Terrain Generation

Srinivasa Santosh Kumar Allu

***Abstract*—This paper discusses about implementing some terrain generation methods like heigh maps and control maps and then discusses about applying lighting and texturesto get realistic view of the terrain.Later it discusses about the limitations of the height maps and also discusses about delaunay triangulation which is one of the optimized method for terrain generation.**

***Index Terms*—Height maps, control maps, delaunay triangulation.**

I. Introduction

Now a days terrain generation is widely used in video games, displayed as background or part of scenes in many animated movies. Many algorithms which can generate terrains look realistic but have poor space and time complexity. Therefore there is a need for generating terrains which are fast,efficient and occupies less space. Height maps and control maps are some of the methods which can be rendered fast,light weight,use quite a small footprint for memory and easy to generate procedurally.Below are the topics which will be covered in the following sections:

1) Height maps and its limitations
2) Control Maps
3) Lightings and Textures
4) Results
5) Delaunay Triangulation
6) Conclusion

II. Height maps

In computer graphics, a heightmap is a raster image used to store surface elevation data for display in 3D computer graphics.A heightmap is a grayscale image which is generally represented by 2-D array of pixels. In heightmaps white pixels indicate the highest point and black pixels indicate the lowest point. The color values will be in the range of 0 and 255 for 8-bit. You can exploit the use of individual color channels to increase detail.For example RGB 8-bit can show only 256 values but for 16-bit or 24-bit image you can represent 2^{16} or 2^{24} different values of heights.By using more number of values you can have different heights and by using wide range of heights terrains looks more realistic without having sharp edges. Heightmaps can be created using cloud filters in Photoshop ,plasma in the GIMP, Perlin noise etc.

A. Implementation

To create height maps you can use any of the methods discussed above. I used GIMP which is a raster graphics editor used for image retouching and editing, free-form editing, resizing, cropping, converting between different image formats and more specialized tasks to create height maps. First select file and then select new option to create an image and then give height and width of the image and then select as gray scale image.After creating the image, go to filters → Render → Clouds → plasma. After selecting plasma a gray scale image will render and then you can apply any of the textures using brush or airbrush from the tool box . You can also create height maps by just creating the image and setting background color as black and then you can apply any of the textures or patterns by selecting any of the brush or airbrush or any tool from the toolbox.After creating height map you should export it as PPM (Portal Pix Map) file.

In PPM file, the first few lines are defined as the image header and provides an overview of the contents of the image.(Fig2: is an example of PPM format).The first line

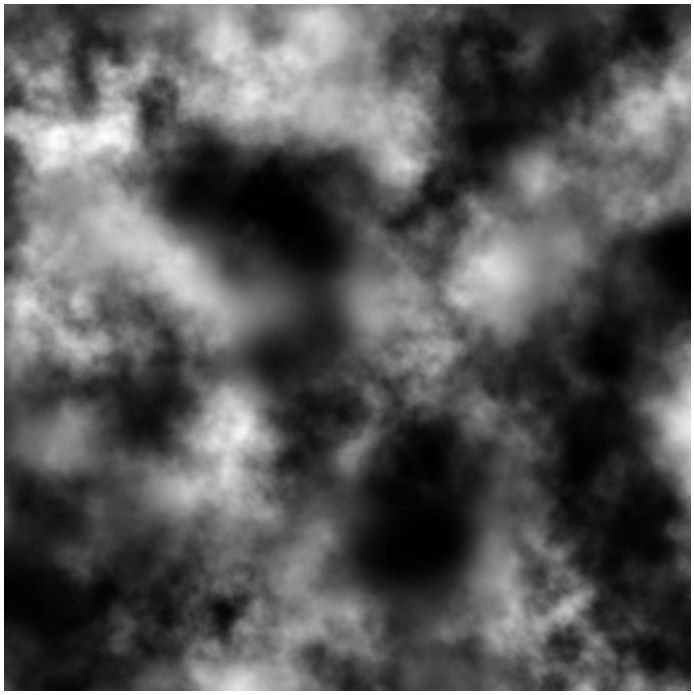

Fig. 1. Sample height map (source: http://en.wikipedia.org/wiki/Heightmap).

```
P3
4 4 number of rows and number of columns
255 - maximum allowed colour
0   0   0     100 100 100     0  0  0     255  255 255
0   0   0     175 175 175     0  0  0     0    0   0
0   0   0     0   0   0       15 15 15     0    0   0
255 255 255   0   0   0       0  0  0     255  255 255
```

Fig. 2. Sample height map (source: http://en.wikipedia.org/wiki/Heightmap).

indicates the image format and second line indicates the number of rows and columns.The next few lines contains pixels with RGB values.As it is a grey scale image the RGB values for each pixel will be almost same. First step includes parsing PPM files for heighmaps and textures. And then each and every pixel of height map and take any of the RGB values as the picture is a gray scale image.The color value represents the height of the pixel. So three dimensional vertices are plotted by taking y-coordinate as color value as height coordinate of the pixel and x and z coordinate as the position of pixel in heightmap 2-d array(row , column position). After plotting the vertices the region is triangulated by joining the vertices.

B. Advantages of Height Map

1) Looks like a real terrain from a distance
2) Very easy to compute height and to create terrains
3) Light weight and very fast to load
4) Easy to generate procedurally
5) Easy to create and edit using graphics editors like GIMP

C. Limitations of Height Map

1) Use memory inefficiently when there are large flat areas
2) Can't represent steep or overhanging terrains
3) If the four vertices that define a sub-quad aren't on the same plane, the split between the two vertices will become visible.

Fig. 3. Hanging Terrain (source: Google Images).

III. Control Maps

A control map is similar to height map but it helps to indicate the regions where more details need to be added. A control map is a gray scale image where white regions are given more importance than the darker regions and this clearly helps in demarcating control regions. By demarcating regions, we can densely triangulate the regions where the camera focuses more often.Control maps also useds in tracks and also used for creating maze which is mostly used in video games.

Fig. 4. Sample Control Map (source:http://www.hindawi.com/journals/ijcgt).

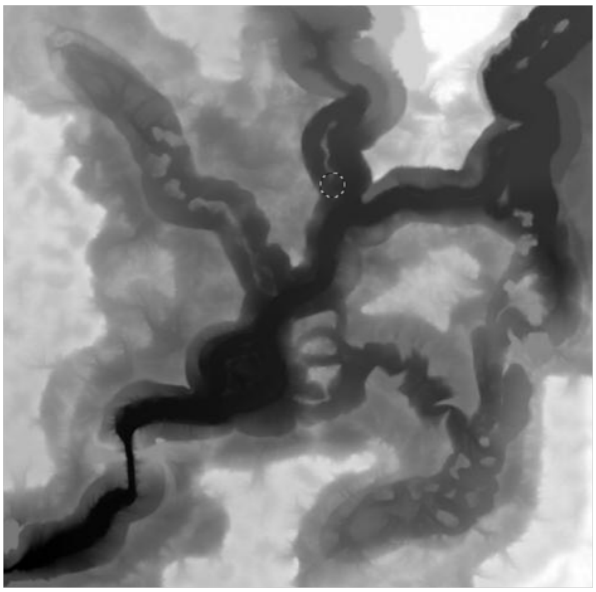

Fig. 5. Sample Height Map (source:http://www.hindawi.com/journals/ijcgt).

IV. Lighting and Textures

For implementing lighting functionality I chose vertex normals over face normals because vertex normals provide smooth gourard shading where as face normals do not provide smooth shading.

A. Vertex and Face normals

A vertex normal at a vertex of a polyhedron is defined as a directional vector associated with a vertex, intended as a replacement to the true geometric normal of the surface.

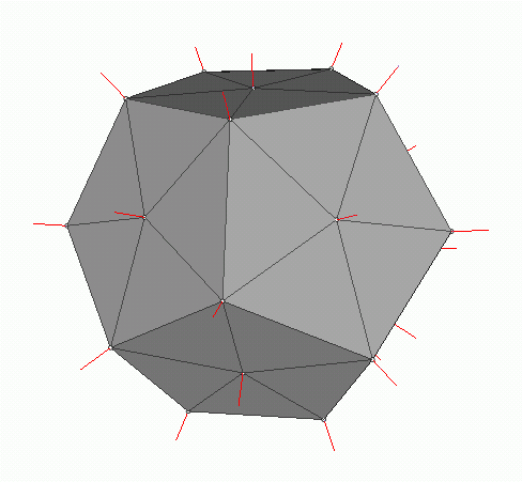

Fig. 6. Vertex Normals

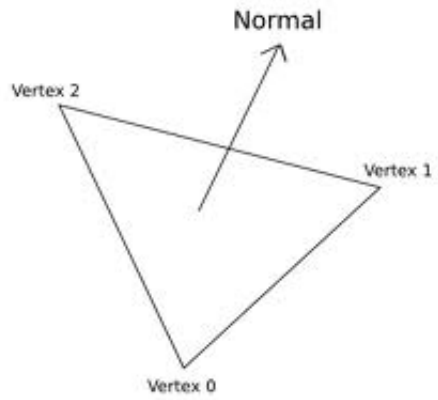

Fig. 7. Vertex Normals

B. Lighting Implementation

For lighting calculation, first I calculated the surface normals by taking cross product between two of its edges and then normalized the result. After calculating the face normals , we calculate vertex normals by taking the sum of the adjacent vertex normals. For each vertex there would be six adjacent vertices connected to it. So six face normals will be formed by joining those vertices for a given vertex. The vertices which are adjacent to a given vertex are the upper, upper right vertex, right vertex , left vertex , down vertex and down left vertex. This scenario is when triangles are connected at 45 degrees and it is vice versa for 135 degrees.

The images displayed in fig 8 and fig 9 are small so you couldn't really validate it but for large images you could clearly see the difference and find that the lighting is not smooth.

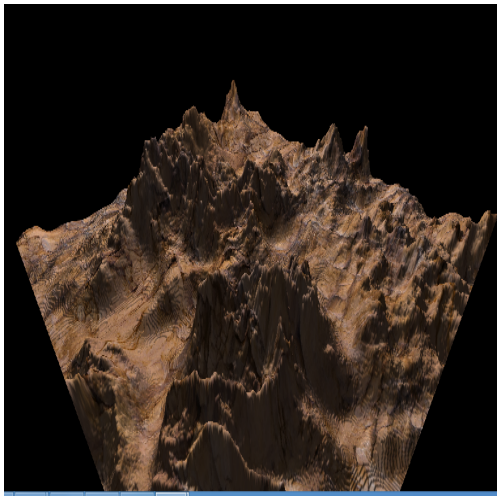

Fig. 8. Light computed using face normals

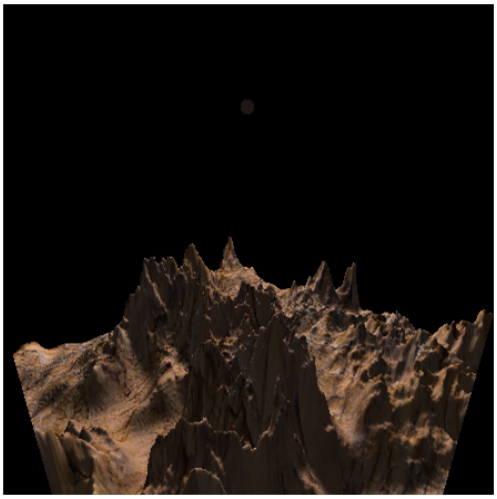

Fig. 9. Light computed using vertex normals

C. Textures

The textures used in this application are loaded as PPM files.First the textures are read from PPM file and passed as a parameter to glGenTextures.I used GL REPEAT when coordinate is out of the range. I used GL NEAREST instead of GL LINEAR because I wanted to display mountains with sharp edges instead of blurred mode.

V. Results

This section displays the results of the project when different height maps and control maps applied with different textures

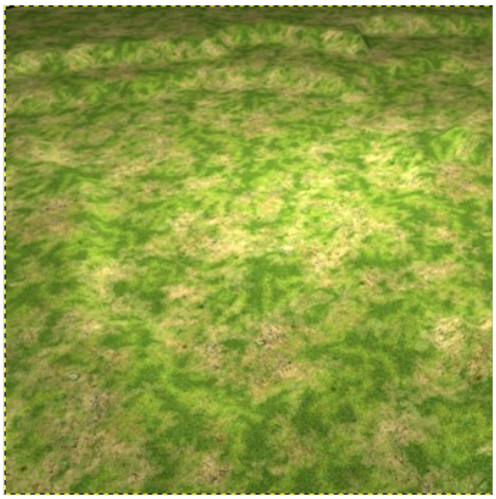

Fig. 10. Grass texture used in the application which is stored as ppm

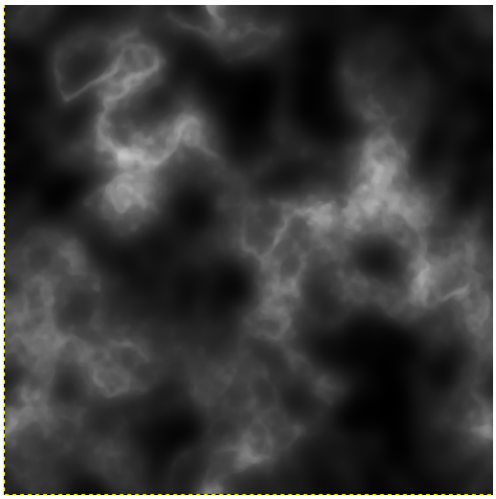

Fig. 11. Height Map used in the application which is stored as ppm

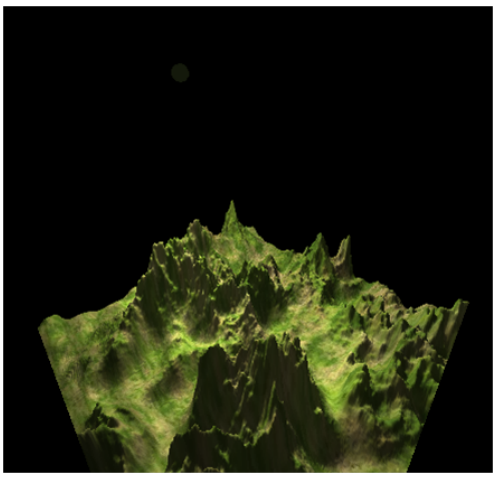

Fig. 12. output after applying texture in fig 10 and heightmap in fig 11

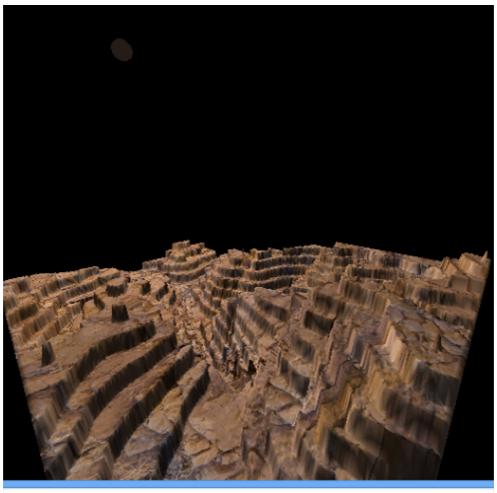

Fig. 13. output after applying stone texture and different height map

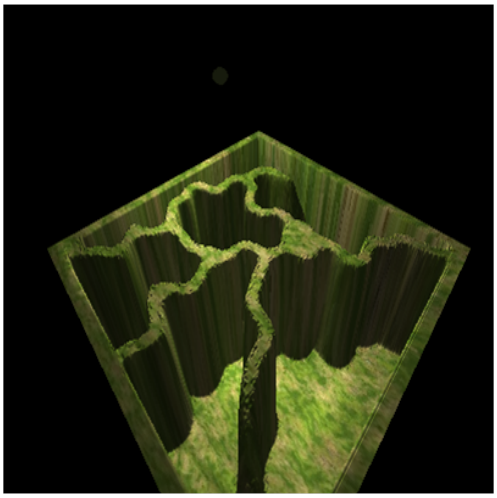

Fig. 14. output after applying grass texture and control map in figure 4

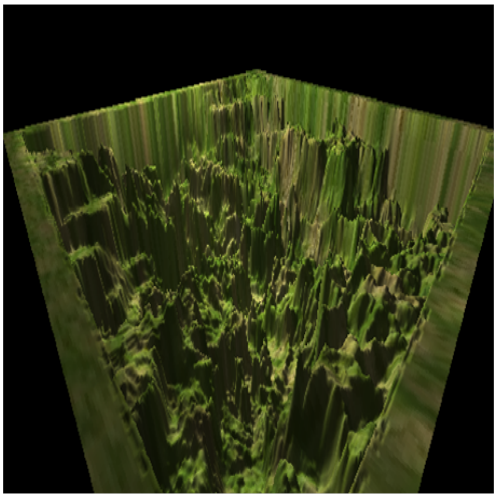

Fig. 15. output after applying grass texture and height map in figure 5

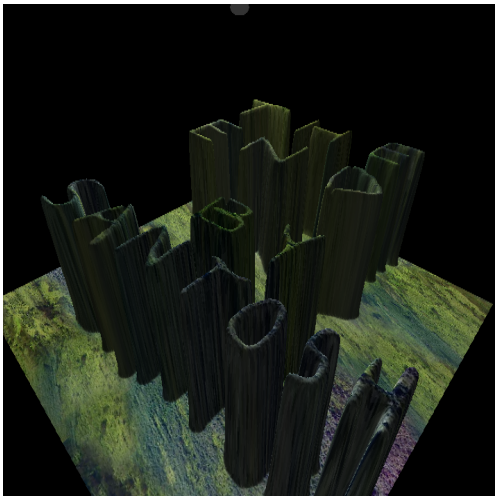

Fig. 16. output after applying different grass texture and a different control map

VI. Delaunay Triangulation

Even though heightmaps are one of the efficient ways of generating terrain, there is a need to create terrains with minimum number of triangles and vertices by reducing skinny triangles. Delaunay triangulation will be helpful to achieve the above scenario. Delaunay triangulation takes a set of points and triangulates them.But which vertices will be taken to construct triangles and remove skinny triangles is implemented using Delaunay algorithm. Delaunay triangulation for set of points in a plane is defined as for any triangle if you draw a circumcircle of that triangle then there should be no point of the other triangle inside of that circle and this applies to all the triangles for given point set. As delaunay triangulation reduces skinny triangles it directly maximizes the minimum angles in the given triangle.

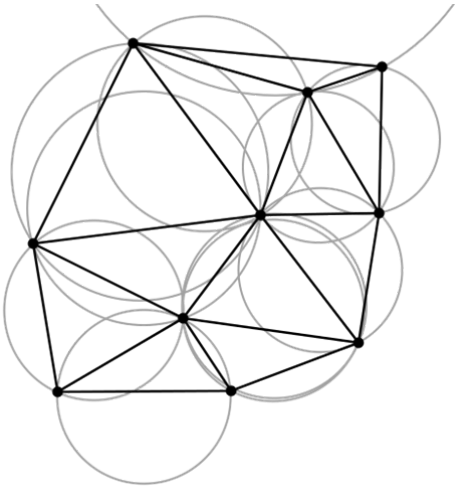

Fig. 17. Delaunay triangulation in the plane with circumcircles(Source:wikipedia)

I have implemented delaunay triangulation using edge flip algorithm for my computational geometry class.Edge flip algorithm is implemented if two triangles do not meet delaunay triangulation property .The edge which is shared between two triangles will be flipped. Flipping the edge is done by connecting the diagonal vertices. As already one diagonal is connected we try flipping with the other diagonal.This algorithm runs in $O(n^2)$. And there are algorithms which implement Delaunay even faster like incremental algorithm runs in $O(nlogn)$. Edge flip algorithm is not applicable for three dimensions or higher.Even though the algorithm I have implemented is applicable for 2d version we can still able to create visual effect of terrain by creating clutter of points. In some regions I have placed more vertices where I could get more triangles and the other regions I have placed less vertices so that there will be less triangles and by proper lighting and coloring I was able to generate approximate visual effect of terrain.

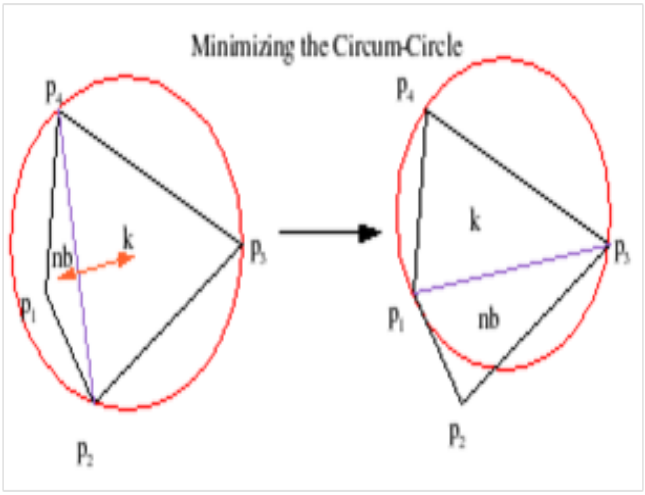

Fig. 18. Edge flip algorithm example

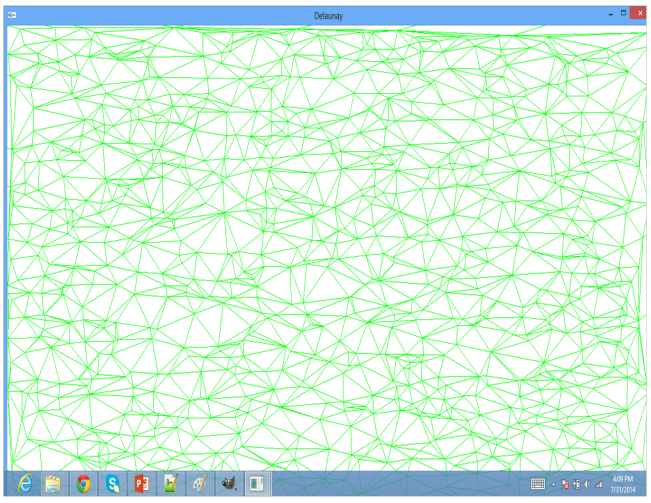

Fig. 19. Delaunay triangulation for a random point set of 500 points and this runs in $O(n^2)$

Fig. 20. Delaunay 2d terrain for a point set with distributed clutter of 500 points and applied color based on vertex coordinate

VII. Conclusion

From the above implemented methods and also by reading different papers I can conclude that height maps are one of the efficient ways of generating terrains and which looks realistic from a distance but there are also methods like delaunay which are really fast and use less space like less number of triangulations and still look realistic . I tried to generate 2-d terrain based on my previous semester classwork assignment and I was partially successfully and the result can be seen in fig 20.In the future I would like to implement delaunay triangulation for three dimensional space and generate terrains.

Acknowledgment

I would like to thank my professor for asking me a question during presentation which me made think and helped me to implement the generation of terrain using delaunay based on the answer I have given about generating points in clusters during presentation .

References

[1] Sundar Raman and Zheng Jianmin, “Efficient Terrain Triangulation and Modification Algorithms for Game Applications,” School of Computer Engineering, Nanyang Technological University, Singapore 639798.
[2] “http://web.eecs.umich.edu/ sugih/courses/eecs494/fall06/lectures/workshop-terrain.pdf”
[3] “http://en.wikipedia.org/wiki/Delaunay triangulation”