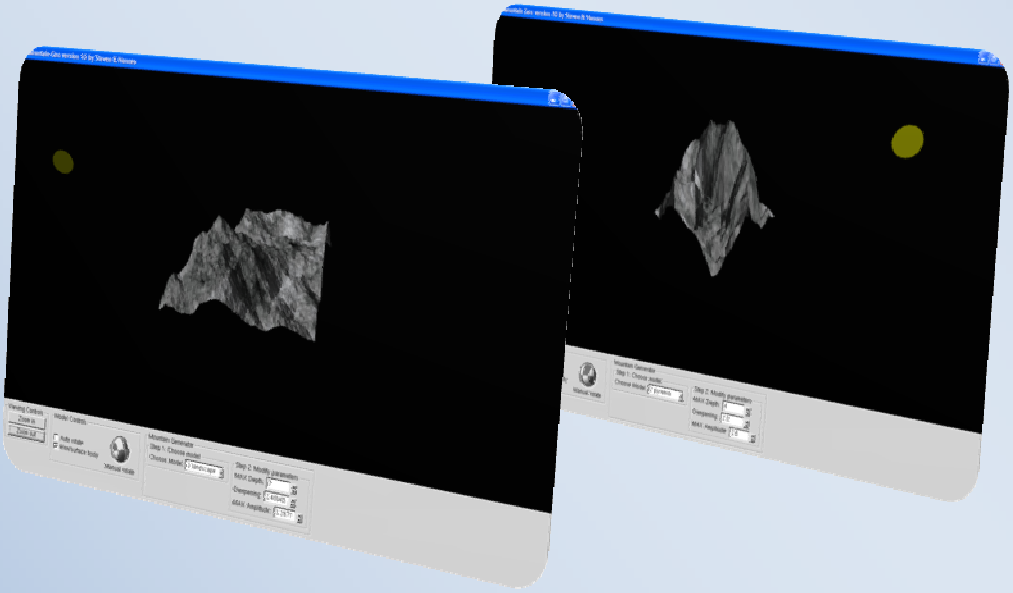
Report assignment 1

Construction of a fractal mountain landscape using a custom made mountain generator



Course

IN4151 3D Computer Graphics and Virtual Reality (3DCG&VR)

Version

Final (oktober 2008)

Group 3

Steven Bos	1319671
Hannes Smit	1228897

Introduction

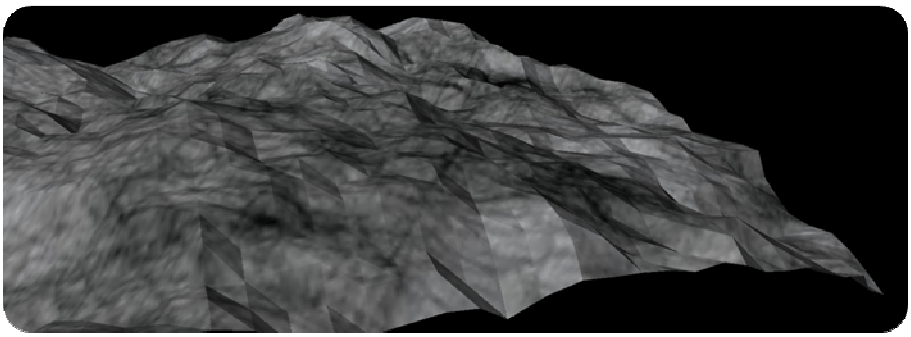
When introduced to the subject of 3D Computer Graphics and Virtual Reality, we decided to take a closer look at Fractals. The Fractal Mountain assignment seemed interesting because we liked the general idea of writing a small piece of code and generate something which appears really complex and realistic. With some experience with the Koch curve, Mandelbrot and the 3D viewing pipeline in general, we started this assignment.
When looking in literature and examples on the web, we were really amazed with whats possible with algorithms like this. Probably more techniques are used in most cases, but we know a lot can be done by using fractal algorithms on itself.
I this report we show what we developed in the past weeks. An introduction to our software is described in chapter 1 (Installation) and chapter 2 (User Interface).
In chapter 3 we discuss our algorithm, our implementation and problems we ran in to. Besides the fractal algorithm we discuss other improvements like shading and textures in chapter 4. In chapter 5 we review the assignment and discuss interesting topics for future work.

Enjoy reading,

Group 3

Hannes & Steven

Table of Contents



1. Installation

S+H Mountain generator is written in C++ using OpenGL and developed in MS Visual Studio 2005. It is tested on Windows XP only. The software is dependent on several libraries which are not present by default. This section will describe how to setup MS Visual Studio 2005 and play with the code. The included SHmountainGen executable does not need any installation; just run it.

§1.1 dependecies

This software makes use of:

- OPENGL and its default GL,GLU,GLUT libraries
- GLUI (an extension of GLUT, providing basic UI controls)
- MersenneTwister (for more realistic random numbers than default rand())

§1.2 setup visual studio 2005

The libraries and header files neccesary for this codeproject are included. The following 7 steps sets-up visual studio 2005, so it can find these libraries and header files.

1 Copy the unzipped folder glut with all its subfolders to the `c:\program files` directory. Result should be `c:\program files\glut\..`

2 Update default search path in windows environment variables by including
`C:\program files\glut\lib`

3 Restart PC to load search path

4 Copy the unzipped folder `mountainProject` with all its subfolder to any desired location. We installed it at `my documents\mountainProject`

5 Open visual studio. Go to `Tools > Options > expand Projects and Solutions` > expand `VC++ directories`. In VC++ directories "select show directories for" and select include files. Add the following line:
`C:\Program Files\glut\include`

6 In the same VC++ directory go again to `"select show directories for"` and now select `libraries files`. Add the following line:
`C:\Program Files\glut\lib`

7 Go to the `mountainProject` folder and go to `\src\msvc` subfolder. Double click the Visual Sudio solutionfile to start up the project. 

NOTE: The specific dependecies (in configuration `properties>linker>input> additonal dependencies`) are already specified in the project file. They should be `glui32d.lib, glut32.lib, glu32.lib, opengl32.lib, odbc32.lib, odbccp32.lib`

2. User interface

This section discusses the user interface controls. The UI is actually very simple, but complete. The primary input-device is the mouse. There are no right or middle mousebutton event-handlers and keyboard input is also limited to numbers only (to modify spinner-control values). The program closes by clicking on the X button in the top right corner. In Fig. 2-1 the available controls are pictured.

Fig. 2-1 *Available user interface control*

§2.1 Controls

The controls are grouped in 3 columns: Viewing controls, Model controls and the Mountain Generator part.

Viewing Controls

Zoom in/out	Closes in/out on the model

Model Controls

Auto rotate	When checked, the model rotates 360° and scales up/down by 20%
Wire/Surface body	When checked, the model has a texture on its surface. When unchecked, a wireframe is shown.
Manual rotate	By spinning the sphere (mouse dragging) the model rotates. The model rotates depending on how fast the sphere was spinned. If the CTRL button is held down the model only rotates horizontally. When the ALT button is held down the model only rotates vertically.

Mountain Generator

Step1: Choose model

Choose a model — There are three basic shapes to select:

1. One standing triangle
2. Piramid (four triangles)
3. Landscape (composed of two flat triangles)

Step2: Modify parameters

MAX Depth — The level of iterations and thus detail of the mountain or landscape.

The amount of triangles grow by 4^{level}. The piramid on level 5 has the same amount of triangles as the triangle on level 6. Computation gets hard on modern workstation computers beyond level 6.

Dampening — The dampeningfactor determines how the random variation behaves in relation to level. The dampening formula is $1/(dampening^{level})$. This means that the larger the dampening factor the faster the random-values approach zero. This results in a mountain/landscape that has large variation in the first iterations and almost none in higher. When selecting a dampening factor that is below 1 the variation increases when the level gets higher. This results in a more or less flat surface with more detailed (smaller) peaks. When a dampening factor below 2 is selected, chances are high that unrealistic mountains are generated.

MAX amplitude — The amplitude factor determines how the random varation behaves in terms of the magnitude of the variation. The random range (a pool of possible random numbers) is between minus amplitude and plus amplitude.

NOTE: the formula for the random variation consists of the following terms:

```
random_pool * dampeningformula
```

where random_pool is a random number between [–max_amplitue and +max_amplitude].

Example:

level =4; MAX amplitude = 5; dampening factor = 2; Then formula is

$$[\text{a random number between } -5 \text{ and } 5] \; * \; 1/2^4$$

Resulting in a number between -0,3125 and 0,3125. This shows that there is almost no variation in the 4th iteration.

3. Fractal algorithm

This section discusses the Fournier fractal mountain algorithm we implemented. Implementing the fractal algorithm (combined with lighting), was the hardest part of the assignment. We researched two mountain fractal algorithms; the *Fournier subdivision algorithm* and the *simplified algorithm* [**1**]. We prefered the first, because the results were far more realistic.

§3.1 The Fournier subdivision algorithm

The basic idea of the Fournier algorithm is finding the midpoints of three sides of a triangle and then displace them in *y* direction with a random factor. This results in four more triangles, each which can now be divided in four more triangles again. This proces is recursive untill the maximum depth level is reached. Fig. 3-1 and 3-2 illustrate this.

The first figure shows how the midpoints are randomly displaced in *y* direction, creating four new triangles (see Fig. 3-1).

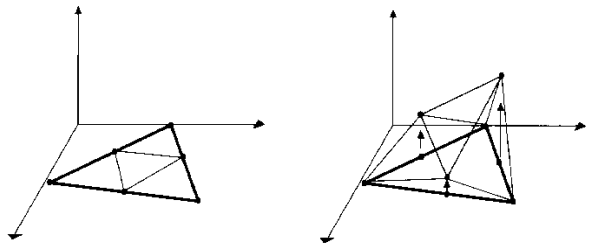
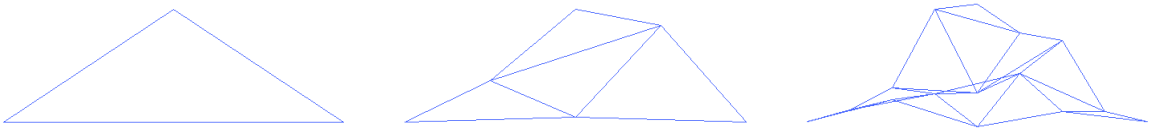


Fig. 3-1 *Midpoints are found and randomly displaced in the y direction*

Figure 3-2 gives a better impression of what the result can be when using this algorithm. Note that the initial and displaced points never move when doing a deeper interation (see Fig. 3-2).



Initial triangle **1st iteration** **2nd iteration**

Fig. 3-2 *Possible results of the first two iterations of the Fournier algorithm*

Although the idea of the algorithm can be written down with just a few sentences, the code to implement it needs significantly more.

The pseudo code of a recursive implementation of the algorithm is written on the next page:

```
/*MyDisplayFunc*/
void myDisplayFunc(void)
{
        drawMountain( 0,point 1, point2, point3);       //initial triangle on level 0 and the start of the recursion
}

/*addRandomValue*/
Point addRandomValue(int level,point p)
{
seed = level + point;                   //this makes sures that the same number is generated for unique level,point pairs
rand(seed);                             //initialize randomizer
randomValue = randomPos_or_Neg * randNr_from_pool * 1/ (dampening^level);
p.y + randomValue;

Return p;
}

/*Recursive Function drawMountain*/
drawMountain(int current_level, point p1, point p2, point p3)
{
        If (current_level == maxLevel)                  //if maxlevel reached draw triangle, stop recursion
        {
                Draw points p1,p2 and p3 ;              //between the point lines can be placed to make a triangle
        }
        Else  /*else find midpoints, add  randomvalues and draw four new triangles */
        {
                Point newPoint 1= addRandomValue(level, findMidpoint(p1,p2));
                Point newPoint 2= addRandomValue(level, findMidpoint(p2,p3));
                Point newPoint 3= addRandomValue(level, findMidpoint(p1,p3));

                current_level = current_level + 1;      //go one level deeper and start recursion four times.

                drawMountain(current_level, p1, newPoint1, newPoint3);          // top triangle
                drawMountain(current_level, newPoint1,p2, newPoint2) ;           // left triangle
                drawMountain(current_level, newPoint3, newPoint2, p3) ;          // right triangle
                drawMountain(current_level, newPoint 2, newPoint3, newPoint1);  // middle triangle
        }
}
```

§3.2 Problems with the Fournier algorithm

With the pseude code ready, implementing would be a piece of cake, not? Well, this was only partially true. The pseude code was implemented fast, but then two problems arose when we wanted to view a mountain on depthlevel 2. The first problem was, that at level 2 and higher, "gaps" appeared between triangles. We weren't the first to come across this problem, Fournier [**1**], mentions this problem calling it an *internal consistency problem*.

The other problem we faced was an unrealistic mountain shape.

§3.2.1 Internal and external consistency problem

This problem, codenamed the "gap problem", proved to be the most timeconsuming. We figured that when the top, left and right triangle displace their midpoints, the middle triangle no longer needs to calculate its midpoints (those are dependent). So far so good, but the problem arises when going to the next iteration. Then the new middle triangle of the middle triangle is dependent on the new triangles of the adjecent triangles. For example, the blue triangles in Fig. 3-3. Because of the recursion, there is no knowledge of the development of the new iterations of adjecent triangles.

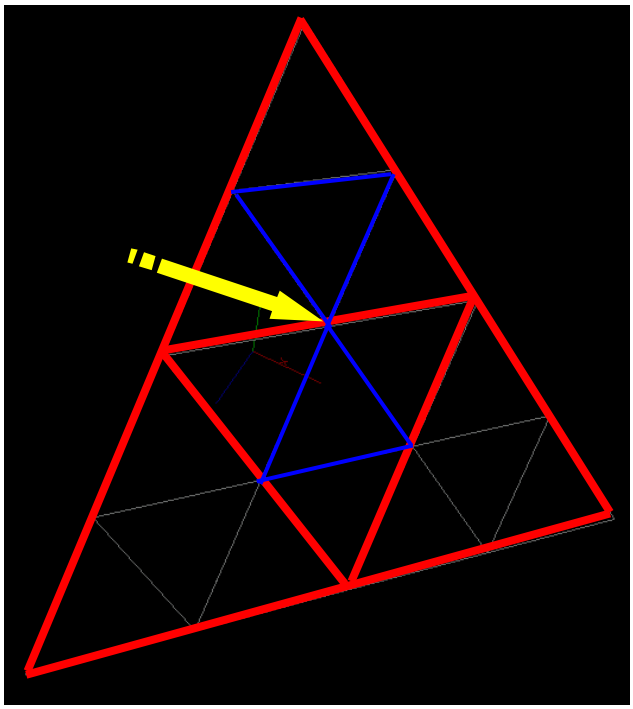


Fig. 3-3 *Example of dependent midpoints*

So for the new middle triangle to "know" what the random displacement of the next iteration of its adjacent triangles are, we created a special random function . This random function is special in the sense that for every unique location-level combination, **the same random value** is found. In this way, triangles "know" what displacements adjacent triangles produce, without actually communicating with them. The advantage of this solution is that it doesn't require the recursion to stop, synchronize by communicating and continue one level, stop the recursion again, etc.

Better yet, with this solution the external consistency problem which involved communicating the direction of displacement to the middle triangle is solved as well.

§3.2.2 Getting realistic shapes

Initially we displaced in *x*, *y* and *z* directions. We weren't satisfied with the results in terms of realistic shapes and changed it to *y* direction only. We figured that generating realistic mountains by displacing in the *x* and *z* direction is natural (it occurs in nature), but it is very hard to define the range and positions where these displacing may occur. In nature, gravity works on the mountain as well as the wind and other weather elements. These factors need to be modelled to produce realistic shapes with displacement in all directions. Nevertheless fractalmountains with only y direction displacements already produce good shapes.

Another shape tune was the use of another random number generator. The Gaussian number generator mentioned by Fournier, could generate an extremely low/high number (to practical infinity) with low probability yielding unrealistc shapes. The default rand() of Visual Studio between -1 and 1 also gave unrealistic results. Based on advise of fellow MKE student S. Panic, we tried a random number generator called Mersenne Twister [**2**], giving us numbers between -1 and 1 with more realistic shapes.

4. S+H Mountain generator features

This section discusses the most important features of our application. It is not a complete list, but it shows that we tried several techniques to improve realism and build user-friendly software at the same time.

§4.1 Improved realism

To improve realism we focused on two elements; lighting and a texture map on the mountain. This results in a mountain with realistic depth and fysical look.

§4.1.1 Lighting/Shade model

The lighting model we used is the default OpenGL lightmodel. It is simple yet provides realistic results. To enable correct (smooth) shading we implemented a function calculating the outerwards pointed normal vector of every vertex. We implemented a "sun" in the scene to indicate where the light originates from. To see the effect of the shading we illustrated two pictures, where we simply rotated the scene. Also notice the difference in brightness of the shading on the mountain. The whiter the surface the more light can reach that surface.

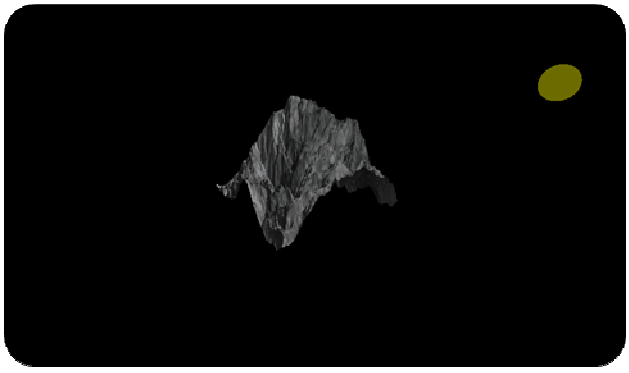


Fig. 4-1 *Front image of the moutain*

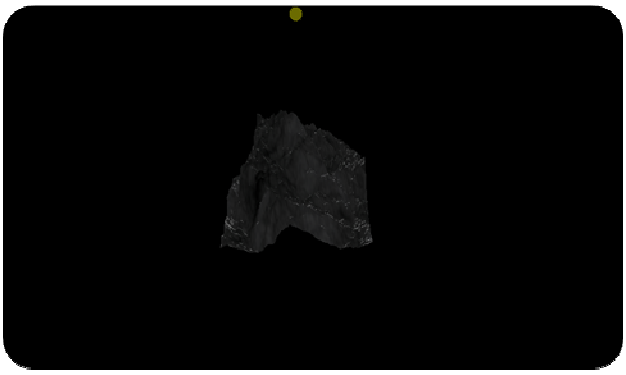


Fig. 4-2 *Back image of the moutain*

§4.1.2 Mountain Texture

While lighting is neccesary for depth realism, a texture gives the mountain "physical" realism. Mountains in general are build from rock, with possibly some cover like earth, water, dense dwellings ,etc. If we neglect the cover and focus on features of rock there is still variation in the pattern and color. We searched the internet for some rock textures (creating a realistic textures ourself is a study on its own) and choose the texture in Fig 4-3. It has the classical rock color and pattern. The texture is "folded" over the mountain by calculating the vertex coordinates in texture coordinates. The result is quite good, especially on a distance, as you can see in Fig. 4-1.

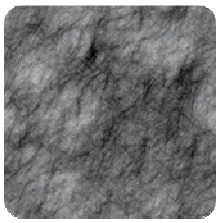


Fig. 4-3 *our mountain texture*

§4.2 Viewing/Model controls and realtime generation

A generator would not be complete without viewing and model controls. Although we didn't exploid every view/model control, we enabled closing in and out on the model,rotate it and give sufficient parameters to significantly change the shape of the mountain/landscape. We used GLUI (an extension of GLUT) for our user interface controls. The controls react realtime (depending on the workstation and model complexity). Each model has its own unique set of parameters, meaning that the same mountain can be generated any time.

§4.3 Flexible framework

The last feature we would like to discuss is our flexible framework. We implemented three basic shapes (triangle, piramid and flat square) and several parameters to modify its shape realtime. The initial shape (defined by a series of points) is the outline of the model. Our software can easily be extended by more basic shapes or even a drawing primitive requiring any multiple of 3 points. Another possible application would be dividing any object in triangles and giving it a rocky surface, again by reusing the *drawmountain* routine.

5. Review and future work

This chapter presents our review on this assignment in the first paragraph and our recommendation for future work in the second.

§5.1 Review

We enjoyed working on this assignment. In these past few weeks we learned that nature can often be mimicked by fractals. A combination of shape, texture and lighting are the basic components to do so. Our final result is satisfying, but like always with software, it can be improved in many ways. The development was a form of rapid prototyping; constant changing of requirements, different design approaches, analyses (discussing the intermediate results) and of course a lot of prototypes. We did not need a lot of sources to succeed; just [**4**],[**5**] and some websources mentioned in the *References*. The assignment was managable given the timeframe although it helped that we worked together before. Finally, the pointers given by S. Busking on our intermediate result, improved our final result greatly.

§5.2 Future work

There are a lot of features still unexplored. Originally, we wanted to make a total scene with trees, moving water and reflections in the water. In OpenGL, reflections are possible using a technique called stenciling [**3**]. Another idea was a procedural texture with a Perlin noise distribution for forming flowerspots. The texture would consist of several layers to simulate a gradient overlap between grass colors depending on the height value of the mountain.

The last unexplored feature we thought of during this project was creating a mipmap so that the mountain has a high resolution texture when zoomed in and low resolution when zoomed out. Of course optimizing code, by for example lowering the amount of computations when doing a redraw with no model changes can also be considered a feature ☺

This project has a lot of interesting possiblities, but development would be purely for developing OpenGL skills. Already, a lot of sophisticated mountain- or even complete scene-generators are available.

References

Webreferences

1) http://www.cs.wpi.edu/~matt/courses/cs563/talks/frac_mnt.html

2) http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html

3) http://www.opengl.org/resources/code/samples/glut_examples/examples/examples.html

Books

4) Hearn,D., Baker, M., (2004). Computer Graphics with Open GL. *3th edition international version, Pearson Education.*

Reader

5) Nieuwenhuizen, P.R, et al. (2007). Computer Graphics Lecture notes. TU Delft reader course in2905-I.