Real Time Hyperspectral Imaging using High Frame Rate Video Camera and GPGPU Processing

Enagnon Aguenounon, Manon Schmidt, Foudil Dadouche, Wilfried Uhring, Sylvain Gioux

Real Time Hyperspectral Imaging using High Frame Rate Video Camera and GPGPU Processing

Enagnon Aguénounon, Manon Schmidt, Foudil Dadouche, Wilfried Uhring, Sylvain Gioux
ICube, UMR 7357, Université de Strasbourg and CNRS, 300 bd Sébastien Brant - CS 10413 - F-67412 Illkirch Cedex, France
Email: wilfried.uhring@unistra.fr

***Abstract*—This paper presents a new method to get real time hyperspectral images using time modulation of light and demodulation by means of General-Purpose computing on Graphics Processing Units (GPGPU). Three different Compute Unified Device Architecture (CUDA) implementations of real time hyperspectral images demodulation are presented. These methods are compared using a numerical simulation. The results show an execution time as low as 18 µs per wavelength and per frame for the Custom-made proposed implementation for a 512x512 pixels frame.**

Keywords-hyperspectral imaging; GPGPU; C CUDA; real time, FFT.

I. INTRODUCTION

The purpose of hyperspectral imaging is to obtain more information on the scene (objects, samples) by adding a spectral dimension. This additional information may be useful in many applications ranging from microscopy [1] to astronomy. Among these applications we can quote geosciences [2][3], medical applications [1][4], food quality and safety [5], art work [6] and many others. To address the specific constraints of those applications, from the infinitely small to the infinitely large, several methods have been developed over time. In terms of instrumentation, these methods are commonly classified in 3 main categories [7]: (1) methods based on the dispersive elements (grating, prism, grism) which deviates the light differently according to the wavelength, (2) the methods based on filter elements that allow to get one wavelength or a spectral band at a time makes use of a filter wheels or tunable filters to get all spectral information and (3) interferometer systems based methods. Alternatively, it is possible to classify the methods in 3 other categories depending of the field of view: whiskbroom, pushbroom and framing [7]. The whiskbroom category refers to a point system using one of the aforementioned instrumentation methods and that requires to scan the entire scene to get the hyperspectral data. The pushbroom category is usually composed of several whiskbrooms that each acquires a line by scanning the scene in the opposite direction. Finally, the framing category, built around a whiskbroom or a pushbroom, is able to scan an entire scene at a time. It may use filters to acquire spectral information over time or be designed by stacked detectors, sensitive to different incident radiation spectrum. Most of these techniques are nowadays integrated into systems and many manufacturers offer both sensors technology and hyperspectral cameras.

In this article, we propose a new method to get real time hyperspectral images using time modulation of light and GPGPU processing. The proposed method allows using a classic monochrome camera to acquire several wavelengths simultaneously. It is robust to noise, does not require a complex optical or mechanical system and offers at the same time a very high spectral and spatial resolution.

The details of this method are described in the following sections. First, the operating principle of the hyperspectral method is introduced in Section II. Section III presents the Graphics Processing Units (GPU) implementation of the wavelengths extraction algorithm. The testing methods are reported in Section IV and the obtained results are presented in Section V where the advantages and current limitations of the proposed method are also discussed. Section VI provides some final observations, as well as future work required to improve reliability, robustness, efficiency and practicality of this hyperspectral imaging method.

II. PROPOSED METHOD

A. Principle

The principle of the method is inspired from amplitude modulation and demodulation used in telecommunication [8]. As shown in the block diagram “Figure 1”, the typical system is composed of a camera, several modulated laser sources, a projection system, a high speed acquisition frame grabber and a GPU.

The approach is composed of two consecutive steps. The first one consists of time modulation of k laser sources with different wavelengths λ_k in a sinusoidal manner at different frequencies F_k. The chosen frequencies are exact frequencies in the discrete Fourier domain given by the formula (1): they depend on the camera framerate (fps), the number of images (N), which will be used later for the demodulation and the spectrum position (k) in Fourier domain. In a case where a modulation frequency does not match an exact discrete Fourier frequency, it will spread its energy on the two adjacent modulation frequencies, leading to possible crosstalk, as illustrated in “Figure 2”.

$$F_k = \frac{k \times fps}{N} \quad (1)$$

In the second step, the spectral contribution of each wavelength can be isolated by using discrete Fourier transform during a demodulation process on a computer. In order, to keep a high framerate at the end of the process the temporal demodulation is performed by using a GPU processing.

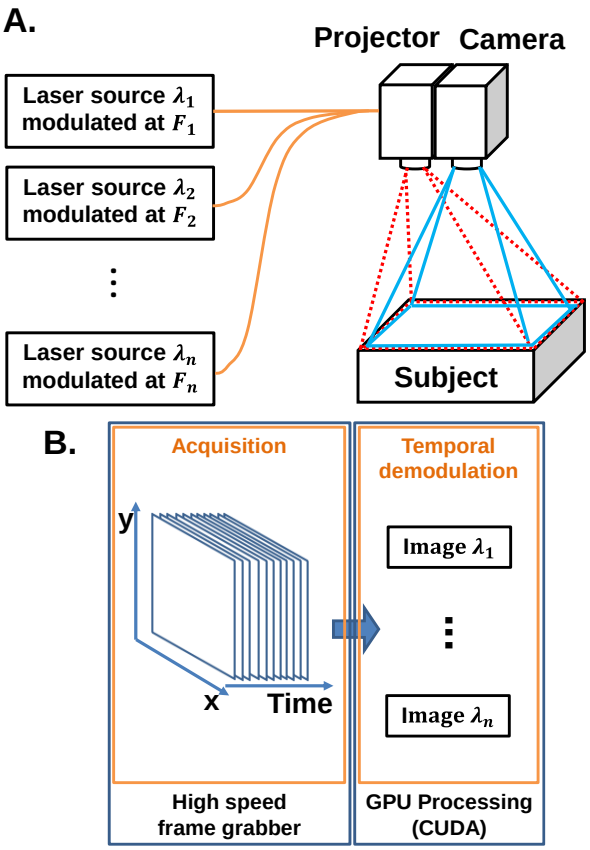

Figure 1. Real time hyperspectral acquisition and processing setup (A: Projection system and camera, B: Acquisition and processing system).

B. Spectral information extraction

In order to get the spectral contributions of each wavelength, a discrete Fourier transform is used from the N acquired images referred in (1).

The discrete Fourier transform of one pixel in time is given by:

$$X_k = \sum_{n=0}^{N-1} x_n * e^{-i2\pi\frac{nk}{N}} \tag{2}$$

With

$$e^{-i2\pi\frac{nk}{N}} = \cos\left(-2\pi\frac{nk}{N}\right) + i\sin\left(-2\pi\frac{nk}{N}\right) \tag{3}$$

Thus:

$$X_k = \sum_{n=0}^{N-1} x_n \cos\left(-2\pi\frac{nk}{N}\right) + i\sum_{n=0}^{N-1} x_n \sin\left(-2\pi\frac{nk}{N}\right) \tag{4}$$

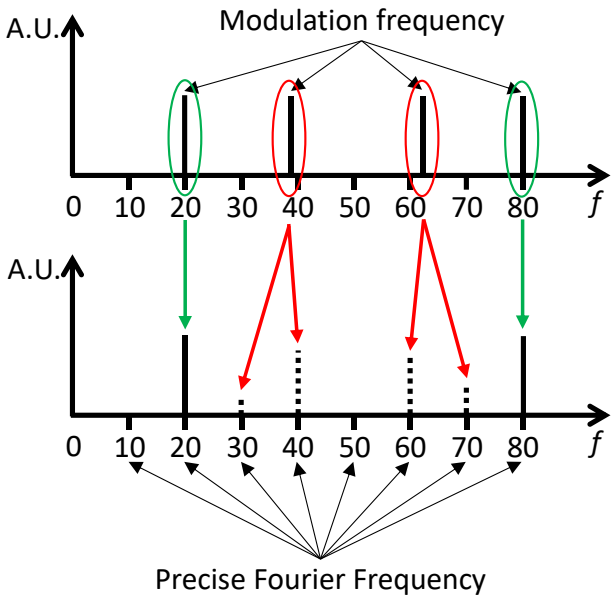

Figure 2. Sample of good/bad Frequency positioning in Fourier domain - arbitrary unit (A.U.).

And the amplitude value of (4) is the spectral contributions of the wavelength modulated at a given frequency.

$$\|X_k\| = \sqrt{\left(\sum_{n=0}^{N-1} x_n \cos\left(-2\pi\frac{nk}{N}\right)\right)^2 + \left(\sum_{n=0}^{N-1} x_n \sin\left(-2\pi\frac{nk}{N}\right)\right)^2} \tag{5}$$

Depending on the application, a windowed step or rolling window is used. According to our experience, the best way to extract a hypercube data is to use the rolling window method. Indeed, the next pixel in time is given by (6) and can be rewritten with (7) and (8):

$$X_k = \sum_{n=1}^{N-1+1} x_n \cos\left(-2\pi\frac{(n \bmod N)k}{N}\right) + i\sum_{n=1}^{N-1+1} x_n \sin\left(-2\pi\frac{(n \bmod N)k}{N}\right) \tag{6}$$

$$\sum_{n=1}^{N-1+1} x_n \cos\left(-2\pi\frac{(n \bmod N)k}{N}\right) = \sum_{n=0}^{N-1} x_n \cos\left(-2\pi\frac{(n \bmod N)k}{N}\right) - x_0 \cos\left(-2\pi\frac{(0 \bmod N)k}{N}\right) + x_N \cos\left(-2\pi\frac{(N \bmod N)k}{N}\right) \tag{7}$$

$$\sum_{n=1}^{N-1+1} x_n \sin\left(-2\pi\frac{(n \bmod N)k}{N}\right) = \sum_{n=0}^{N-1} x_n \sin\left(-2\pi\frac{(n \bmod N)k}{N}\right) - x_0 \sin\left(-2\pi\frac{(0 \bmod N)k}{N}\right) + x_N \sin\left(-2\pi\frac{(N \bmod N)k}{N}\right) \tag{8}$$

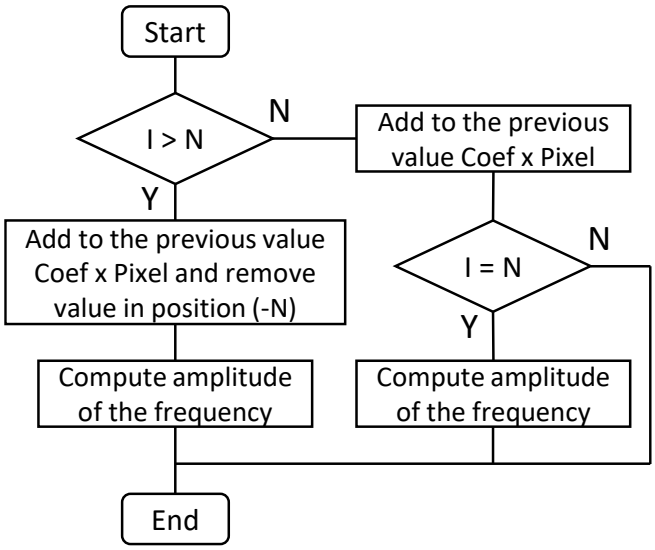

Figure 3. Demodulation algorithm applied to all the pixels of the images from the high-speed camera.

This technique, depicted in the “Figure 3”, where I is the number of acquired frames, replaces N multiplications with the subtraction of the contribution of the first sample of the rolling windows and the addition of the new sample contribution, resulting in a significant gain of resources since processors use more resources for multiplication than for addition or subtraction.

III. GPU PROCESSING

Each pixel being totally independent in time from its neighbors, the method is perfectly adapted for General Purpose GPU computing, because GPU is particularly useful for operations on arrays thanks of its massive parallel architecture.

In this section, we present three different ways to implement the temporal demodulation on the GPU and their important operations are illustrated by CUDA code snippets. Two of them consist of using CUDA Fast Fourier Transform (FFT) library of NVIDIA based on one dimension cudaFFT real to complex (R2C) function. The third one consists of a custom-made implementation inspired by the equations shown previously.

A. Implementation using cuFFT 1D

The first implementation starts with the initialization where the GPU resources are allocated for the N images of the FFT.

A ring buffer, having a dimension N corresponding to the number of images, is used for this purpose. It also requires resources for the output images of the function "cudaFFT 1D R2C" and for the amplitudes of the images, which represent the spectral contribution of each wavelength. Once the allocations have been made, it is necessary to configure the execution plan of the "cudaFFT 1D R2C" function. The role of this operation is to provide information on how the data will be read and stored and how many FFTs shall run. In our case, the function must find pixels in time in a cube of N images before executing the FFT and then reform the images. The loop processing follows the initialization. In the loop, once the buffer is full and for each new image, the function "cudaFFT 1D R2C" executes the temporal FFT. The call of the "cudaFFT 1D R2C" function is followed each time by the function that calculates the amplitudes. These amplitudes must be calculated using custom functions, because the "cudaFFT 1D R2C" function only returns complex values. “Figure 4” summarizes these different steps, starting from the creation of the plan, the transfer of the image to the GPU, the execution of the function cudaFFT and the calculation of the amplitudes by our custom function.

```
cufftHandle plan;
cufftPlanMany(&plan, 1, N, 0, (Size_X * Size_Y), 1, 0, (Size_X *
Size_Y), 1, CUFFT_R2C, (Size_X * Size_Y));
cudaMemcpy((d_image + (Size_X * Size_Y * position)), h_datain,
Size_X * Size_Y * sizeof(cufftReal), cudaMemcpyHostToDevice);
cufftExecR2C(plan, d_image, d_outdata);
amplitude_function << < grid, block >> > (d_outdata, d_ amplitude);
cudaDeviceSynchronize();
```

Figure 4. A part of cuFFT 1D (Direct) C CUDA code.

B. Improved implementation using cuFFT 1D

The idea behind the second implementation with the "cudaFFT 1D R2C" function is to reduce the time taken to perform pixel search and alignment by directly controlling it with a custom GPU function. The function created has the role of temporally aligning the pixels by respecting the ring buffer setup. This implementation requires a modification of the code for the function that calculates the amplitudes allowing the reconstitution of the images after FFT execution. The resources needed for this operation are allocated as before and the execution plan of the function "cudaFFT 1D R2C" is configured. In loop processing, as soon as an image arrives, the pixels are aligned in the buffer. As described in “Figure 3”, when the system has received at least the N number of images needed by the FFT, the "cudaFFT 1D R2C" function executes. The amplitudes are then calculated and the images reformed. “Figure 5” is given below to summarize this improved implementation.

```
cufftHandle plan;
cufftPlanMany(&plan, 1, N, 0, 1, N, 0, 1, (N / 2 + 1), CUFFT_R2C,
(Size_X * Size_Y));
cudaMemcpy(d_image, h_datain, Size_X * Size_Y * sizeof(uint16),
cudaMemcpyHostToDevice);
alignment_function << < grid, block >> > (d_image, d_image_
aligned);
cufftExecR2C(plan, d_image_ aligned, d_outdata);
reformation_and_amplitude_function << < grid, block >> >
(d_outdata, d_ amplitude);
cudaDeviceSynchronize();
```

Figure 5. A part of cuFFT 1D (Improved) C CUDA code.

C. Custom-made DFT 1D Implementation

This last implementation is inspired from the equations shown in the previous Section. As usual it needs to reserve the GPU memory spaces necessary for processing: (1) a memory space is reserved for the sin () and cos ()

coefficients needed for the calculation of the Discrete Fourier Transform (DFT), (2) a ring buffer with a dimension of N+1 is setup for the camera output images, (3) a memory space is reserved for the images after treatment, and finally (4) another memory space is allocated to store intermediate results of real and imaginary parts. After accomplishing memory allocation, the coefficients for the DFT are then calculated on the CPU and they are transferred to the GPU.

Once the initialization is finished, the processing loop can begin “Figure 6”. In this step, each time an image arrives it is stored in the ring buffer of size N+1. The function of the “Figure 7” is then called to compute the DFT as explained above. This function directly returns an image for each demodulated wavelength.

```
cudaMemcpy((d_image + (Size_X * Size_Y * position)), h_datain,
Size_X * Size_Y * sizeof(uint16), cudaMemcpyHostToDevice);
DFTcomputationGPU << < grid, block >> > (d_image, d_amplitude,
Reel, Imag, Cos_coef, Sin_coef);
cudaDeviceSynchronize();
```

Figure 6. A part of DFT 1D custom implementation C CUDA code.

```
__global__ void DFTcomputationGPU(uint16 *d_image, float
*d_amplitude, …) {
unsigned int id = (threadIdx.y + (blockIdx.y * blockDim.y)) *
Size_X[0] + (threadIdx.x + (blockIdx.x * blockDim.x));
…
for (int i = 0; i < N_lambda[0]; i++) {
…
if (Num_image[0] < N[0]) {
Reel[idx] = Reel[idx] + (d_image (id + Present_idx[0]) *
Cos_coef[ind]);
Imag[idx] = Imag[idx] + (d_image (id + Present_idx[0]) *
Sin_coef[ind]);
if (Num_image[0] == (N[0] - 1)) {
d_amplitude[idx] = (sqrtf((Reel[idx]^2) + (Imag[idx]^2)));
}}
if (Num_image[0] >= N[0]) {
Reel[idx] = Reel[idx] - (d_image (id + Past_idx[0]) * Cos_coef[ind])
+ (d_image (id + Present_idx[0]) * Cos_coef[ind]);
Imag[idx] = Imag[idx] - (d_image (id + Past_idx[0]) * Sin_coef[ind])
+ (d_image (id + Present_idx[0]) * Sin_coef[ind]);
d_amplitude[idx] = (sqrtf((Reel[idx]^2 ) + (Imag[idx]^2)));
}}}
```

Figure 7. Custom-made DFT 1D computation function.

IV. NUMERICAL SIMULATION

A. *Measurement simulation*

In order to evaluate the performance of these three CUDA codes, we measured the execution time considering a one dimension DFT on stack of 8, 16, 32, 64, 128 and 256 images of size 512 x 512 pixels. These images allow extracting respectively 5, 9, 17, 33, 65 and 129 wavelengths by demodulation. The numbers of images have been chosen so that functions based on the cudaFFT library produce optimal performance [12]. The images used in this experience are created using Matlab (MathWorks) calculation tools by addition of multiple sinus signal for one pixel and then simply duplicating M times to obtain the desired image size.

B. *GPU configuration*

In GPGPU computing, the processing time depends strongly on the configuration. For our implementation, the major configuration we chose is related to the number of threads and their organization for the execution of the written functions. In our study, since we have been working on 512 x 512 pixel size images, and the pixels are independent in time, we chose to create one thread for each pixel, which implies 512 x 512 = 262144 threads that we broke down into 4096 blocks of 64 threads.

V. RESULTS AND DISCUSSION

The measurements of the different implementations depending of the number of images used for demodulation were performed. These measurements were performed with two configurations; one with the GPU Quadro K2200 which has 640 CUDA cores and another with the Geforce GTX 1080 Ti with 3584 CUDA cores. The obtained results are shown in “Figure 8” and “Figure 9”. As we can see, the implementation "Custom DFT 1D algorithm" is the fastest in execution time followed by the two implementations "cuFFT 1D (improved)" and "cuFFT 1D (direct)". The relatively long execution time of the two last implementations (compared to the Custom implementation) is explained by the many matrix transpositions they performed. Indeed, the performance of matrix transpositions is strongly limited by the bandwidth and the memory accesses [9]. For the fastest implementation, the Quadro K2200 took 11400 µs for the demodulation of 129 wavelengths, corresponding to 88 µs per wavelength, and the Geforce GTX 1080 Ti took 2250 µs for the same number of frames, or 18 µs per wavelength.

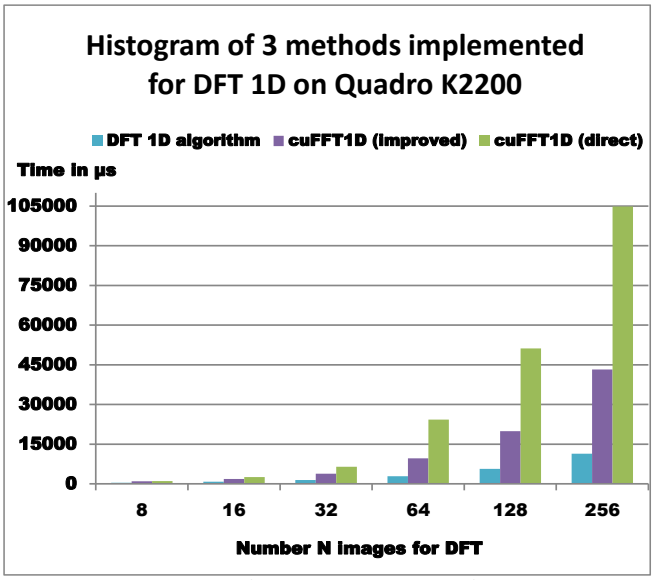

Figure 8. Histogram of 3 methods implemented for DFT 1D on Quadro K2200.

The first conclusion drawn from these results is that the execution time is inversely related with the number of

CUDA cores: here multiplying the number of CUDA cores by 5 results in dividing the execution time by 5. This leads to suppose that we can still reduce the execution time and therefore increase the acquisition frame rate with additional GPU cores. Another important point is that the custom-made implementation has the particularity to allow the user to just compute the chosen modulation frequency.

Nevertheless, this method also has some limitations. A first limitation is the constant delay due to the first number of images needed to execute the first demodulation. Depending on the application, such a delay may be or not a drawback. This delay also implies no motion of the sample during the acquisition, or at least motion that is much slower that the time needed to acquire N frames. A way to mitigate this issue consists of reducing the number of wavelengths wanted by taking fewer frames for the FFT. A second limitation is the number of wavelengths that can be acquired, a direct consequence of the dynamic range of the camera. Indeed, the bit depth of a camera is generally 8, 10, 12 or 16 bits. Consequently, the higher the number of modulated wavelengths, the more the dynamic will be subdivided. And with low modulation amplitude the demodulated image quality will decrease due to a low signal to noise ratio. One solution to circumvent this limitation would be to use one or more camera with tunable or wheel band filter at the cost of a more complex and expensive system.

VI. Conclusion

A new method to acquire hyperspectral images was presented by combining temporal modulation of light and temporal demodulation using GPGPU processing. Three CUDA codes allowing to get spectral information are presented and are compared. With a simulation test we demonstrate that it is possible to reach very high acquisition frame rate by calculating the DFT with accumulation and subtraction at the arrival of each image. We also discussed about hardware system limitations essentially due to the cameras dynamics. As a future work, it will be interesting to study with a real system the effect of increasing the number of wavelengths on the quality of demodulated images.

Acknowledgment

Funding for this research was provided by European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program under grant agreement No 715737 (QuantSURG), France Life Imaging, University of Strasbourg IdEx, and ICube Laboratory.

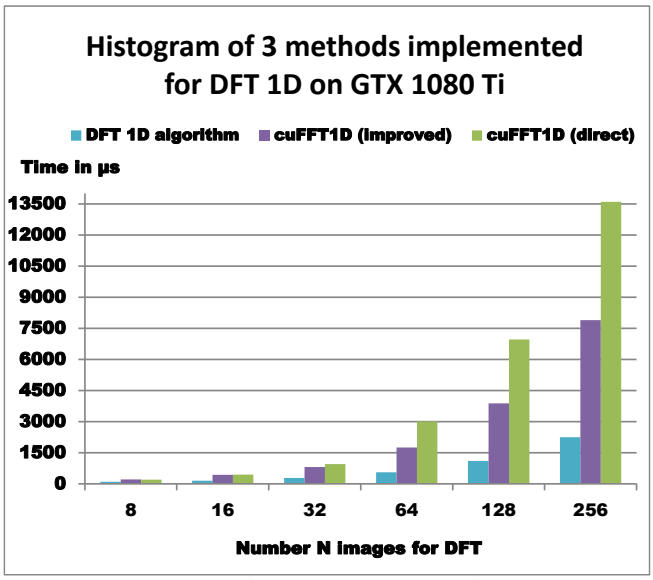

Figure 9. Histogram of 3 methods implemented for DFT 1D on GTX 1080 Ti.

References

[1] L. Gao and R. T. Smith, “Optical hyperspectral imaging in microscopy and spectroscopy – a review of data acquisition”, J Biophotonics, vol. 8(6), June 2015, pp. 441-456, doi:10.1002/jbio.201400051.

[2] T. Adão et al, “Hyperspectral Imaging: A Review on UAV-Based Sensors, Data Processing and Applications for Agriculture and Forestry”, Remote Sensing, vol. 9(11), Oct. 2017, pp. 1-30, doi:10.3390/rs9111110.

[3] L. Rousset-Rouviere et al, “SYSIPHE system: a state of the art airborne hyperspectral imaging system. Initial results from the first airborne campaign”, Proc. SPIE, vol. 9249, Oct. 2014, doi:10.1117/12.2066643.

[4] G. Lu and B. Fei, “Medical hyperspectral imaging: a review”, J Biomed Opt, vol. 19(1), Jan. 2014, pp. 1-23, doi:10.1117/1.JBO.19.1.010901.

[5] Y. Liu, H. Pu, D.-W. Sun, “Hyperspectral imaging technique for evaluating food quality and safety during various processes: A review of recent applications”, Trends in Food Science & Technology, vol. 69, part A, Nov. 2017, pp. 25-35, doi.org/10.1016/j.tifs.2017.08.013.

[6] A. Polak et al, “Hyperspectral imaging combined with data classification techniques as an aid for artwork authentication”, Journal of Cultural Heritage, vol. 26, July-August 2017, pp. 1-11, doi.org/10.1016/j.culher.2017.01.013.

[7] R. G. Sellar and G. D. Boreman, "Classification of imaging spectrometers for remote sensing applications”, Optical Engineering, vol. 44(1), Jan. 2005, pp. 1-3, doi:10.1117/1.1813441.

[8] A. B. Carlson and P. B. Crilly, Communication Systems: An introduction to signals and noise in electrical communication, McGraw-Hill Education, 2009.

[9] J. Cheng, M. Grossman, T. McKercher, Professional CUDA C Programming, John Wiley & Sons, 2014.

[10] J. Sanders and E. Kandrot, CUDA by Example: An introduction to general-purpose GPU programming, Addison-Wesley Professional, 2010.

[11] Nvidia, docs.nvidia.com, “CUDA C Programming Guide”, [Online] Available from: http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf, [retrieved: March, 2018].

[12] Nvidia, docs.nvidia.com, “CUFFT Library”, [Online] Available from: http://docs.nvidia.com/cuda/pdf/CUFFT_Library.pdf, [retrieved: March, 2018].