CAP 5705 Computer Graphics
Dr. Corey Toler-Franklin

OpenGL Programming
DUE: November 9th, 11:59 pm

Overview | Details | Resources | Getting Help | Submitting

Overview

Gaming applications use the OpenGL graphics pipeline to create 2-D raster images of 3-D scenes for interactive graphics. In this assignment, you will gain experience with aspects of the OpenGL pipeline including vertex processing, matrix transformations and shader programs.

Details

Getting Started

You are free to alter the provided coding framework to accomplish your task. The framework is simply a guide. Extract the file proj3_4_opengl.zip to your local folder. The source code framework is in the *src* folder. Datasets (meshes, textures and scenes) are found in the *data* folder. Other helpful materials are in the *docs* folder. You may add folders and files to your project zip file but keep the same directory structure for the framework you were given.

To build the proj3_4_opengl application on linux, type *make* in the *src/mainsrc* folder. Use any platform but **verify that your make file compiles on the linux systems *thunder* or *storm* before you submit. You will lose points if your code does not compile and run on *thunder* or *storm*.** The TA will post information on how to access *thunder* and *storm* in Canvas. Included in the package is libst, an open source openGL wrapper, and tinyojloader (by Syoyo Fujita), an OBJ file object loader, and librt from proj2_raytracer. The program is executed from the main function in the file mainsrc/mainsrc_proj3_4.cpp. **Search the project files for *TO DO: Proj3_4 opengl*. In the following sections, you will insert or alter code at these locations to complete the assignment.** It is suggested that you progress through the assignment in the order presented below.

Part I Scene Manipulation

In *Part I*, you will leverage the OpenGL pipeline, specialized hardware and linear algebra to create images in real time. You will implement 3-D manipulators, selectable graphics that allow you to interactively control the position of objects in a scene graph.

Your work in *Part I* will be evaluated based on the following criteria (1) source code completion and correctness 30% (2) render tree traversal 20% (3) camera matrix updates 20% (4) matrix transformations and manipulators 20% (5) *Part I* of your written report 10%.

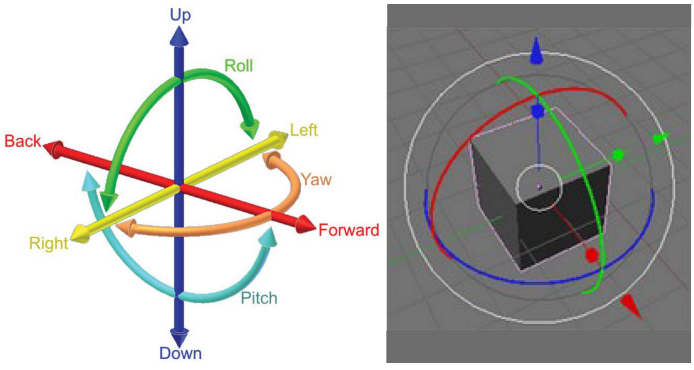

Figure 1: 3-D Manipulators

Render Tree Transversal
The *Scene* class stores objects in a tree structure. The root node has no parents. Each node in the tree holds a scene object (lights, cameras, triangle meshes). Recall from lecture that groups are nodes with children. Transformations at the group level are applied to all children in the group. The current group's transform is an accumulation of all transforms in the path up to this group. Apply transformation matrices to objects so they have the correct world coordinates. Complete the function *PropogateTransforms* that traverses the scene tree and propagates group transformations to child nodes. The order of matrix multiplication determines whether transformations occur locally, or in the parent coordinate system. Remember to preserve surface normal orientations.

Camera Matrix
Next, modify the camera viewing transformations. These are updated by the mouse callback functions located in the mainsrc_proj3_4.cpp file. To render the scene, you need a viewing transformation (camera transformation) to map world space points to camera space, and a projection transformation that maps the camera space to the conical view volume. Examine the *Camera* class' *Rotate* and *Translate* functions. The translation function moves the camera forward and backward by mapping changes in mouse motion to changes in the camera position. The *Rotation* function should either rotate a stationary camera about the viewpoint (*fly* mode) or about an object it is inspecting (*orbit* mode). Press c to toggle between these modes. As rotation matrices are not cumulative, you must handle ambiguous mouse motions that involve multiple axes. Make sure the aspect ratio of the camera viewport size is adjusted properly in relation to the 2-D screen size.

Manipulators
Manipulators are user controllable 3-D graphics that map movements of an input device in 2-D screen space to 3-D matrix transformations in the scene. Complete the *Rotate* and *Translate* functions in the *scene* class to create the manipulator shown in Figure 1. There are two manipulation modes. One updates matrix transformations locally in the object coordinate system, while the other updates transformations in relation to the parent coordinate system. This is controlled by the order of matrix multiplications. Use the *k* key to toggle manipulation modes. When you select an object in *local* or *parent* mode, the manipulator appears centered around the object. Drag the manipulator geometry to update matrix transformations in proportion to the amount of mouse movement. Code is already implemented to select objects, add manipulators, detect when manipulators are selected and call your code when the user drags the manipulator parts. You will compute the changes to the matrix transformations of the object. Examples of similar manipulators are found here https://www.youtube.com/watch?v=KprOKi7OY18.

Rotation To control rotation in each of the three axes, the user selects and rotates one of three intersecting circles perpendicular to the axis of rotation. To implement this functionality, you will complete the *Rotate* function for the given rotation axis. Tracking mouse positions for this manipulator is non-trivial. Therefore, you may map the vertical mouse motion directly to the rotation angle. Update the rotation angle by some reasonable constant value.

Translation To control translation in a given direction, select and drag the red, green or blue arrow parallel to the x, y and z axes respectively. Recall the definition of a *ray* from proj2_raytracer. The translation manipulator will translate the object along a ray from some original position e along an interval t. To implement the *Translate* function, you should map the change in the mouse motion to the coordinates of the selected transform.

Save screen shots that demonstrate (1) your manipulator (translation and rotation) and (2) your camera translation, orbit and fly. Include these, along with a brief description of how you completed *Part I* in your report. Section *Submitting* lists what to submit for *Part I*.

Part II Shaders

Shader programs are essential components of the modern OpenGL pipeline. In *Part II*, you will use the OpenGL shader language, GLSL, to implement shader programs that process high level algorithms efficiently using the graphics processing unit (GPU).

Your work in *Part II* will be evaluated based on the following criteria (1) source code completion and correctness 30% (2) Reflectance direction calculation 10% (3) Cube map 20% (4) Texture lookup 20% (5) Result image 10% (6) *Part II* of your written report 10%.

Your goal is to learn shader programing. You will focus most of your time on implementing the shader program in the specified shader files. The code framework provides infrastructure for compiling, loading and running your shaders. The *STShaderProgram* is the main class for managing your shader. Examples of vertex and fragment shaders can be found in *mainsrc/kernels*.

Environment lighting uses texture-mapping (instead of point lights) to implement complex scene lighting. This is different from other examples that compute reflectance by plugging parameters into a specific lighting equation. Implement a specular (mirror-like) reflection under environment lighting. To do this, perform a texture look-up in a given direction to determine the specular reflection of a point on the object. The color of the object at the specified point comes directly from your texture.

Figure 2: Reflections: Environment Lighting

Implement vertex and fragment shaders to complete this task. Refer to the lecture notes and section 11.6 of the textbook for more details. OpenGL uses a cube map texture to implement environment lighting. This texture stores six images on a cube to represent the environment. The environment light intensity in a certain direction is determined by sampling a point along a 3-D vector that intersects a cube map. The texture look-up is implemented in the shader function *getEnvColor*. Simply look-up the color of the environment in the direction of the viewing ray. First, complete the implementation of the skybox which has been partially implemented for you. The interior of the cube map is your background. It is important for realism to look-up the background environment intensity for rays that do not hit the model. Next, given a camera viewing direction and a normal direction, compute the specular reflection direction under environment lighting using the reflectance equation. Use the result to sample the cube map. You will implement the environment map texture lookup in the *envmap.vert* and *envmap.frag* shader files. Your specular reflection computation will be implemented in the *reflectionmap.frag* and *reflectionmap.vert* files. These can be found in the *mainsrc/kernels* directory. Figure 2 shows an environment mapping result.

Save your reflectance result and include it in your packet and in your written report. Section *Submitting* lists what to submit for *Part II*.

Resources

The openGL Programming Guide, available online https://www.opengl.org/sdk/ is a good reference. Appendix A also provides information about GLUT and GLEW. Chapter 12 of the textbook, Fundamentals in Computer Graphics, covers meshes. Information about the *.obj* file format is in the *docs* folder.

If you are not in the CISE department, and would like computer lab access, you can register for a CISE account at https://www.cise.ufl.edu/help/account. The source packet proj1_mesh.zip will run on machines in the computer labs on the first floor of the CSE building. Labs are open 24 hours (see https://www.cise.ufl.edu/help/access). **Remember to back-up your work regularly!!!** Use version control to store your work. DO NOT PUBLISIZE SOLUTIONS.

Getting Help

Source Code Please do not re-invent the wheel. Use the source code framework (and comments) and review the notes in the docs folder.

Discussion Group Post questions to Canvas (everyone benefits in this case), or send me an email at ctoler@cise.ufl.edu. I will check both daily.

Office Hours Stop by my office, CSE 332 (or lab CSE 319) during office hours MWF 10:40 to 11:30am or see the TA in room 339 TH 3:00 to 5:00pm.

Collaborating Work independently. Remember to always credit outside sources you use in your code. University policies on academic integrity must be followed.

Submitting

Upload your proj3_4 _OpenGL.zip file to Canvas. It should include:

- Source code with make file. Before you submit, build and run on *thunder* or *storm*.
- One written report that includes:
  - screen shots demonstrating manipulator translation and rotation
  - screen shots demonstrating camera translation, orbit and fly
  - reflection environment map result
  - a few lines that explain your implementation *Part I*
  - a few lines that explain your implementation *Part II*
  - anything you would like us to know (how to run your code, bugs, difficulties)
- original reflection environment map result from shader