CPSC 679 - Final Project

Gerardo Carranza

May 11, 2016

Abstract

The goal of this project was to create 3D printable models of the topography of the Earth and Mars. We developed a Java Program *createSTLFile.java* which received GeoTIFF files as input and output a completed STL File. We started by getting GeoTIFF (image) files, extracting the necessary attributes, and using these attributes to create a collection of 3D points (Digital Elevation Model or DEM). We then converted the DEM into binary STL Files. Using the 3D points as vertices, we programmatically derived the triangular faces of the model. Using the NASA Shuttle Radar Topographic Mission and Mars Orbital Data Explorer databases, we converted GeoTIFF files into corresponding STL Files.

1 Introduction

The purpose of this project was to develop an automated method to convert topography data from a GeoTIFF file into a 3D printable model in the form of an STL File. This was completed by creating a Java Program coded in the Eclipse IDE Mars.1 Release 4.5.1 (32 bit) with three *.java* files, *getEarthDEM.java*, *STLWriter.java*, and *createSTLFile.java*. *createSTLFile.java* used the results from getEarthDEM(TIFF_Filename), in the form of a digital elevation model, to create an STL File. *STLWriter.java* defined the format in which to write the binary STL File [1]. To write *getEarthDEM.java*, it was necessary to understand the GeoTIFF file format. It should be noted that GPS coordinates in this report are written as (longitude,latitude).

2 GeoTIFF File Format

The topography data that was extrapolated to create a 3D model was stored in a GeoTIFF file. A GeoTIFF is a type of TIFF file. Tagged Image File Format, or TIFF, is a file format for storing raster graphics images. TIFF files are defined as a sequence of 8-bit bytes; the largest possible size for a TIFF file is 2*32 bytes in length. The main feature of TIFF's is that they include header tags that define an image's geometry. A TIFF File begins with an 8-byte image file header that points to an image file directory, or IFD, which includes the defined tags for that TIFF. For more information about TIFF Files, see the TIFF (Revision 6.0) specification document [2]. A GeoTIFF is simply a TIFF which has included tags which are specific to the topography data. For more information see the GeoTIFF Specification [3]. The following are the TIFF Tags which were utilized in this project:

2.1 Tag 257 = Image Height

The height of the image in pixels. Data type is an int.

2.2 Tag 256 = ImageWidth

The width of the image in pixels. Data type is an int.

2.3 Tag 33550 = ModelPixelScale

The amount of decimal degrees per pixel. Decimal degrees express latitude and longitude geographic coordinates. Latitude and longitude values are bounded by $\pm 90°$ and $\pm 180°$ respectively. Data type is a double.

2.4 Tag 33922 = ModelTiePointTag

Conversion from pixel coordinate to latitude & longitude coordinate for the TIFF. Data type is a double array of size 6 (double [6]). The format for the array is [I,J,K,X,Y,Z], where (I,J,K) represents the pixel coordinate and (X,Y,Z) represents the longitude,latitude. K and Z are typically zero. If (I,J) = (0,0) and (X,Y) = (50,50) then the left corner pixel corresponds to (50°,50°) on a map. Moving to the left across the image corresponds to increasing longitude; moving down across the image corresponds to decreasing latitude.

2.5 Tag 34737 = GeoASCIIParamsTag

A description of the geocoding coordinate system used. The World Geodetic System (WGS) is a standard for use in cartography,geodesy, and navigation. The latest revision, WGS 84, is the reference coordinate system used GPS. Data type is an ASCII String.

2.6 Tag 34735 = GeoKeyDirectoryTag

Conversion from Raster Data to Earth model. Elevation data for WGS 84 is in meters. Data Type is a double. A value of 1 for the GeoKeyDirectoryTag means that the value for the pixel is equal to the number of meters.

3 Acquiring Digital Elevation Model(DEM)

The class *getEarthDEM.java* was programmed to receive a TIFF File as input and output a $Point3D$ two-dimensional array ($Point3D[][]$). The two main libraries that were used for this were the Java Advanced Imaging API (32 bit, Release 1.0.1) [4] and the JavaFX API [5]. The first step of the program was defining a $FileSeekableStream$ which would be used to access the TIFF file. A $TIFFDirectory$ Object was defined which provided us with the necessary tag information. A $TIFFImageDecoder$ Object was also created to decode the TIFF file as a Raster. We then created the Point3D two dimensional array $earthDEM$ to store the elevation values from the Raster.

4 Writing STL File for DEM

Once we had $earthDEM$, we needed to be able to convert from a set of 3D points to an STL File. Java does not have any suitable libraries for reading and writing STL Files, so we wrote an exporter for binary STL Files. The exporter was heavily modeled after an open-source code posted online by 'kschmidt'[6]. Modifications were made to accommodate the JavaFX VecMath Class [7]. The motivation for writing binary STL Files as opposed to ASCII STL Files is that they are smaller and more memory efficient, and thus would be a faster process. For models which have millions of faces, it is important to make them as memory efficient as possible. In addition, we thought it would be interesting as a technical challenge to create an STL exporter in Java so that the process from TIFF file to STL File would be as automated as possible.

The STL File exporter, $STLWriter.java$ provided a template to write each of the faces of the STL File. The actual STL File was created in $createSTLFile.java$, which received the TIFF Filename String as input and output a completed STL file. $createSTLFile.java$ was the overarching program which saw the whole process from TIFF to STL. Each pixel was treated as a vertex on a grid-like surface. For every 4 adjacent points on the grid, two triangles were created. The TIFF images were W*H pixels, which meant the number of triangular faces on the surface equaled 2*(W-1)*(H-1). To create a complete 3D model, we also had to consider the edges and bottom of the "box" that would contain the topographical surface. The faces on the edges equaled 4*(W-1) + 4*(H-1). The number of faces on the bottom of the model was 2. This led to a total of 2*(W-1)*(H-1)+4*(W-1)+4*(H-1)+2 faces. Once all these faces were written to the STL File, the 3D printable model was complete.

5 Results and Discussion

The GeoTIFF files for Earth and Mars were acquired from the NASA Shuttle Radar Topographic Mission [8] and Mars Orbital Data Explorer databases [9], respectively. Figure 1 shows one of the STL models of Earth's topography and Figure 2 shows one of the STL models of Mars' topography. The Earth model is at the location (-125,55) in the top left corner and extending 5 decimal degrees east and south. Taking a visual inspection from Google Earth, it is not immediately clear how accurately the model was depicted. This might have been due to how we sampled the data (for explanation see next paragraph). The same is true for the STL File for Mars located at (50,-10). In the case for the Mars files, this might have been due to the fact that the GeoTIFF files we requested were oddly sparsed.

In terms of improving this method and future work, there should be an algorithm incorporated into this method to be able to stack various GeoTIFF files side by side so that various topographical areas can be considered. Also, the resolution of the model is based on sampling that was taken at uniform intervals. An improvement to this would have been to average the elevation values over each interval. This is what might have caused our resulting STL files to not be as visually similar to other models. The conversion from decimal degrees to meters was done using a linear scaling factor. This is not technically correct because of the fact that the conversion from decimal degrees to meters is dependent on the location of the project, i.e 1 degree at the equator is not the same as 1 degree north/south of the equator. Finally, another improvement that would be made is how to accomodate for negative z values and "NO_DATA" z values in a better fashion (as of now they are just set to 0).

Figure 1: STL File for Earth, located at (-125,55).

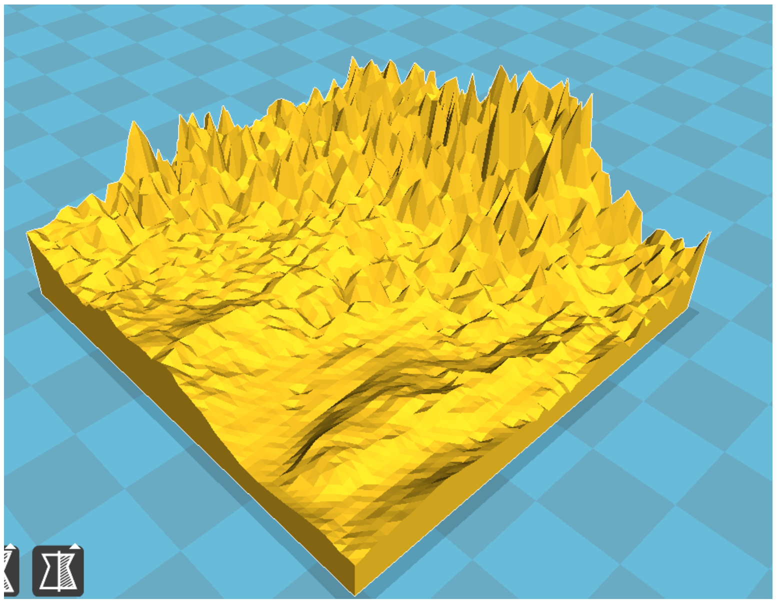
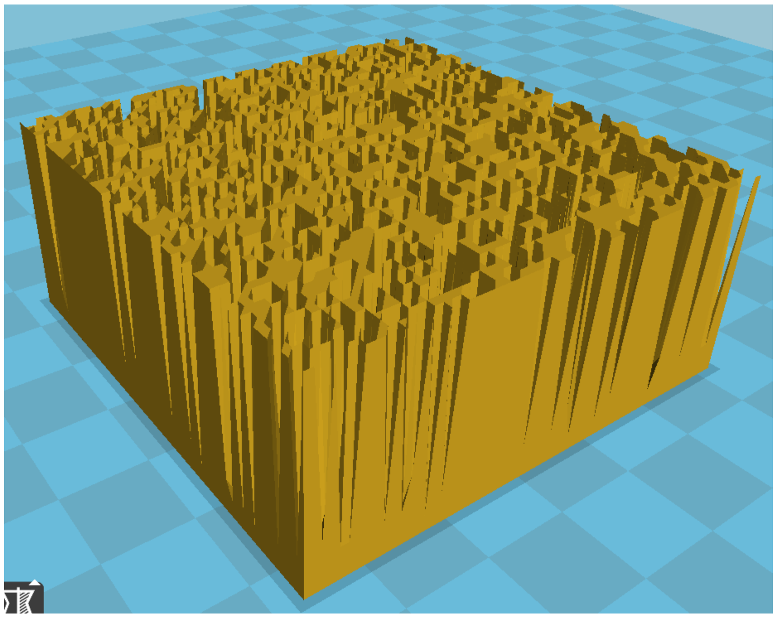


Figure 2: STL File for Mars, located at (50,-10)



References

[1] "STL (file Format)." Wikipedia. Wikimedia Foundation, n.d. Web. 10 May 2016. $< https : //en.wikipedia.org/wiki/STL\_(file_format) >$.

[2] "TIFF Spec." (1992): n. pag. TIFF Revision 6.0 Spec. 23 June 1992. Web. 10 May 2016. $< https : //partners.adobe.com/public/developer/en/tiff/TIFF6.pdf >$.

[3] Ritter, Niles, and Mike Ruth. "GeoTIFF Format Specification GeoTIFF Revision 1.0." GeoTIFF Spec. N.p., 28 Dec. 2000. Web. 10 May 2016. $< http : //www.remotesensing.org/geotiff/spec/geotiffhome.html >$.

[4] Croft, David Wallace. "Advanced Java Game Programming." (2004): n. pag. Oracle. Web. 10 May 2016. $< https : //docs.oracle.com/cd/E19957 - 01/806 - 5413 - 10/806 - 5413 - 10.pdf >$.

[5] "JavaFX Point3D Class." Oracle. N.p., n.d. Web. 10 May 2016. $< https : //docs.oracle.com/javase/8/javafx/api/javafx/geometry/Point3D.html >$.

[6] Kschmidt. "STL Writer." N.p., n.d. Web. 10 May 2016. $< http : //toxiclibs.googlecode.com/svn/tags/20091123 - release/src.core/toxi/geom/util/STLWriter.java >$.

[7] "Vector3d (Java 3D API)." Vector3d (Java 3D API). N.p., n.d. Web. 10 May 2016. $< http : //www.cs.stir.ac.uk/courses/CSC9N6/Java3D/html/javax/vecmath/Vector3d.html >$.

[8] "CGIAR-CSI." CGIARCSI. N.p., n.d. Web. 11 May 2016. $< http : //www.cgiar - csi.org/data/srtm - 90m - digital - elevation - database - v4 - 1 >$.

[9] "NASA Planetary Data System MOLA PEDR Query Tool V2.0 (Beta)." NASA Planetary Geoscience's Granular Data System (GDS) MGS MOLA PEDR Query Tool. N.p., n.d. Web. 11 May 2016. $< http : //oderest.rsl.wustl.edu/GDSWeb/GDSMOLAPEDR.html >$.