Dustin Greene (d.greene)

CS 529 Final Project

12/3/2014

3D Asteroids

Sections:

* Project Information
* Bonus
* Requirements
* Final Grade: **99** and calculation is shown believe in the last section

**Information**

This project is a 3D version of asteroids. You start with 2 lives and each time you collide with either a planet or asteroid, you lose a life. The goal of the game is to destroy all the asteroids. There are two screens, one for the win condition and one for the lose condition.

In order to open in Visual Studio, you will need to have the correct version of CUDA installed. I am targeting CUDA 6.5 and VS will not open without that installed. I have also included the Debug folder with the executable, Resource files and DLL’s so you should be able to run the application by running the .exe. Any questions, please email me. I will be around Monday and can respond via email after that. Currently builds in Debug mode, runs within Visual Studio and runs from outside Visual Studio by running the executable.

Controls:

* W key fly’s ship forward as long as pressed. If you want more control for testing, uncomment the code within GameEngine::UpdateScene() to get keys A,S,D,Q,E for additional movement.
* Left mouse click fires laser bullets.
* O cycles between several post processing effects.

1. Standard Lighting with textures. No post process effects
2. FXAA: edges will be blurred.
3. Posturized effect
4. Predator vision mode
5. Dream vision
6. Noise
7. Scan lines Chromatic

* N turns debug draw on/off
* M turns menu/controls screen on off

To stop playing or to exit the game, simply hit the ‘Esc’ key.

**Bonus Points**

* Object management +2: Objects in the game are stored and managed in several containers, one static and the rest are dynamic. When an object is destroyed it is flagged, released and removed from container. Objects are maintained in the World class.
* Resource management +2: I have special container classes for Textures, Meshes and Shaders. Simple interface for using and retrieving these resources is very useful.
* Creation of objects from data files +2: Static objects such as the planet objects are defined in their respective .txt file. This file stores information such as relative size and orbit position. There is also a simple level file that stores a list of all the object files that need to be read and loaded into the game during initialization. Most of this code is within the Planet class.
* Game is fun to play and interesting for a few minutes +4: I believe that my interpretation of 3D asteroids is fun and engaging and has some cool post process effects.
* Game is in 3D with a first person style camera +4: Going from 2D to 3D brings with it additional factors and difficulty and I believe that by doing a fully 3D game, the point bonus is reasonable.
* Modern OpenGL with Shaders: +3: All the shaders are located in the Shaders folder. All the code for handling shaders is located in the Shader class.
* Shader and Uniform classes for managing Shaders +2: This makes using individual shaders much easier and cleaner. You don’t have to worry what kind of uniform you need to update the shader has a uniform class that makes sure the correct uniforms are added and updated as needed. Located in Shader and Uniform classes.
* Lighting +2: Located within the basic.vert and basic.frag shaders. Due to the fact that this game takes place in space, I do not add a constant ambient amount and there is no specular component as well. The diffuse reflectivity material color is taken from the texture of the planet, which makes sense.
* Texture mapping with my own Texture class +2:
* Native Win32 program, not using any windowing API +2: I felt this was appropriate because I am not using SDL or any windowing API, I am handling everything myself with all my own callbacks and handling things like window creation manually. This also reduces overhead since I don’t have to call through any API like SDL. Most of this code is in the Engine class.
* Camera with mouse movement +3: Camera is an FPS style camera allowing the user to fly anywhere. It also allows for mouse movement just like an FPS game. This code is within the Camera class.
* Cuda +3: Located within the CudaMesh file, we can create meshes of varying sizes. For example, I can create Spherical meshes of higher and higher detail. Attributes like position, normal and texture coordinates are all calculated on the GPU and stored within Vertex Buffer Objects.
* Cuda/OpenGL interoperability +2: Instead of transferring data back and forth between CPU and GPU, this technique allows CUDA to compute data on the GPU and store the data locally within Vertex Buffer Objects, thus eliminating the need to transfer this data to the GPU. It is calculated on the GPU where the data is needed! This code is with the Mesh files and kernel.cu file.
* Post processing using frame buffer objects and subroutines +2: By hitting ‘O’ on the keyboard, the game will cycle through several post process effects. This code is located in FrameBufferObject and PostProcess classes.
* FXAA +2: While not perfect and it does need some tweaking, it does help with aliasing. The effect is subtle but noticeable but does blur a little too much, but I think it mostly works. This code is one of the subroutines within postprocess.frag.
* Runs at 60 fps instead of the minimum of 30 fps +1: While I believe there is a lot of room for optimization, by using Index Buffer Objects and other techniques helps that game run more efficiently. The game can even have 500 asteroids within the game and it still runs at 60 fps on my system!
* Debug console +1: Used to display output from the program such as shaders failing to compile and other error messages printed to the console. Only in debug mode. Had to set this up manually since in Win32 you don’t get a console.

**Game Requirements**

* Player is in control of a ship in first person mode.
* Game runs at 60fps
* Two different types of enemies: Asteroids and Planets
* Two different types of weapons/power ups: Single laser/bullet shot and power up for a more powerful double laser shot. Red and blue lasers. Similar to Star Fox power ups.
* Collision between player, bullets, asteroids and planets
* Controls screen: Hit ‘M’ on keyboard for the controls screen
* Win condition: destroy all the asteroids
* Lose condition: Get hit by objects twice. Only have two lives.
* Written in C++

**Final Project Grade**

**Grade: 60 - 0 + 2 + 2 + 2 + 4 + 4 + 3 + 2 + 2 + 2 + 2 + 3 + 3 + 2 + 2 + 2+ 1 + 1 = 99.**

I have implemented every requirement listed in the syllabus:

* Debug drawing: ‘N’ button turns this on off. Since I am using spheres, the sphere data is used as its bound volume.
* Basic C++ objects
* Basic Events: New events are added upon collision detection and handled at the end of the update by calling each events update method. Events are deleted every frame.
* Basic text serialization:
* Hardware accelerated (OpenGL)
* Basic transformations: scale, rotate and translate
* Basic collision detection between circles (spheres in my case)
* Collision response: Each object has a collide method that is called. Its response is based on what type of object it is and what type of object it is colliding with.
* Player control of ship
* Runs at 30fps minimum
* Two types of enemies: asteroids and planets
* Two types of weapons or power ups: Power up located within the game world
* Controls Screen: hit ‘M’ on keyboard to show controls info
* Win lose condition
* Written in C++
* Game is fun interesting
* Runs on lab machines\*: talked with you about this issue with CUDA SDK/toolset needing to be installed in order to open in Visual Studio.