CSC 305 Assignment 2 – Real-time Renderer

Due Thursday June 16th, 2016

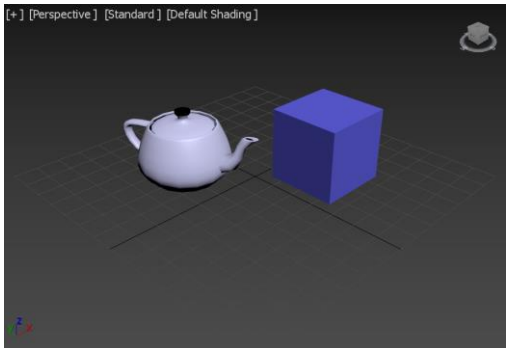


Figure 1: A view of a 3D scene in Autodesk 3ds Max 2017, using some basic shading

Figure 2: An advanced lighting technique demonstrated on the Crytek Sponza scene (by "Synce")

Overview

In the first assignment, you were asked to render a basic scene using CPU ray tracing. In contrast, this assignment asks you to render a basic scene using GPU rasterization. Like the first assignment, there are basic and advanced requirements. Feel free to suggest your own advanced requirements. Project submission and demo will be done like assignment 1 (submit a zip file with a README, then 1-1 demo).

Basic Requirements (70%)

- Render a cube.
- Implement basic camera controls using the mouse and/or keyboard.
- Transform the cube to clip space using a ModelViewProjection matrix in the vertex shader.
- Shade the cube with a single point light using Phong shading in the pixel shader.

Advanced Requirements (30% - pick three)

- Render a floor grid similar to the one in Figure 1 using line primitives. (10%)
- Render a second cube that orbits around the first. (10%)
- Render the cube with a texture. (10%)
- Implement loading and rendering of a Wavefront obj model from a file. (10%)
- Render a mirror under the cube. (10%) (See “Depth and stencils” on https://open.gl/)

The Sponza Challenge

If you’ve done this stuff before and already rendered your share of cubes, I challenge you to download the Sponza scene (Figure 2) from http://graphics.cs.williams.edu/data/meshes.xml and implement as many rendering features as you can. You should still have camera controls, Phong shading (or better), and textures.

Technical Requirements

- You must use the C or C++ programming language.
- You may use OpenGL 3 or OpenGL 4 only if the "Core Profile" flag is set.
- You may use DirectX 11 instead of OpenGL. Recommended if interested in the game industry.
- You may work on your own computer; your code doesn't have to work on the lab computers.

Third-Party Libraries

- OpenGL users may use SDL, GLFW, or SFML to create a window and handle user input.
- OpenGL users may use GLEW to access modern OpenGL functions.
- DirectX users must use the Windows API and DXGI to create a window and handle user input.
- You may use SOIL (OpenGL only) or stb_image to load images for textures.
- You may use GLM or DirectXMath for doing vector/matrix math in C++.
- You may use tinyobjloader to load 3D models.

Additional sample code will be provided during lab hours.

Additional Resources

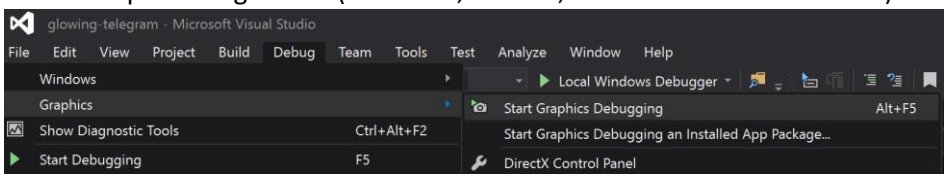
- This tutorial can help you with most of this assignment: https://open.gl/
- Use the OpenGL Wiki to learn more about OpenGL: https://www.opengl.org/wiki
- If you have OpenGL bugs, read this first: https://www.opengl.org/wiki/Common_Mistakes
- You can search for OpenGL functions documentation using http://docs.gl/

OpenGL and DirectX Debuggers

Programming with a GPU can be like working with a black box. It's hard to see what's happening inside. If you get a blank screen, nonsense results, or a GPU driver crash, it can be hard to find what caused it. To make your life easier, consider using a debugger to investigate the GPU step-by-step to find bugs.

The following debuggers are recommended:

- Crytek's RenderDoc (Windows, OpenGL/DirectX, **works on lab computers as portable zip**)
  - https://www.cryengine.com/renderdoc
- Visual Studio Graphics Diagnostics (Windows, DirectX, built-in Visual Studio 2015)



- NVIDIA Nsight (Windows, OpenGL/DirectX, NVIDIA Only, Visual Studio plugin)
  - https://developer.nvidia.com/nvidia-nsight-visual-studio-edition
- AMD CodeXL (Windows, Linux, OpenGL/DirectX, AMD Only, Visual Studio plugin)
  - http://gpuopen.com/compute-product/codexl/
- Intel GPA (Windows/DirectX or Linux/OpenGL, Intel Only)
  - https://software.intel.com/en-us/gpa