CPS511 Assignment 2: Interactive Terrain Generation

Due Date: Friday, Nov. 4 at 11:59pm

You will implement an OpenGL-based 3D Interactive Terrain Generation program. This programming assignment will increase your knowledge of 3D modeling, interactivity using 3D picking (option), camera control, surface lighting and shading, and polygonal meshes. **You must do this assignment alone – no groups.** Begin designing and programming early! This project is worth 15 percent of your mark. **If there is some part of the assignment you do not understand, please see me (or email me) as soon as possible and I will clarify the issue.**

Figure 1

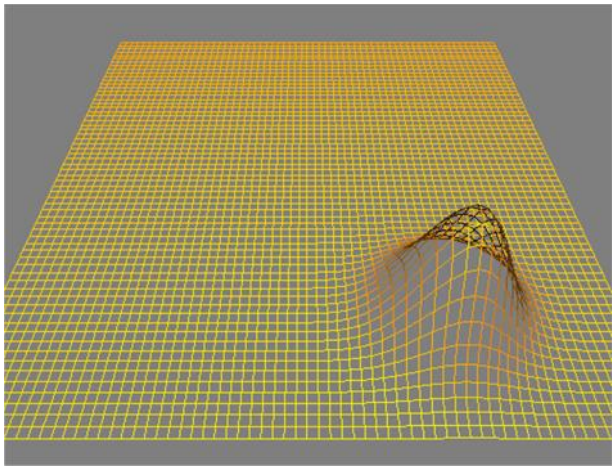
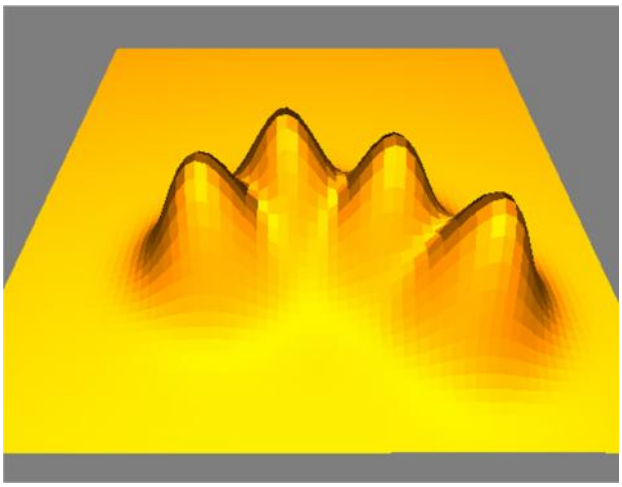


Figure 2



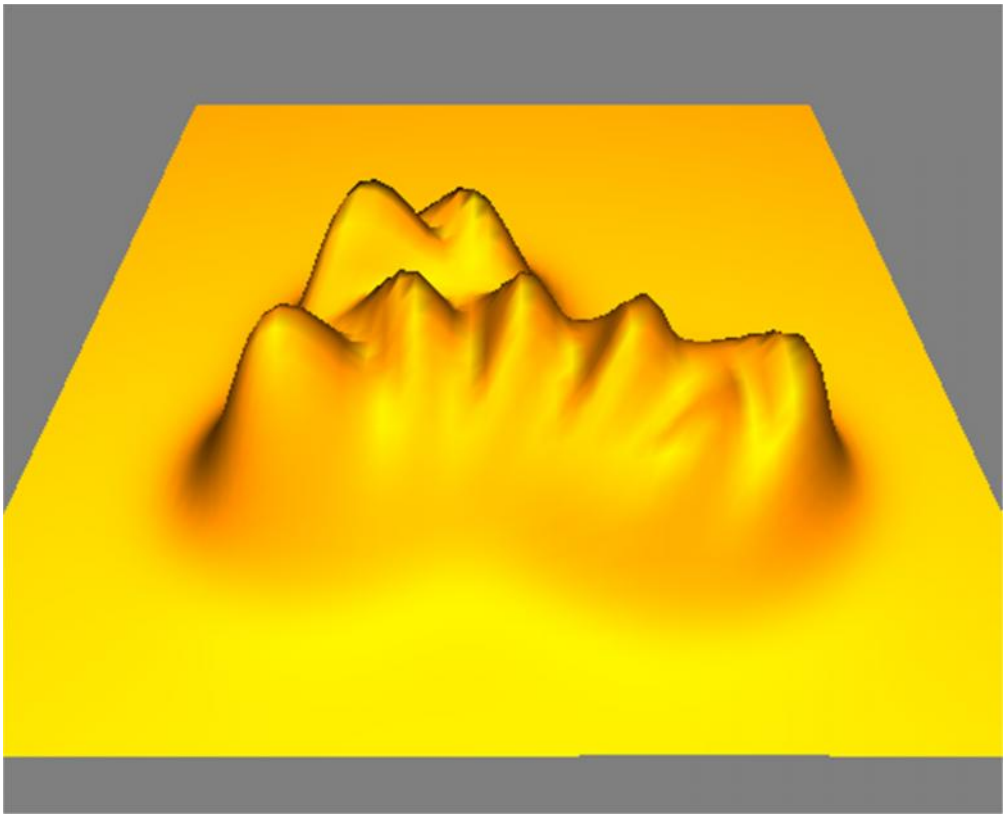


Figure 3

Terrain Generation

This goal of this simple yet useful program is to allow a user to create complex smooth hills and valleys (a terrain) on a mesh of quadrilaterals (Figure 1). You will use a technique known as blobby objects or *metaballs*. Figure 1 shows a 64X64 mesh of quads with a single Gaussian blob. Figure 2 shows another example using flat shading (you can see the quads). Figure 3 shows an example terrain generated by positioning several Gaussian blobs and using smooth shading.

Program Requirements:

- The user should be able to use the mouse (or the arrow keys) and move a blob around on a mesh of quadrilaterals. When the user releases the mouse button (or hits a key like F1 if you are using a key based interface) then the blob is deposited and the mesh is smoothly deformed to create a hill or mountain. The user can then reposition the mouse and create another blob – which will be smoothly blended with any previously deposited blobs.
- The user should be able to change the **height** and **width** of the currently active blob interactively (through a menu, keys, sliders etc.)
- The mesh should be rendered smoothly using proper lighting and shading. This means you will need to compute normal vectors at each mesh vertex as the mesh is deformed by positioning a blob. This has been done for you in the quad mesh code provided in assignment 1. **Alter this code to have the option of using a single normal vector per quad rather than per vertex**. Use the **F2** function key to toggle back and forth between the two and observe the difference. Some notes have been provided for you to read about normal vector calculation.
- After you have finished computing the heights using the blobs, add a random displacement to make it appear less smooth and more natural.

- **Camera Control:**

    a. ***Mouse-controlled world-view navigation***: Imagine your scene is surrounded by an imaginary hemisphere. Use the mouse to slide the camera along this hemisphere. For simplicity, the camera can always be looking at the origin of your world. Left/right motion of the mouse controls the “azimuth” (i.e. imagine this type of control as similar to moving along a line of latitude around the earth). Up/down mouse movement controls the “elevation” of the camera (i.e. imagine this type of control as similar to moving along a line of longitude of the earth – towards or away from the north pole.) You are permitted to restrict the range of elevation so that the camera never drops below the terrain of your world. Also provide the ability to zoom in and out of your world. Use the **F1** function key to switch to camera control if you are using the mouse to control blobs.

Hints (READ THESE!)

- Begin by creating a mesh of quadrilaterals. Use the quad mesh code handed out in the first assignment. The **y** coordinate is used as the height above the mesh.

- To create a blob, you may use many smooth functions. One simple but effective function to use is a Gaussian. In the equation below, *k* Gaussian blobs are added together to determine the height (**y** value) of a vertex in the terrain mesh. The influence of a blob on the height of a vertex depends on how close the blob is to the vertex. The **r** variable is the distance from a mesh vertex to the *k*th blob. The **b** variable controls the height of a blob. If **b** is negative, you can create a valley. The **a** variable controls the width of a blob.

$$f(x,z) = \sum_k b_k e^{-a_k r_k^2}$$

- When a blob is created or moved, you need to update your mesh. You loop over each vertex in the mesh. For each vertex, you compute the distance from that vertex to a blob – this is the **r** value. You plug in the **r** value (along with the **a**,**b**, values for the blob) and determine the height (f(x,z) see equation above) and add it to the height (**y** value) for this vertex. You repeat this for each blob, adding its contribution to the current vertex height. You then proceed to the next vertex and repeat. Do not forget to reset the height of each vertex to zero before updating it.

- You may use keys to create a new blob and move the current blob around the mesh. You may also want to use

the mouse to control blob creation and movement (extra marks). To do this you need to implement picking in OpenGL in order to pick a position on the mesh. There are several ways to implement picking. I have posted a couple of documents to help you. The simplest technique is to convert the mouse screen position to world coordinates by reading a z value from the depth buffer and using gluUnProject(). Another technique is to implement ray casting in eye/camera space.

Bonus (1 mark)

1) Render mesh using a VBO and implement the mesh displacement via meta balls in a vertex shader (0.5 marks)
2) Implement Phong lighting+shading in a fragment shader (0.5 marks)

Grading (Out of 15 marks)

Mesh of quads properly constructed and rendered using either vertex normals or quad normals (F2 key)	1 marks
User can use keys to create and interactively position a blob. Multiple blobs blend together properly.	7 marks
User can use mouse and picking to create and interactively position a blob. Multiple blobs blend together properly.	9 marks
The height and width of the current blob can be interactively controlled by the user (using keys/mouse/sliders).	2 marks
User can use mouse to move camera over surface of imaginary sphere surrounding scene (F1 key to switch to	2 marks

camera mode). User can zoom in and out.	
Program structure, style, comments etc.	1 mark
Bonus	1 points
Total Possible	16 points

Program Submission

Use D2L to submit your assignment. Submit all your source files. You may use C, C++, or java for your program. Zip everything up into one file. **Do not include executable files.** If your program runs under Windows, include a README file describing how to compile your program. If you want to inform the TA about your program (special features, bonus work etc) include this information in program comments and the README file. Include all makefiles (for Linux) or project files (for example, if you used Visual Studio). **It is your responsibility to ensure the TA has enough information so that she can, with little effort, compile and run your program on a Windows environment**. I am being flexible in terms of your programming language and operating systems choice so you must make an effort to meet me halfway. If the TA has trouble compiling your program, she will have the discretion to deduct marks and/or she will ask you to compile and run your program in his presence.