectures by NVIDIA: GeForce GTX 590 and GeForce GTX TITAN. Experimental results indicate that signficant performance could be obtained using the GTX TITAN device. Further experimentation with additional real hyperspectral scenes and high performance computing architectures (such as field programmable gate arrays or digital signal processors) are desirable in future developments due to their capacity to be used as onboard processing modules in airborne and (particularly) spaceborne Earth observation missions.

6. REFERENCES

[1] J. M. Bioucas-Dias, A. Plaza, G. Camps-Valls, P. Scheunders, N. Nasrabadi, and J. Chanussot, "Hyperspectral remote sensing data analysis and future challenges," *IEEE Geosci. and Rem. Sens. Mag.*, vol. 1, pp. 6–36, 2013.

[2] A. Plaza and C.-I Chang, *High Performance Computing in Remote Sensing*, Taylor & Francis: Boca Raton, FL, 2007.

[3] J. M. Bioucas-Dias, A. Plaza, N. Dobigeon, M. Parente, Q. Du, P. Gader, and J. Chanussot, "Hyperspectral unmixing: geometrical, statistical, and sparse regression-based approaches," *IEEE Journal of Selected Topics in Applied Earth Observ. and Rem. Sens.*, vol. 5, no. 2, pp. 354–379, 2012.

[4] Y. Altmann, N. Dobigeon, and J.-Y. Tourneret, "Unsupervised post-nonlinear unmixing of hyperspectral images using a hamiltonian monte carlo algorithm," *IEEE Trans. on Image Proc.*, vol. 23, no. 6, pp. 2663–2675, 2014.

[5] S. Bernabe, S. Sanchez, A. Plaza, S. Lopez, J. A. Benediktsson, and R. Sarmiento, "Hyperspectral unmixing on GPUs and multi-core processors: a comparison," *IEEE Journal of Selected Topics in Applied Earth Observ. and Rem. Sens.*, vol. 6, no. 3, pp. 1386–1398, 2013.

[6] S. Bernabe, S. Lopez, A. Plaza, and R. Sarmiento, "GPU Implementation of an Automatic Target Detection and Classification Algorithm for Hyperspectral Image Analysis," *IEEE Geosci. and Rem. Sens. Letters*, vol. 10, no. 2, pp. 221–225, 2013.

[7] P. Vandergheynst M. Golbabaee, S. Arberet, "Compressive source separation: Theory and methods for hyperspectral imaging," *arXiv preprint arXiv:1208.4505*, 2012.

[8] G. Martin, J. M. Bioucas-Dias, and A. Plaza, "Hyperspectral coded aperture (HYCA): A new technique for hyperspectral copressive sensing," *IEEE Trans. on Geosci. and Remo. Sens.*, vol. 53, no. 5, pp. 2819–2831, 2015.

[9] J. M. Bioucas-Dias and J. M. P. Nascimento, "Hyperspectral subspace identification," *IEEE Trans. on Geosci. and Remo. Sens.*, vol. 46, no. 8, pp. 2435–2445, 2008.

[10] J. M. P. Nascimento and J. M. Bioucas-Dias, "Vertex Component Analysis: A Fast Algorithm to Unmix Hyperspectral Data," *IEEE Trans. on Geosci. and Remo. Sens.*, vol. 43, no. 4, pp. 898–910, 2005.

[11] J. Eckstein and D. Bertsekas, "On the Douglas-Rachford splitting method and the proximal point algorithm for maximal monotone operators," *Mathematical Programming*, vol. 5, pp. 293–318, 1992.

[12] Gavin SP Miller, "The definition and rendering of terrain maps," in *ACM SIGGRAPH Comp. Graph.*. ACM, 1986, vol. 20, pp. 39–48.