SIMULATOR OF HYPERSPECTRAL IMAGE CAPTURE AND GENERATION

Efraín Alberto Padilla Zepeda[1,2], Héctor Manuel Gómez Gutierrez[1], Deni Librado Torres Román[2], Diego Armando Castellanos Beard[1], Jesús Simeone Ambrocio Hernández[1], Rodrigo Rivera Morán[1], Axel Rafael López Orozco[1], Ángel Abraham Inda Cázares[1], Ángel Quirino Jiménez Sánchez[1]

Universidad de Artes Digitales (UAD)[1]
Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional (Cinvestav)[2]
Efrain.Padilla@cinvestav.mx

ABSTRACT

Although hyperspectral images have advantages, compared to other earth observation products (e.g., RGB or multispectral images), in terms of the spectral characterization of the materials that compose them, they face other problems, such as the high cost of acquisition and storage. In addition, the low availability of this data is due to the fact that only some space agencies and the private sector have the financial resources or licenses to use it, for security reasons. If labels are required from direct observation on the earth's surface, the options are reduced to a few datasets available for research and applications development. This work provides to the scientific and developer community, fully labeled hyperspectral images in three classes: soil, water, and vegetation. The products of this simulator could help to the development of post-processing tasks, for example; classification, noise reduction, and atmospheric correction. The proposed method is to procedurally generate a matrix of labels of those three classes, each of the labels will describe a pixel of a virtual world, subsequently rendered with high-definition textures and captured in Unreal Engine. Using the label matrix, the hyperspectral signature is generated with a neural network model based on a multi-layer perceptron, implemented in Python, previously trained with real hyperspectral data from the "University of Houston" dataset of 144 spectral bands, captured using the Compact Airborne Spectrographic Imager (CASI) sensor in 2012. In addition, thermal and photonic noise are added to the final product, with the proportion and signal-to-noise ratio (SNR) defined by the user. The prediction results of the neural network show the characteristic spectral signatures of the three classes used, adding variations depending on the reflectance of the pixel in the simulator.

Keywords: Hyperspectral, data generator, procedural generation.

INTRODUCTION

This work presents to the scientific community a generator of completely labeled hyperspectral images. This framework can be used for research and development of post-processing algorithms.

Hyperspectral imaging provides hundreds of frequency bands in the visible infrared and ultraviolet wavelengths of the electromagnetic spectrum, in order to obtain spectral signatures. The main problem is the scarcity of Hyperspectral Images (HSI) since most of them are held by the private sector or space agencies. Therefore, they are not accessible to the public. These products can be requested, but require administrative processes, as they may contain sensitive information.

To obtain an HSI it is necessary to have specialized hardware which is prohibitively expensive for most researchers. Another option is to work with popular public datasets, but most of them are a relatively small single image, in addition to the fact that not all the pixels are labeled.

Hence, this work proposes to solve this problem generating labeled hyperspectral data from a completely generated map that does not belong to any location in the world, making use of common tools and techniques used for procedural map generation for video games.

The use of tools such as Unreal Engine (UE) works in our favor, since they are optimized for the use of the Graphics Processor Unit (GPU) to make the render of the map. In a matter of a few seconds the map and Ground Truth (GT) is ready to subsequently generate the hyperspectral data.

There are different methods to generate spectral signatures, summarized in Table 1. This categorization is presented in (Han and Kerekes, 2017).

Table 1. Categories of spectral signature generation techniques.

Category	Description
Empirical approach	Used in the development of satellite missions, based on the architecture of spectral sensors (Guerin et.al., 2011; Hook et.al. 2001).
Image Modification	Insert artificial pixels into real images to generate hyperspectral data. Usually, extracting endmembers to generate another HSI (Agarwal et.al., 2004; Eches et.al., 2010).
Statistical Approach	Create synthetic imagery through the use of statistical or probabilistic models, or more recently, machine learning algorithms (North, 1996; Rivera et.al., 2015).
Physical Modeling	Simulate light interacting with 3D models of a simulated land. Incorporate Ray Tracing models and account of true radiometry and propagation of light from its source through the atmosphere off of the surface, and into the sensor aperture (Govaerts & Verstraete, 1998; Jensen, 2001; Gastellu-Etchegorry et.al., 2004, 2015; Ontiveros et.al., 2011; Jakubowski et.al., 2007; Scanlan et.al., 2004).

Based on Table 1, our work belongs to the statistical approach category, since the spectral signature is generated based on the prediction of a neural network trained with real hyperspectral data. This work aims to test the concept of spectral signature generation based on a procedurally generated GT and its reflectance, simulated on a video game engine.

CONTENT

This section presents a big picture of the proposed framework which is divided into two major blocks. The first one is an implementation in C++, which corresponds to Ground Truth (GT) generation and the rendering for image capture. The second block refers to the Python implementation, where an Artificial Neural Network (ANN) is trained with real hyperspectral data. The products of this framework are the generated hyperspectral data with its corresponding GT labels. Also a high resolution RGB image with a quick visualization through a user interface.

1. Procedural Map Generation: For Ground Truth (GT) generation, we have implemented an algorithm based on Perlin noise for our application, described in general terms in (Perlin, 1985). Considering a third-order tensor $\boldsymbol{\mathcal{G}} \in \mathbb{R}^{X \times Y \times 2}$, representing a grid, with X horizontal and Y vertical lines. For each intersection, a gradient vector is generated and assigned at each fiber $\boldsymbol{g}_{x,y,:}$. The gradients are generated randomly (there are not pre-computed components) by generating a random angle θ in radians, where $0 \leq \theta \leq 2\pi$, and then, it is computed a unit vector through the cosine and sine functions of θ, such that, $g_{x,y,0} = \cos(\theta)$ and $g_{x,y,1} = \sin(\theta)$. For each grid cell (also called quadrants), it is defined a set of candidate points $p = (x_i, y_j)$. For each intersection, correspondent to the grid cell (there are four intersections per cell), it is computed an offset vector $\boldsymbol{o}_{x_i y_i}$, which is a displacement from the intersection to the candidate point. Then, it is computed the dot product between the gradient vectors $\boldsymbol{g}_{x,y,:}$ associated with the grid cell, and the offset vector $\boldsymbol{o}_{x_i y_i}$. Once the four dot products are computed, a 2D interpolation is performed, which is based on a single interpolation on both axes. Then, these results are interpolated again to get the value for the altitude map, corresponding to the candidate point selected. The Perlin noise generation shown in Figure 1. The algorithm is described in Table 2.

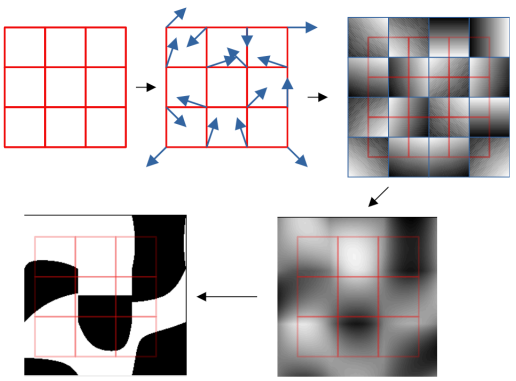

Figure 1. Steps of Perlin Noise Algorithm.

Table 2. Perlin Noise Generation algorithm

Perlin Noise Generation Algorithm

$Interpolate(a,\ b,\ t)$:
  return $w(a + b)\ + a$

$DotGridGradient(\boldsymbol{v}_i,\ \boldsymbol{v}_f, \boldsymbol{v})$:
  $\boldsymbol{a} \leftarrow randomVector(\boldsymbol{v}_i + \boldsymbol{v})$
  $\boldsymbol{b} \leftarrow \boldsymbol{v}_f - \boldsymbol{v}$
  return $dot(a, b)$

$Perlin - Noise(\boldsymbol{v})$:
  $\boldsymbol{v}_i \leftarrow floor(\boldsymbol{v})$
  $\boldsymbol{v}_f \leftarrow \boldsymbol{v}\ - \boldsymbol{v}_i$
  $\boldsymbol{a} \leftarrow DotGridGradient(\boldsymbol{v}_i,\ \boldsymbol{v}_f, [0, 0])$
  $\boldsymbol{b} \leftarrow DotGridGradient(\boldsymbol{v}_i,\ \boldsymbol{v}_f, [1, 0])$
  $\boldsymbol{c} \leftarrow DotGridGradient(\boldsymbol{v}_i,\ \boldsymbol{v}_f, [0, 1])$
  $\boldsymbol{d} \leftarrow DotGridGradient(\boldsymbol{v}_i,\ \boldsymbol{v}_f, [1, 1])$
  $\boldsymbol{e} \leftarrow Interpolate(\boldsymbol{a},\ \boldsymbol{b},\ \boldsymbol{v}_f.\text{getX}())$
  $\boldsymbol{f} \leftarrow Interpolate(\boldsymbol{c},\ \boldsymbol{d},\ \boldsymbol{v}_f.\text{getX}())$
  return $Interpolate(\boldsymbol{e}, \boldsymbol{f}, \boldsymbol{v}_f.getY())$

The result of the 2D interpolation is assigned to each element $a_{i,j}$ in the altitude map, such that $-1 \leq a_{x,y} \leq 1$, represented in a matrix $\boldsymbol{A} \in \mathbb{R}^{X \times Y}$.

To achieve a natural distribution of the GT, more layers are added to the amplitude map, these layers are called octaves. An octave is defined as a twice resolution version of the original grid. The amplitude represents the contribution of each octave to the final elevation map and is reduced by a factor of $\frac{3}{4}$ each octave, described in equation (1):

$$\boldsymbol{A}_f = \sum_{1}^{n} \left(\frac{3}{4}\boldsymbol{A}_n\right)^{n-1} \quad (1)$$

The combination of multiple octaves causes a natural appearance in the elevation map. For this project, we are using three octaves. If less octaves are added, it causes an artificial appearance with more straight lines. If more octaves are added it would have too much noise to make a meaningful image, compared to remote sensing images of the earth surface.

Subsequently, a label matrix $\boldsymbol{L} \in \mathbb{N}^{I \times J}$ is computed using the altitude map. Depending on the altitude at each coordinate, a corresponding label is assigned. For this work we use three different labels, corresponding to water, soil and vegetation. This matrix is used for two purposes; to generate a pre-texture with a color representing each label for the map generation and rendering, and a .csv file, which contains the GT corresponding to each pixel of the image, this information is used to predict the hyperspectral data.

2. Unreal Engine Implementation: Unreal Engine (UE) is a graphic engine of creation tools for game development, simulation, and other real time applications. Has a blueprint visual scripting system with the concept of node-based interface to create gameplay elements within Unreal Editor. We will use this technology to generate the ground truth and terrain rendering. Although this software is widely used in the video games development area, we will use it for research purposes.

2.1. User Interface: The User Interface (UI) is handled in a simple way where researchers set the parameters of the simulator framework. Some parameters to define by the user are: Image Resolution ($X \times Y$ pixels), Renderization Quality and RGB Image Factor Scale (scale for the RGB image resolution). The UI sends these parameters to UE for terrain render, also to the Python script for spectral data generation. The user is able to visualize in a hud, the settings used for virtual world generation, as well a quick look of the generated virtual world

2.2. Terrain Generation: This section refers to the static mesh terrain generation, to be rendered in world space, meaning all the space in the scene where an object can be placed. For this, it is needed the sizes, s_x and s_y (in terms of the virtual world), and the quality $(I \times J)$ required by the user, previously obtained at the start of the simulation. Using that data, it generates the mesh, with a quite simple process. The main algorithm generates, vertex by vertex, a grid $T \in \mathbb{R}^{I \times J}$, filled with data structures called vertices (v_{ij}). Each one contains a 3D vector for its position in world space; a 2D vector that describes its texture coordinate (UV), that represents a point on an image of a 2D texture, defining the percentage (horizontally and vertically) of how far it is from the bottom left corner of the image, used by the Graphics Processor Unit (GPU) to know how to color each part of the final mesh. Another 3D vector that is called the normal direction, which

is just a unit vector pointing in the direction the vertex is facing, for light calculations and other techniques that are not used on this project, because we are rendering a flat surface.

The data for each part of the vertex is filled in that order. First, the positions p_{ij} are determined by $p_{ij} = [s_x \cdot \frac{i}{I}, s_y \cdot \frac{j}{J}, 0]$. Then, the UV coordinates $t_{ij} = [\frac{i}{I}, \frac{j}{J}]$. Finally, for each vertex, a normal vector $n_{ij} = [0, 1, 1]$ is assigned. This is a vector pointing up, for the UE coordinate system, and that is because the triangles in the constructed mesh are facing up, in the direction of the camera.

Then, the index array is generated, which is an array of integers. These indices tell the GPU how to connect each vertex to form triangles, because this is the simplest shape that can be used to construct any type of 3D meshes, and it is a standard for computer graphics (Varcholik, 2014). These indices are generated using a function named Create Grid Mesh Triangles provided by the engine, and with the vertices and indices generated, this data is sent to another function called Create Mesh Section also given by the engine, that combines all the data to construct the final mesh.

2.3. Texture Generation: The terrain needs a material to be rendered with, simulating a real earth surface looking terrain view, and to be displayed to the user interpretation. For this, are required the ground truth labels, generated in a previous section, received in the form of a Red-Green-Blue (RGB) texture, each color meaning a different label, in case of needing more labels, they can be made by combinations of these three. This texture is processed, applying a blur to it, so that the edges do not look so defined, but rather blend together, and it is made by moving in the four directions the image, by a tiny amount, then adding and subtracting colors so that, at last, it ends up with the original image, but with the color sections blended with each other. This process is made several times, with different offset values, to create a nice gradient between the colors. Thus, with the blurred image, the different colors, meaning the three RGB colors, or combination of them, of the texture, are separated and transformed into its final textures, as shown in Figure 4. In this case, each different texture represents a different terrain surface: water, soil and vegetation. Then, they are added back together to form the final texture and now it can be applied to the terrain's static mesh.

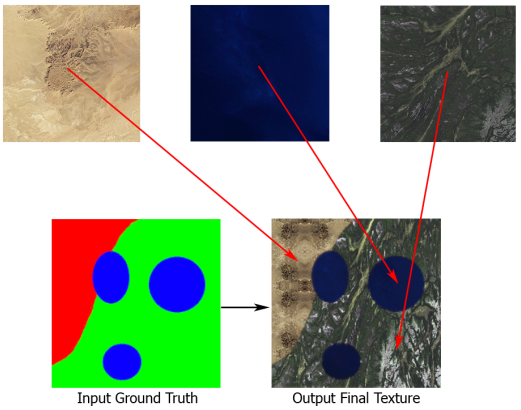

Figure 2. Ground truth to final texture transformation.

2.4. Image Capture: Once the material is applied to the terrain it needs to be transformed back to a texture in order to capture his current state. First a Render Target (RT) is created and set in the graphic pipeline, subsequently the terrain's material is drawn in a quad, and that quad is rendered in the newly created RT. Finally it is exported as a HDR image using the blueprint function that UE provides. With this approach we get the texture perfectly aligned with the GT, avoiding camera perspective problems.

3. Hyperspectral Data Generation: ANNs use machine learning algorithms to fit a mathematical model to perform predictions. The predictions are based on patterns present in the training data. In this project, we use MLP instead of a state of the art ANN, because we wanted to implement a simple prototype to test the concept. MLP allows us to train and predict the hyperspectral data faster, given its low complexity compared to state of the art models.

The model is trained with real hyperspectral data captured by the Compact Airborne Spectrographic Imager (CASI)(Babey, 1989). Captured in 2012 over the University of Houston, the hyperspectral data consist of 144 bands (380-1050nm) with 2.5m of spatial resolution. We have used the labeled pixels of the following three classes: "Grass Healthy", "Water" and 'Soil'. This dataset was made available by the Image Analysis and Data Fusion Technical Committee of IEEE GRSS in 2012.

3.1. Multi-layer perceptron architecture: The architecture used for the spectral signature generation is not based on any particular one. The number of layers, neurons, normalization

and activation functions were selected empirically, based on intensive testing. The architecture is presented in Figure 5.

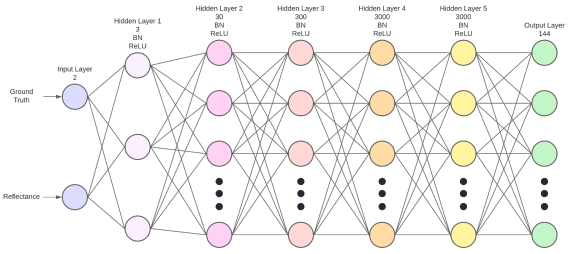

Figure 3. Big picture of the Multi-Layer Perceptron used for the spectral signature generation.

3.2. Training: Our Neural Network has two input neurons, the inputs are the ground truth and the reflectance values obtained from a gray-scale version of a RGB image from real hyperspectral data. Adam optimization algorithm is used for training, with Mean Squared Error loss function, comparing the regression with the corresponding real spectral signature.

3.3. Prediction: For the prediction, the input is feeded with the reflectance value generated in UE in gray-scale and the ground truth previously generated procedurally. With this, we introduce correlation between the virtual world and the predicted spectral signature.

3.4 Results:

Figure 4 shows the RGB version of one realization, product of the renderization of the virtual world.

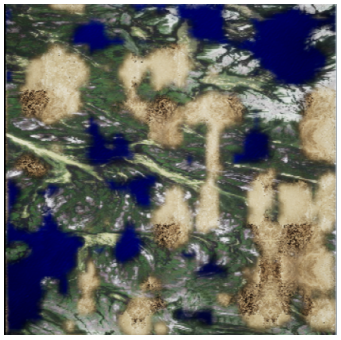

Figure 4. RGB renderization of the virtual world.

In Figure 5 is shown a spectral signature comparison between the ones of the University of Houston dataset and the ones generated by the trained MLP on average. The predicted spectral signatures show the characteristic shape of each class, if the same reflectance is used (as the original dataset), we get an almost perfect match. If the reflectance value is taken from the simulator, the spectral signature on average is different, but it preserves the characteristic shape of the class.

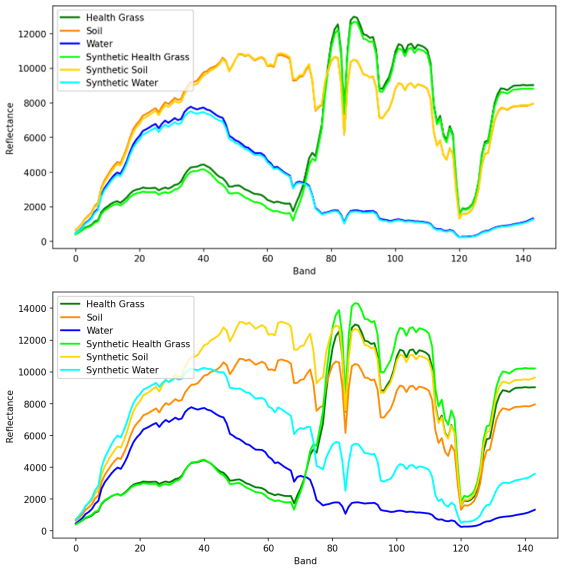

Figure 5. Comparison (on average) between Houston dataset spectral signatures and the spectral signatures generated by the trained model. Superior graphic: same reflectance as the original. Inferior graphic: reflectance from simulator.

3.5 Open issues: The proposed framework was developed in limited time as an undergraduate academic project. In consequence, we have implemented the simplest solution to test the concept on every step. There is room for improvement for the spectral signature generation and procedural generation, using state of the art models and algorithms. Moreover, it is possible to add more classes (e.g. snow, different classes of vegetation and soil, etc.) and fixed-pattern objects, such as urban structures and clouds with his respective projected shadows over the earth surface.

CONCLUSIONS

We achieved a hyperspectral image generator composed of two parts: the virtual world simulation implemented in Unreal Engine (v4.26) and the hyperspectral data generator itself implemented in Python (v3.7). The spectral signature is predicted through a Multi-Layer Perceptron, trained with real spectral data from the University of Houston dataset. With the tools that UE graphic engine provides we managed to generate hyperspectral images of any size, all from the simulation's camera pixels. Furthermore, the simulation parameters can be easily defined through a user interface. Even though this is a project addressed to a specific group of scientists, it can be used as an educational tool.

BIBLIOGRAPHY

Agarwal, S., Edara, T. K., Swonger, C. W., & Trang, A. H. (2004, September). Image-based synthesis of airborne minefield MWIR data. In *Detection and Remediation Technologies for Mines and Minelike Targets IX* (Vol. 5415, pp. 1140-1150). SPIE.

Altmann, Y., McLaughlin, S., & Hero, A. (2015). Robust linear spectral unmixing using anomaly detection. *IEEE Transactions on Computational Imaging*, *1*(2), 74-85.

Babey, S. K., & Anger, C. D. (1989). A compact airborne spectrographic imager (CASI). *Quantitative Remote Sensing: An Economic Tool for the Nineties, Volume 1*, *2*, 1028-1031.

Eches, O., Dobigeon, N., & Tourneret, J. Y. (2010). Estimating the number of endmembers in hyperspectral images using the normal compositional model and a hierarchical Bayesian algorithm. *IEEE Journal of Selected Topics in Signal Processing*, *4*(3), 582-591.

Gastellu-Etchegorry, J. P., Martin, E., & Gascon, F. (2004). DART: a 3D model for simulating satellite images and studying surface radiation budget. *International journal of remote sensing*, *25*(1), 73-96.

Gastellu-Etchegorry, J. P., Yin, T., Lauret, N., Cajgfinger, T., Gregoire, T., Grau, E., ... & Ristorcelli, T. (2015). Discrete anisotropic radiative transfer (DART 5) for modeling airborne and satellite spectroradiometer and LIDAR acquisitions of natural and urban landscapes. *Remote Sensing*, *7*(2), 1667-1701.

Govaerts, Y. M., & Verstraete, M. M. (1998). Raytran: A Monte Carlo ray-tracing model to compute light scattering in three-dimensional heterogeneous media. *IEEE Transactions on geoscience and remote sensing*, *36*(2), 493-505.

Guerin, D. C., Fisher, J., & Graham, E. R. (2011, May). The enhanced MODIS airborne simulator hyperspectral imager. In *Algorithms and Technologies for Multispectral, Hyperspectral, and Ultraspectral Imagery XVII* (Vol. 8048, pp. 214-224). SPIE.

Han, S., & Kerekes, J. P. (2017). Overview of passive optical multispectral and hyperspectral image simulation techniques. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, *10*(11), 4794-4804.

Hook, S. J., Myers, J. J., Thome, K. J., Fitzgerald, M., & Kahle, A. B. (2001). The MODIS/ASTER airborne simulator (MASTER)—A new instrument for earth science studies. *Remote Sensing of Environment*, *76*(1), 93-102.

Jakubowski, M. K., Pogorzala, D., Hattenberger, T. J., Brown, S. D., & Schott, J. R. (2007, September). Synthetic data generation of high-resolution hyperspectral data using DIRSIG. In *Imaging Spectrometry XII* (Vol. 6661, pp. 153-163). SPIE.

Jensen, H. W. (2001). *Realistic image synthesis using photon mapping* (Vol. 364). Natick: Ak Peters.

North, P. R. (1996). Three-dimensional forest light interaction model using a Monte Carlo method. *IEEE Transactions on geoscience and remote sensing*, *34*(4), 946-956.

Ontiveros, E., Gartely, M., Brown, S., Raqueño, R., & Pogorzala, D. (2011, May). Realism, utility, and evolution of remotely sensed simulations. In *Algorithms and Technologies for Multispectral, Hyperspectral, and Ultraspectral Imagery XVII* (Vol. 8048, pp. 584-592). SPIE.

Perlin, K. (1985). An image synthesizer. *ACM Siggraph Computer Graphics*, *19*(3), 287-296.

Rivera, J. P., Verrelst, J., Gómez-Dans, J., Muñoz-Marí, J., Moreno, J., & Camps-Valls, G. (2015). An emulator toolbox to approximate radiative transfer models with statistical learning. *Remote Sensing*, *7*(7), 9347-9370.

Scanlan, N. W., Schott, J. R., & Brown, S. D. (2004, January). Performance analysis of improved methodology for incorporation of spatial/spectral variability in synthetic hyperspectral imagery. In *Imaging Spectrometry IX* (Vol. 5159, pp. 319-330). SPIE.

Varcholik, P. (2014). Real-time 3D rendering with DirectX and HLSL: A practical guide to graphics programming. Addison-Wesley Professional.