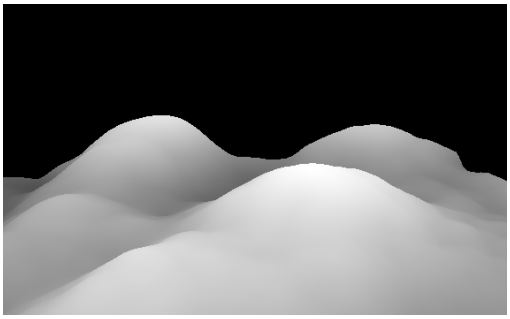
COMP SCI 3GC3: Computer Graphics
Assignment 2

Due: Monday November 10, 2015 at 12:00 noon.
Accepted Late until Saturday, Nov. 15, 2015, 12:00pm, at a 20% per day penalty.
This assignment is worth 8% of your final grade.

Terrain Modeling



Write a graphics program using C/C++ and OpenGL to produce and display an interactive terrain mesh. Terrain meshes are models typically comprised of either triangle or quad polygons that make up a 3D surface. They are often used to represent outdoor virtual environments (e.g., fields, hills, mountains, etc.). An example terrain is shown to the right. For the purpose of your assignment, you will need to represent the terrain as a heightmap (a 2D array of height values), procedurally produce height values according to a specified algorithm, and display the resulting terrain using either triangle or quad (strip) primitives. See below for details on each point.

When "make" is called, a program called Terrain.x should be compiled and generated.

Representation

As discussed in class, terrains can be represented as a heightmap, a 2D array of height values. Then, every 2x2 sub-array of this heightmap represents the four corners of one quad (or the four corners of two triangles with a shared edge). If the height direction is represented by the y axis, then the x and z directions can be used as indices into the array for the y height value for each xz vertex. The size of the grid should be selectable by the user. Your program should support a terrain of sizes ranging from 50x50 to 300x300 vertices (you can also go higher if you want, but bear in mind the larger the terrain, the more computationally intensive it will be to render).

Producing height values

Initial height values should be generated using the so-called "circles" algorithm. The idea of this algorithm is to randomly choose many points in the initially flat terrain. Then, all vertices within a fixed size circle around these randomly chosen points are perturbed upward (or downward) to produce a round bump. Points closer to the centre of the circle are higher than those at the edge. After numerous such circles are created, eventually the terrain looks like that depicted above. Note that the number of iterations will need to be higher for larger surfaces, otherwise, the terrain will look sparse. Details of this algorithm can be found here: http://www.lighthouse3d.com/opengl/terrain/index.php?circles

Rendering the terrain

Once the heightmap is loaded with values, the terrain needs to be actually rendered. You can use triangles or quads for this (at your discretion). You can also use triangle or quad strips – these primitives may make rendering the terrain easier. To render the terrain, you need simply index the heightmap for every xz vertex, and use the corresponding y value. To colour the terrain, you can initially use grayscale colours (as shown above) corresponding to the height of the vertex. Assuming there is a maximum and minimum height in your terrain, treat the lowest height as a value of 0 and the highest height as a value of 1. Then simply use the height value as the RGB colour value for each vertex. For lighting purposes, you can use a material colour of your choice.

Other Necessary Functionality

- Interactive viewpoint control
  Pressing the arrow keys should rotate the viewpoint. The left and right arrow keys should rotate the scene about the y axis, while the up and down arrow keys should rotate about the x axis (to some limit to prevent flipping the camera upside down).
- Wireframe representation
  Pressing the 'w' key should toggle wireframe mode between three options. The first (and default) setting is solid polygons (shown above). The second setting should display only the wireframe version of the terrain. The third setting should display both the solid polygons and the wireframe. You can use glPolygonMode to toggle the rendering settings for polygons. (Note that the third option will likely require you to render the entire scene twice!).
- Lighting
  Implement lighting in the scene, which can be toggled with the L key. Note that this means you will need to compute surface normals for each face of the terrain (Hint: Write a function to do this once immediately following creation of the terrain). You should support two point light sources, initially positioned above opposite corners of the terrain. The colour of the lights is up to you. The position of the lights should be user controllable (i.e., by pressing keys).
- Shading
  Provide the option to toggle between flat shading and Gouraud shading.
- Reset – "R" key
  Pressing the "r" key should generate a new random terrain using the heightmap generation algorithm.
- Graphics Features
  You should use backface culling and double buffering. The program should run in a window (i.e., not full screen).

Extra Features

Implement two of the following extra features. Implement a third feature for bonus marks.

1. Improved Camera
   Improve viewpoint control by adding a fully interactive camera. This should allow you to move the camera viewpoint to arbitrary points in the scene, and rotate it around its internal xyz axis.
2. Additional terrain algorithm(s)
   Add support for an additional terrain generation algorithm. Examples include the fault algorithm and the midpoint displacement algorithm. Note that you can select this option twice, i.e., for both of your extra features. See http://www.lighthouse3d.com/opengl/terrain/index.php for details of these algorithms. You can also create your own terrain algorithm.
3. Display 2D terrain overview
   Display a second GLUT window, and draw a 2D overhead representation of the terrain in this window. For example, if your terrain is 300x300 vertices, this 2D view would be 300x300 pixels in size. Each pixel in the view would be coloured according to the grayscale colours described above, i.e., using the height for the colour value.
4. Terrain modification
   Allow users to interactively modify the terrain once it is created. Note that this will most likely be easier if you also implement Extra Feature #3 (2D terrain overview) above. By left clicking the mouse on the terrain (or in the 2D overview, if applicable) the terrain should bump upward, i.e., add a new circle according to the circles algorithm. Right clicking should perform the inverse operation, i.e., add a "dent" (push the terrain downward).

5. Add a character
   Add a character (or characters) to the scene. These may be controlled by the user, or may move around according to a random walk algorithm. The character(s) should appear to move around on the terrain, i.e., their height should be determined by the height of the terrain at the point they are moving across. One idea here would be to combine the snowman tutorial with your assignment, and have a number of randomly moving snowmen moving around on the terrain. Note that the character need not be animated, i.e., simply sliding across the terrain surface is fine!
6. Load heightmaps from image files
   Since the heightmap is simply a 2D array of height values, it is possible to load grayscale images and use each pixel colour as a height value (again, according to the colour algorithm given earlier). For these purposes, you may use an image library, or image loading code. Be sure to cite your sources to avoid academic misconduct charges. We will also discuss the loading of a specific image file format (the PPM file) later, but this may not be soon enough.
7. Improved map colouring
   Colour your terrain according to topographic maps. The lowest parts should be shades of green, and higher parts should become progressively more orange/red. The highest points (mountains) should be gray. See here for details: http://en.wikipedia.org/wiki/Hypsometric_tints.
8. Your own groovy feature
   Implement something along the lines of items 1 – 7 listed above of your own design. Be sure to document any references you use, and clearly indicate what this feature does.

Submission Notes

All programs must initially write to the DOS/Unix shell a list of commands (keyboard or menu items) and their uses in the program. The marker should not have to look at your source code to figure out how to run a program. Marks will be deducted otherwise!

You have the option of implementing your assignment on your platform of choice, BUT please make sure your programs can be fully compiled and executed on the departmental machines (e.g., ITB 237). Also be sure to include any additional libraries that you use. If the TA has to hunt for a specific library to compile your code, you will lose marks.

Submit your source code, makefile, readme, and any other resources to A2 folder of your SVN trunk. All source code and makefile should be located in A2 and not a subfolder. When 'make' is called, it should produce programs named: "Terrain.x".

Familiarize yourself with the department's policy on plagiarism and the university regulations on plagiarism and academic misconduct. Plagiarism will not be tolerated, and will be dealt with harshly.
THIS ASSIGNMENT IS INDIVIDUAL WORK.

Have fun, and be creative!