CS195V Project 2 - Warp

February 23, 2012

1 Overview

In this project you will use some noise generation techniques to create some cool effects, then map them to some sort of 3D tessellated geometry. You can generate your noise in real time using the GPU and use tessellation shaders to create cool geometric effects.

This project is due 11:59PM on Tuesday, February 28.

To get the support code for the project, copy the gpusupport directory again form /course/cs195v. Note that the support code has changed a bit since the first project. You can certainly implement this assignment in the older version, but you may find the new version easier to use. If you are overwriting your old support code, you may want to back up the old version just in case.

As always, run "cmake ." in the support code directory after you have copied it to set up your build chain, and set up the run settings in QtCreator to point to the correct targets. You can find a new version of QtCreator in /contrib/projects/Qt/QtCreator/bin (this is version 4.8 which adds some new features you may find useful).

2 Requirements

For the simulation step, you need to generate some kind of noise in at least 2D. You cannot use purely random white noise generation or the noise*() functions in GLSL as your final noise function. You can, however, use them as building blocks to generate a more complex noise function. You must have a way to display your noise to the screen (i.e. draw it as a full screen quad with brightness representing value).

For 3D, you must use a tessellation shader to add vertices to a simple primitive and then use your noise function to do cool things to the primitive (displacement mapping, etc). In determining your level of detail, you must use some sort of adaptive algorithm (see the lecture slides for some examples).

In determining your object color, you must implement some kind of non-trivial material shader (it can't just be one solid color). Phong shading or something comparable is fine.

You must also implement at least one post processing shader. SSAO, motion blur, FXAA, bloom, HDR are some examples.

As always, avoid using old/deprecated OpenGL features and ask us if you have any questions.

3 Technique

For displaying your noise function, you can just draw using a specialized fragment shader. If you are using 2D noise, you can save the results of this shader to a framebuffer to use later. If you are using a higher dimension noise function for your 3D step, just show us two dimensions of your noise function in 2D mode so that we know what kind of values your noise function generates.

4 Instructions

Most of the work for this project will be in writing your tessellation and post processing shaders. You may need to write some additional 3D primitives for your visualization. We do not expect you to derive your noise function from scratch; feel free to copy implementations found online.

5 Tips

When drawing a primitive for use in the tessellation shader, *make sure that the draw mode is GL_PATCHES.* Otherwise, you will probably not see anything. Also make sure you define the patch size to whatever primitive size you are tessellating (3 is the default).

6 Helpful Links

- Tessellating Triangles(includes Icosahedron primitive)
- Tessellating Quads
- Displacement Mapping a Sphere
- GLSL Simplex Noise Code
- Marching Cubes