CS 559: Spring 2013, Project 3

Nathan Deisinger (deisinge, [ndeisigner@wisc.edu](mailto:ndeisigner@wisc.edu))

Nick Heindl (heindl, [nheindl@wisc.edu](mailto:nheindl@wisc.edu))

README:

Instructions:

C – Toggle bird’s eye view

W – Toggle wireframe

X – Close program

P – Pause

S – Toggle shadows (only on first shader)

A – Toggle axes

N – Toggle normal

F2 – Toggle skydome

F3 – Toggle shader

Bonus Features:

Load Screen

A simple load screen implemented via a call to FreeGLUT’s text functions before initializing. Not very complex, but since we generate noise ourselves at startup, it looks much better to have this than have the program apparently hang for 10-20 seconds.

Gooch Shading

A somewhat cartoony shader that uses a ‘warm color’ and ‘cool color’ and the angle between the light source and the surface to determine how we blend the two. The shader also makes use of thick black borders around objects, which are generated via careful use of the art of Chi Ting – we render the back faces in wireframe mode with a thick border before performing the actual render pass.

“Fire” Shader on Spheres

A simple shader that uses Perlin noise. The noise is generated at initialization through the use of the libnoise library. The “fire” shader itself uses various octaves of noise and time to create a varying effect on the surface of the sphere, as well as uses the noise to slightly vary the vertex coordinates of the sphere.

Noise Shader on Floor/Spheres

A “let’s-see-what-happens-if-we-do-X” shader. We sample the noise and use that value to adjust the normal for the surface, creating a pulsating, glowing effect.

Dynamic Shadows

When using basic textures and Phong shading, shadows can be toggled by pressing ‘s’. When activated, we perform an extra rendering pass from the light’s point of view into an FBO before rendering the actual scene. Upon rendering the scene, we use a special matrix to translate points in the scene back into the coordinate space of the light’s POV, and then use the depth value stored in the FBO to determine if the point in question is in shadow. If so, we only use the ambient light component; otherwise, we do Phong.

Problems we had:

Getting Nick’s computer to initially work was tough. It turns out that how shaders were implemented initially (passing a struct as a uniform block) was the problem, even though Nate’s and the lab’s computers could run the program. The latest driver on his Nvidia card caused this problem. It was found by debugging like mad, until it was found that the shader was getting garbage data.

Conversely, attempts to use subroutines in the shaders were stymied by Nate’s graphics card, which despite allegedly having support for the feature (ARB\_shader\_subroutine) did not correctly handle subroutines. This would have been made use of in dynamic shadows – in the final version, a simple Boolean is used instead.

Implementing Box2D was troublesome at first. We kept getting very odd orbiting effects. We pinpointed the problem down, and it happened to be that even though the physical and graphical objects lined up, the physical sphere (created first), was created at some offset to the origin within Box2D.

Freetype was attempted to be implemented late in the game, due to a misinterpretation of the project requirements. In the end, we could not get the version of Freetype we were using (code by NeHe productions) to work with our project once we began actually rendering the game world; thus, text in-game is rendered using FreeGLUT’s text functions.

Libraries we used: freeglut, glm, devil, libnoise, box2d, and an attempt at freetype