CPSC 314, Project 3: Bumpy Terrain

Out: Thu 26 May 2005
Due: Fri 1 June 2005 11:59pm PST
Value: 9% of final grade
Points: 100

In this assignment you will implement bumpy, shaded terrain. You can fly around it using the navigation controls from the previous project.

- **Variable Height (15 pts):** You need to make a bumpy ground plane, where the color at each vertex is random and the height of each vertex in the plane varies randomly. Specifically, the variance in the height should be +/- 20% of the width of the face. (If your faces are not equal in length and width, use whichever is bigger.) As in project 2, your plane should lie in the xz plane, with the bumps in the y direction. Your terrain should be a grid of $100x100$ vertices in the xz plane, with the corners of the terrain at (0,0,0), (100,0,0), (100,0,-100), and (0,0,-100).
- **Per-face Normals (20 pts):** Calculate and use the normals at each face, by finding the unit vector perpendicular to the face. Toggle switching between per-face normals and per-vertex normals as below with the 'v' key.
- **Per-vertex Normals (25 pts):** Calculate and use per-vertex normals that average the normals of the faces that surround each vertex.
- **Normal Drawing (10 pts):** Drawing the normals that you're using for the lighting calculations as geometric objects in the scene: short lines radiating from each vertex. Note that you will see multiple normals emanating from each vertex in per-face normal mode, and only one in per-vertex normal mode. Use the 'n' key to switch normal drawing on and off.
- **Lighting (15 pts):** You should have one light in the world coordinate system, placed at the point (80.0, 10.0, -80.0). You should also specify one headlamp in the camera coordinate system, at the origin, that moves along with your point of view. It should be a spotlight pointed down and to the front of the current viewpoint. Your headlamp should be able to be turned on and off using 'h' as a keyboard toggle switch. You should specify light positions exactly and only when necessary.
- **Shading (5 pts):** You should support switching between a flat shading model, where the shading calculation is done only once per face, and the smooth shading model, where the shading is calculated at each vertex, using the `glShadeModel` command. Use the 'f' key as a toggle. Notice the interplay between the shading model and the normal calculations.
- **Regenerate Terrain (5 pts):** Use the 'u' key to replace your terrain with a new set of randomly generated geometry. You will need to regenerate heights, colors, and normals.
- **Toggle Terrain Colors (5 pts):** Use the 't' key to toggle between grey and the random colors. You will find it much easier to understand whether your shading and lighting is correct when your terrain is uniformly colored.

Suggested Strategy

- The easiest way to make a bumpy plane is to compute vertex locations on a flat plane in the x and z directions, then perturb the height of your vertices in the y direction.
- A rectangular region where each vertex can have a different height may not be planar. Triangles are always planar. If you have a rectangular grid of vertices, for each rectangular region you can send two triangular faces to the OpenGL pipeline, and these two faces will be planar with a crease in between them on the diagonal of the rectangle.
- To compute the normal vector for a face: construct two vectors from the points that determine the face, take their cross product, and normalize that vector to be unit length. Remember to traverse the vertices for a face counter-clockwise, according to the OpenGL convention, both for computing these normals and when sending the geometric data to OpenGL.
- First compute per-face normals, then use those to compute your per-vertex normals. To compute the normal vector for each vertex: interpolate between the normal vectors for all the faces that share a vertex. Compute normals during initialization of a new terrain, then send those precomputed normals to OpenGL on every frame.
- Note that normal drawing can help you debug your normal calculations, so you consider implementing that functionality sooner rather than later.

- Since you will need to interpolate per-face normals to create per-vertex normals, you will need an easy way to find all the faces around a particular vertex. I recommend storing your vertices in a $100x100x4$ array, so that you have a homogeneous 4-tuple at each grid point. You can traverse this data structure to first do your vertex position calculations, then your per-face normal calculations, then your per-vertex normal calculations, and finally to send the normals along with your geometric data to OpenGL on every frame.
- Consider what kind of a data structures you need to store the normals for each face, and to store the colors for each vertex.
- When using per-face normals, specify just one normal for each face, and OpenGL will use it for all the vertices in the face. For per-vertex normals, send a new normal before each vertex.

Handin/Grading/Documentation

The grading, required documentation, and handin will be the same as with project 2, except use the command 'handin cs314 proj3'.