# Term Project - Design Document

## “Map to Mesh Generator”

**Authors: Tyler Dinardo & Zachary Sullivan**

**Course:** IMD 3002 – 3D Computer Graphics

**Group Members:** Tyler Dinardo & Zachary Sullivan

**Lecturer:** Dr. Christopher Joslin

**Last Updated:** April 18th, 2015

## Table of Contents

## Revisions

| Date | Description of Changes | Version |
|---|---|---|
| February/26th/2015 | Initial Version of Document | V1.0 |
| March/19th/2015 | Updated Document Featuring Prototype | V1.1 |
| April/18th/2015 | Updated Document Featuring Final Build | V1.2 |

## 1. Objectives

### 1.1 Global Objective

The overall objective of this term project is to create a map to mesh generation tool. This tool will allow a user to import a map image and generate a custom 3D terrain mesh.

### 1.2 Sub Objectives

In order to modularize our script, the tool will be broken down into the following functions. Subsequent **beta** modifications or updates will be marked in red, while comments regarding the **final build** will be marked in green.

**Import New Map Image:** User will be able to choose an image to load and manipulate to create a terrain mesh. The application must be able to save this image for terrain generation. Development for this objective has started and is currently ongoing. Currently users must import a jpeg image, but we intend to support a wider range of file types (Jpeg, PNG, GIF, TIFF).

Users are able to import a variety of different image types, we have tested Jpeg, PNG, TIFF, and Targa. Each of these file types are confirmed to work with our map to mesh generator. It is to be noted that GIF images are not supported in our generation tool.

**User customization:** The user will be given the option to set their own custom color values to define how the map to mesh tool will read the elevation values.

User customization has been fully implemented, as well as a fully functional user interface.

**Toggle to Voxel Art Style:** Users will be able to toggle between two different art styles of final map generation. The first option would allow for a voxel style generation, a layer of cubes will be created to generate a detailed terrain model. The second option would allow for a smoothed mesh terrain. Our team has begun implementation of the voxel generation style. During the development of this feature, I [Zach] ran into a problem when trying to manipulate each individual vertex Y position to correspond to an HSV hue range (between 0 - 1). I believe this issue was caused by an excess amount of for looping, causing the generation tool to significantly decrease in performance. This issue should be resolved for the final build.

Toggle to voxel art style was partially completed in the final build. Our team has implemented map generation through vertex manipulation of a target poly plane. Issues encountered during the beta prototype have been resolved. Our team did not include the voxel art style, as we found that performance significantly decreased when generating layers of poly cubes. Instead, we opted to focus development on map generation through vertex translations of a poly plane.

**Generate Map:** The tool will allow for manipulation of geometry to create a detailed terrain model of the user imported image. A significant amount of progress has been made on map generation. Tyler was able to create a HSV color conversation algorithm to change the input RGB values of the user selected image. The hue range (a value between 0 or 1) is used to dictate the vertical scaling factor of each mesh polygon. With this technique, we are able to import a jpeg map of the world and generate a detailed voxel style.

Map generation has been completed in the final build. Tyler was able to further refine the map generation based on the previous HSV conversion algorithm featured in our beta submission. Our final build now features two different generation options which the user can choose between. The first is based on a greyscale map which distinguishes elevation between brightness. The second option allows the user to generate a map based on varying hue values featured in the map.

## 2. Specifications/Requirements

Each section outlined below refers to the sub objectives outlined in Section 1.2, with the name of the group member in brackets next to the task.

**Import New Map Image (Tyler):** User will begin by navigating through a file structure to load an appropriate map image into the program. Users will be able to import images which meet the required file type, currently this is Jpeg images. The application then saves this image for terrain generation later.

**Set Hue Range (Zach):** The user specifies a custom hue range, modifying the lower and upper bounds of the map generation through a slider input. These hue values will later have the max and min elevation range associated to them accordingly. Currently hue range is obtained directly from the pixel value of the HSV image, this does currently result in small terrain generation errors (purple for instance registers as a low terrain elevation rather than middle transition between max red and min blue heights). Users will be able to customize the hue range in the final build.

Set hue range has been implemented in the final build. Users can now choose a maximum and minimum hue value to distinguish varying elevations.

**Set Elevation Range (Zach):** Users will need to specify a custom elevation range, setting both the lowest and the highest possible terrain elevation. Users will change this value through a slider input. This feature is currently in the early stages of development, elevation in the prototype is currently fixed to a pre-defined scale value. This fixed elevation value is scaled by the hue range found in the RGB to HSV conversion. A user defined elevation range will be incorporated in the final build.

Set elevation range has been completed and implemented in the final build. Users can specify a maximum value which will scale the map generation vertically, a larger elevation value will result in an exaggerated elevation map. Originally our team proposed both a maximum and minimum elevation slider. This has been replaced by a single max elevation option, as our map generation only transforms the mesh vertices vertically, while lowest elevation points are not transformed. This results in the lowest points of the map to remain flat while mountain peaks are translated vertically.

**Set Resolution (Tyler):** Users will be given the ability to set the accuracy of the terrain generation through a percentage value calculation. The will user begin by specifying the percentage of subdivisions found in the overlaid matrix through a slider input. The program then later determines how many vertices are generated to represent the image data, then stores this value.

Resolution has not been currently implemented in the prototype version of the script.

Set resolution has been implemented and completed. Users can either increase or decrease the resolution of the final map. Lower resolution maps will be generated faster, but will have less detail then higher resolution maps. This is achieved by scaling the vertex count by a user defined percentage.

**Set Surface Smoothing (Zach):** Users will be provided with the option to smooth the final terrain, as well as manipulate the parameters of the smoothing algorithm. Due to issues in manipulating individual vertex position based on hue range, mesh surface smoothing has not been implemented in the prototype. This feature is still planned for the final build, but may be restricted to either voxel smoothing or mesh smoothing.

Set surface smoothing has been completed and implemented into the final build. Users can increase the smoothing division count. A higher division count will cause a map generation times to increase, but will result in a smoother transition between maximum and minimum elevation points.

**Toggle to Voxel Art Style (Tyler & Zach):** Users can specify a toggle to either create a polymesh or a polycube voxel terrain. If users chose to generate the terrain using a voxel art style, they could declare if the color of each polycube should match the corresponding elevation hue value on the original map.

**Generate Map (Tyler & Zach):** The python script will take the previously imported image and query all relevant color information based on each given pixel within the image. The number of evaluated pixels depends on the user specified resolution of the generated mesh. The program then takes this pixel information and depending on the hue and elevation ranges specified by the user, dictates the y position of a new vertex. The x and z values of the vertex are determined by the pixels x and y coordinates in the image. These vertices are then used to define a polymesh, which will be the final terrain mesh.

## 3. Layout and Structure

### 3.1 GUI Procedure

The first procedure the user will interact with is the GUI. The GUI will contain all elements associated with the creation of the mesh, as well as all elements related to user customization.

The GUI can be launched from the Python Script dialogue box and will function such that only one instance of the interface can ever be active at any given time.

The GUI has not been implemented in the current prototype, development has instead focused on generating a detailed map directly from a loaded image. A GUI will be featured in the final version.

The GUI has been implemented in the final build, we have separated user interface into three main sections (General Settings, Hue-Based Settings, and Image Importing/Generation).

### 3.2 General Methods

The following methods handle tasks and operations related to our main objectives.

- *importImage( fileName , fileType );* Handles image file importation and stores pixel information
- *generateMap( imagePixels[][] );* Takes pixel information and generates a mesh based upon user specified parameters

Currently map generation resides within the importImage() function, this was done to simplify the logic early on and focus on creating an effective map generation algorithm. Python code to generate the terrain map will be removed from the importImage() function in the final version.

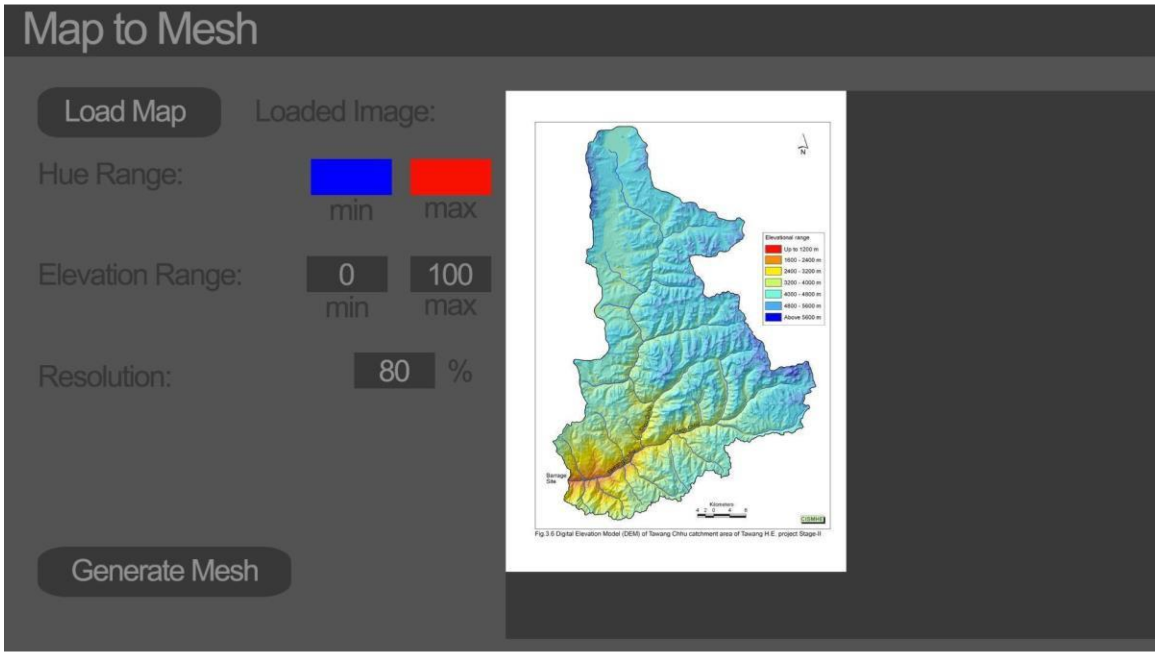
The final script has been updated to follow a modular design. Map generation no longer resides within a single overall function. New functions have been implemented to contain each major requirement.

## 3.3 Secondary Methods

The following methods handle tasks and operations related to user customization and other secondary objectives

- *setHueRange();* Specify the minimum and maximum hue ranges that will be associated with the lowest and highest heights respectively.
- *setElevationRange();* Specify the minimum and maximum height of the final generated mesh.
- *setResolution();* Specify the resolution of the final generation operation. A higher resolution would result in a higher percentage of pixels being evaluated.
- *setSurfaceSmooting();* Specify the subdivision amount to be applied when smoothing final mesh.
- *toggleVoxelStyle();* Specify whether the final render will be outputted as one generated mesh  or as a series of voxel-like blocks stacked atop one another to create a block representation of the map.

## 4. Design Sketches

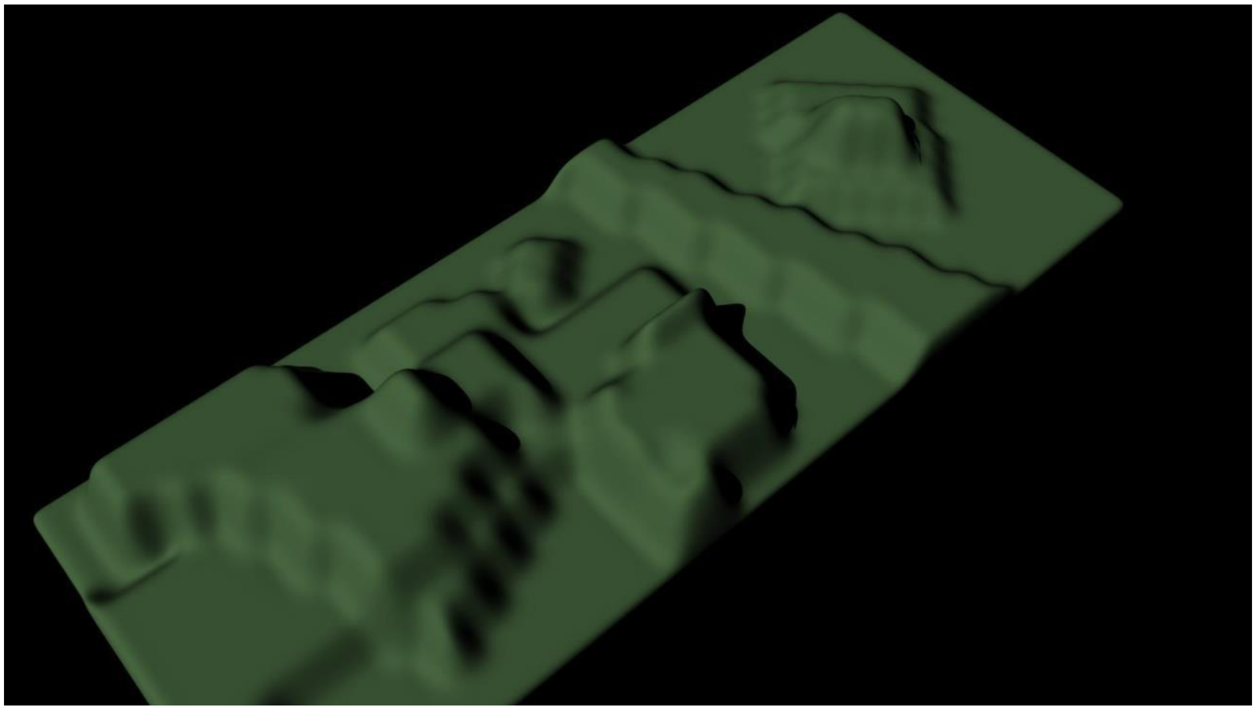


Hue Range: 🟦 min 🟥 max

Elevation Range: 0 min 100 max

Resolution: 80 %

The following sketches provide an early concept of the map-mesh plug-in for Maya.
*(Figure 1 above) A prototype sketch of the GUI Interface for the Map to Mesh Maya Script*
*src: [http://www.apspcb.org.in/tawang_heps.htm](http://www.apspcb.org.in/tawang_heps.htm)*



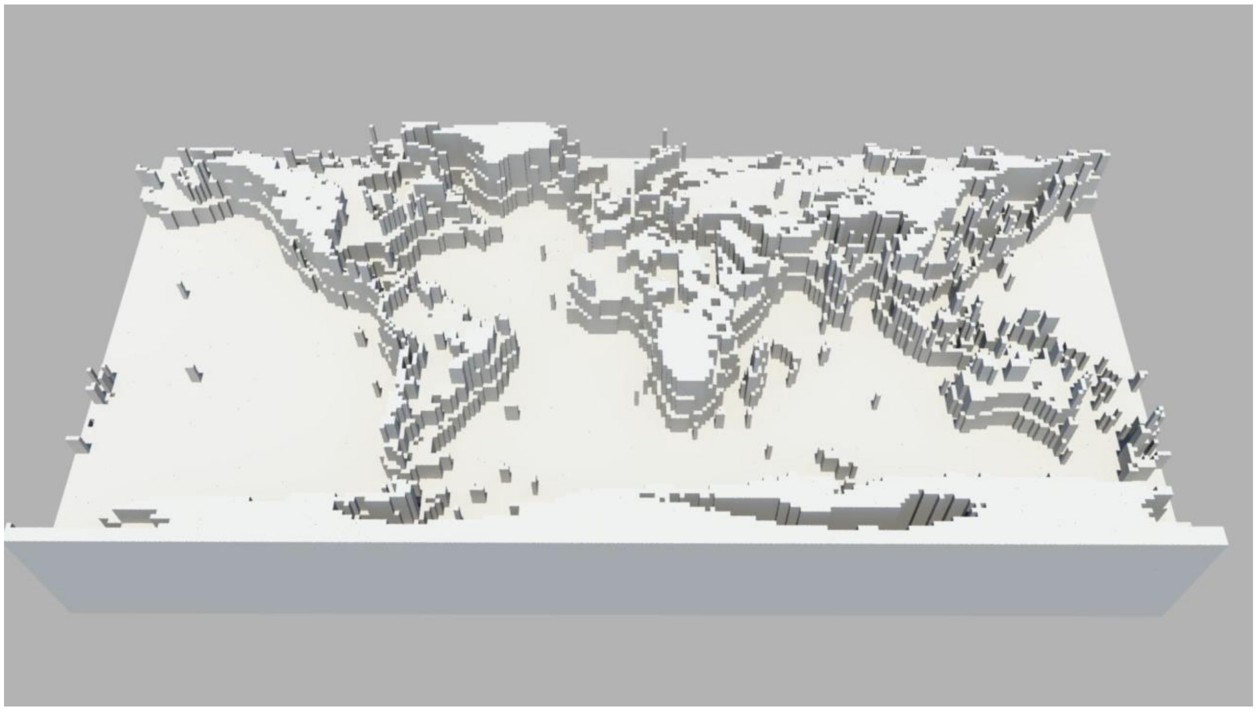
*(Figure 2 above) A hand crafted example of what the final map render would look like after being rendered in Maya.*

## 5. Progress Made (Beta)

### 5.1 Development Procedure

Our team began prototype development by creating a method for users to import an image into Maya. Originally we had intended to create an image plane and create a 3D polygon mesh over top. Instead, we opted to utilize the Python image library (*Python Imaging Library PIL*, 2009) to analyze the RGB value of each pixel. This was done to reduce overly complex code, as well as optimize the mesh generation algorithm (although obtaining the image pixel values will likely need further optimization). Once the RGB values of each pixel were stored, they needed to be converted to HSV to obtain the hue range. Utilizing a method taught in Design Studio 3, we applied a conversion algorithm to pass each pixel's RGB value and obtain the corresponding HSV value.

After obtaining the HSV value for all pixels found in the imported image, we created a simple polyCube to manipulate the terrain into. The elevation value in the current prototype is fixed to a pre-defined scale value. This fixed elevation value is then scaled by the hue range found in the RGB to HSV conversion. This process is then repeated for each pixel found in the image, with a corresponding polyCube created and translated to the pixel location, resulting in a final model seen in figure 3 below.



*(Figure 3 above) Map to Mesh generation prototype*

## 5.2 Prototype Changes

A few functional changes were made during the development of the prototype. For instance, our document originally stated that we would allow users to manipulate the elevation of the terrain model. This was replaced by a fix value in the prototype, in order to focus development time away from creating a GUI interface and instead create the core functionality of the tool.

## 5.3 Future Challenges

During this prototype development, I found that manipulating the individual vertices by each corresponding HSV pixel value to be more challenging than originally anticipated. This may cause future issues in generating a smoothed terrain mesh based on a polyMesh's individual vertex elevation.

Another issue I faced during the prototype development was the ability to apply individual colors to each polyCube that made up the terrain. The objective of this was to improve the visualization of the model, and show users the range color values which correlated to the terrain's mesh. In order to accomplish this I had planned to provide each polyCube a unique name. Our generation tool created a terrain from a 240x120 image, for a total pixel count of 28,800 pixels. In our current prototype, each pixel has a unique polyCube associated to it, resulting in a total of 28,800 cubes. When assigning a unique name to each of these cubes, our mesh generator began to significantly loose performance and freeze. As a result our group chose to focus on developing a detailed terrain generation tool for the prototype to prove its capability, rather than redesign a newly optimized generation method.

In the next revision of our tool, we will need to address this not optimized method to both increase performance, as well as allow for improved visualization through colored terrain. This will likely be the hardest part going forward with this project.

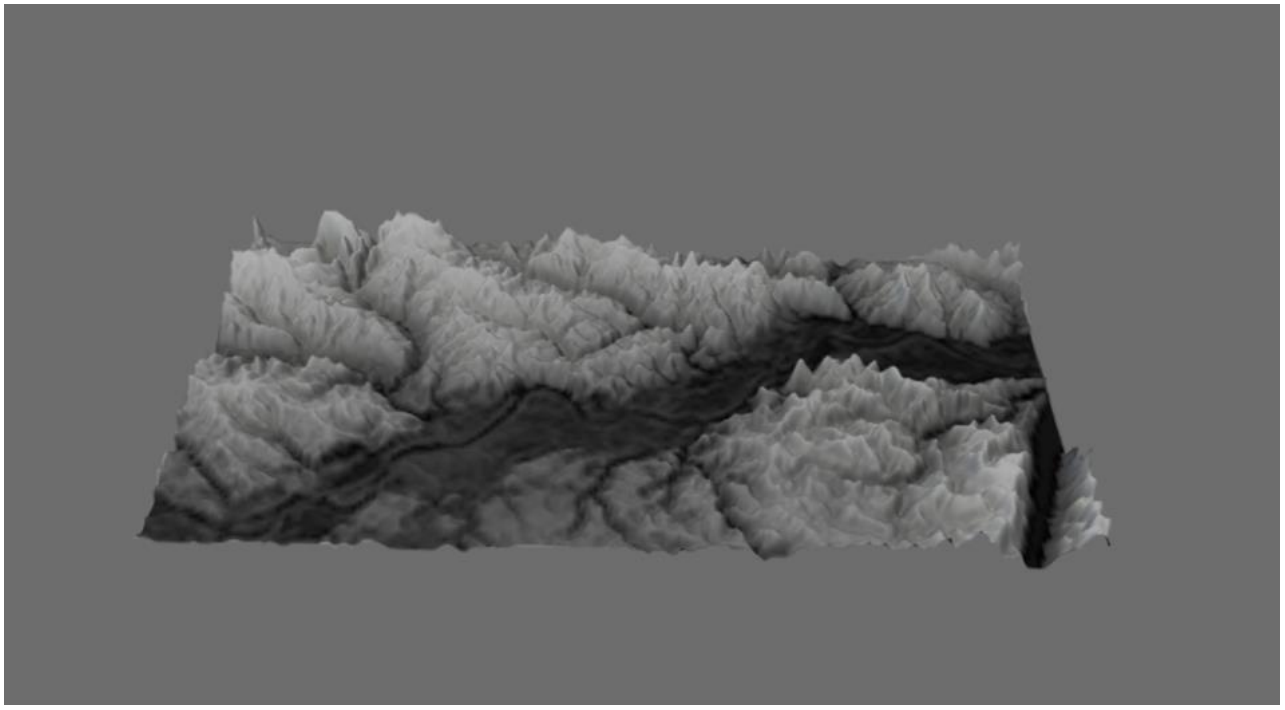
## 6. Progress Made (Final Build)

### 6.1 Development Procedure & Methedology

Development of the final product began by updating the method for users to import an image into Maya. Originally the script would automatically open a file explorer, this was changed with the implementation of our GUI. Now users can change various map generation settings before processing an imported image.

Python image library (*Python Imaging Library PIL*, 2009) is still being utilized to obtain each RGB pixel value of an imported image. As previously mentioned, this was done to reduce code complexity, as well as optimize the mesh generation algorithm. Once the RGB values of each pixel were stored, they are either then converted to HSV to obtain the hue range or passed to a lightness function which will average the maximum and minimum RGB values.

After obtaining the HSV value for all pixels found in the imported image, a base polymesh is then created, and each vertex is translated vertically based on one of two different generation algorithms. The map is produced by either the brightness or hue of the pixel. This is chosen by user selection before map generation begins. Once all vertices found in the mesh have been cycled through, the final generated map is produced (Figure 4 below showcases a map generated on source pixel lightness).



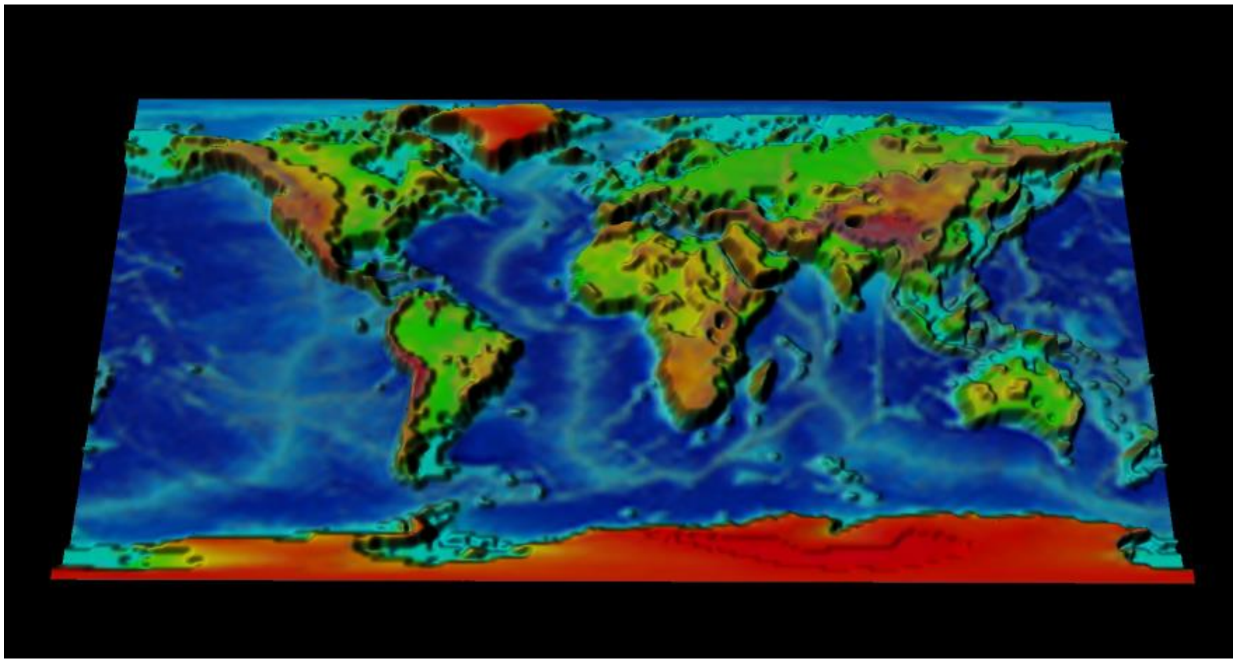
*(Figure 4 above) Map to Mesh generation based on pixel brightness*

## 6.2 Final Build Changes & Experiments

A few changes were made during the development of the final build. For instance, we originally stated that we would allow users to toggle between a polycube and polymesh generated map. This was modified to only allow a polymesh as map generation option, as map generation which utilized the polycube option suffered significant performance loss. This occurred as a result of large polygon counts, and further escalated when users increased the resolution of the map generation.

Our team also removed the option to modify the minimum elevation of the map generation. This was done as only positive vertical changes in elevation are applied to the vertices found on the target polymesh. This reduces the amount of computations required and as such reduces the overall map generation time.

During the development of the final build, I began experimenting with various methods to apply color to the generated polymesh map. I first implemented a method to iterate through each vertex of the polymesh and assign the corresponding pixel's RGB value to the color of the vertex. I found this method to be extremely tedious, and replaced this by creating a unique shader for each map created, and assigning the corresponding map image as a texture to the polymesh. This updated method produced a significantly improved aesthetic with reduced code complexity (An example can be found in figure 5 below.).



*(Figure 5 above) Map to Mesh generation featuring source image texturing & unique shaders*

Finally a polished user interface was implemented in the final build. Users are no longer prompted immediately to choose a source map image, and can now modify various generation options (such as elevation, mesh smoothing, map resolution, etc.). A preview image will now display when a source image has been selected, as well as a progress meter will show the final map generation time.

### 6.3 Challenges

I found that creating a map generation algorithm which utilized polycubes rather than a single polymesh to be the most challenging issue during the final development stage. Rather than utilizing the subdivision count found on the polymesh to increase the density of the map, the polycube method instead required us increase the number of polycubes on screen. This resulted in a significant loss in performance. In order to resolve this issue, I began development on a new method which extruded the faces of a polymesh. While this method did increase performance during map generation, the final product did not visually distinguish itself enough from the standard vertex transformation method. As such, we decided to focus development on the vertex manipulation approach.

Another issue I faced during the final build development was the ability to apply individual colors to the final map mesh. As previously mentioned in section 5.3, the objective of this task was to improve the visualization of the model, and show users the range color values which correlated to the terrain's mesh.

In our current build, a unique shader is created every time a new map is generated. A random value will be assigned to the shader (as well as the texture and shading group) to distinguish it from future maps. Once this process has been completed, the source image file used in the generation of the map, is assigned as a texture to the mesh.

## 7. References

PythonWare. (2009). *Python Imaging Library (PIL)*. Retrieved from http://www.pythonware.com/products/pil/

APSPCB. (2009). *Arunachal Pradesh State Pollution Control Board*. Retrieved from *http://www.apspcb.org.in/tawang_heps.htm*