Assignment 1: “Hello, Graphics World”

CS 4600 Computer Graphics
Fall 2016
Ladislav Kavan

The goal of this assignment is to code up a simple graphical application. This is the very first assignment in the course and therefore it should be very easy. It serves as a warm up and gentle introduction to the world of computer graphics. Note that subsequent assignments will be much more difficult and time consuming. This is an individual assignment, i.e., you have to work independently. All information needed to complete this homework is covered in the lectures and discussed at our Canvas Discussion Boards. You shouldn’t have to use any textbooks or online resources, but if you choose to do so, you must reference these resources in in your final submission. It is strictly prohibited to reuse code or fragments of code from textbooks, online resources or other students -- in this course this is considered as academic misconduct (https://www.cs.utah.edu/academic-misconduct/).

The framework code is written in C++ with the following dependencies:

- OpenGL 1.0
- GL Utilities (GLU)
- C++ STL
- OpenGL Extension Wrangler Library (GLEW)
- GLFW3

The recommended IDE is Visual Studio 2013 Community Edition, which is available free of charge for educational purposes. The framework code provides precompiled dependencies for Visual Studio 2013. If you choose to use a different platform or IDE version it is your responsibility to build the dependencies and get the project to work. The provided source code, after being successfully compiled, linked, and executed, should display a static teapot with constant gray color (no shading).

The assignment consists of two parts: normal computation and view matrix computation. Both parts should be implemented in the *main.cpp* file using specified subroutines. No other source code / dependencies / libraries are needed or allowed for this assignment.

Introduction

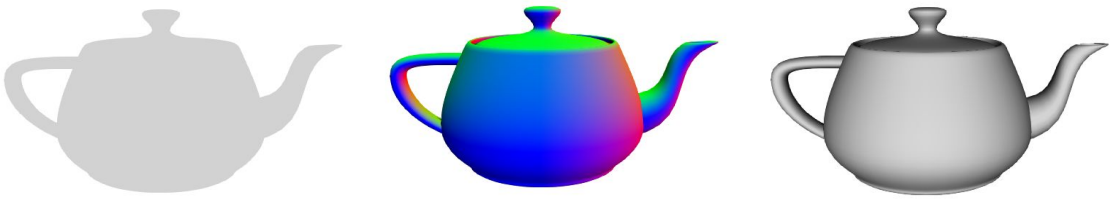
This assignment uses the very basic, yet easy to learn and understand fixed function pipeline of OpenGL. In modern OpenGL, using the programmable pipeline gives more freedom to the programmer, but it also makes it more difficult to understand. OpenGL is a state machine. Once you set a state anywhere in the code, which runs in a thread of an OpenGL context, it remains in that state until set otherwise. The OpenGL states can be

changed by calling procedures with prefix “gl”, such as *glEnable, glMatrixMode, glColor3f*, etc.

When the program starts and the main function is called, a new window with an OpenGL context is initialized using the GLFW library. The program continues by changing a few states of OpenGL, i.e. enabling lighting, setting background color and finally setting the projection matrix. Next, the Utah teapot mesh is loaded from a file TEAPOT.OBJ into a std::vector of floats, *g_meshVertices*, so that it contains the sequence of vertex coordinates $(x_1, y_1, z_1, x_2, y_2, z_2, \ldots)$. Vertex indices for each triangle of the mesh are also stored sequentially in a std::vector of integers, *g_meshIndices*, as $(a_1, b_1, c_1, a_2, b_2, c_2, \ldots)$, where a_1, b_1, c_1 are vertex indices of the first triangle and so on. Finally, a rendering loop is called. This loop sets the OpenGL modelView matrix, renders the teapot using OpenGL immediate mode (i.e. by calling *glBegin, glVertex3f,* and *glEnd*) and checks for I/O events until ESC key is pressed or the window is closed. But do not worry too much about the technical details. Instead, we will focus on some elementary graphics operations.

1 Compute Normals (50 points)

The goal of this part is to compute normal vectors for each vertex of the Utah teapot mesh in order to get basic lighting to work. OpenGL fixed function pipeline uses Blinn–Phong reflection model that describes the way a surface reflects light as a combination of an ambient, diffuse, and specular reflection (we will cover reflection models later in this course). In this task, the main thing is that this reflection model requires a normal of a surface point at each vertex in order to compute the color of reflected light. Lighting enhances realism of the rendering significantly as shown in the figure below:



(a) Lighting disabled (b) Color coded normal directions (c) Lighting enabled

Normals of each vertex are stored sequentially in a std::vector of floats, *g_meshNormals,* in order of the vertices, $(n_{x1}, n_{y1}, n_{z1}, n_{x2}, n_{y2}, n_{z2}, \ldots)$. If a vertex is shared by multiple triangles, the normal for such vertex is an average of all surface normals of all triangles sharing this vertex. All normal vectors should be normalized, i.e. length(normal) = 1. Your task is to complete the *computeNormals* function. You may want to use functions *crossProduct* and *normalize* which we have prepared for your convenience.

2 Make the teapot rotate (50 points)

Your second task is to make the teapot rotate about the vertical axis, as shown in this figure:

Rotating Utah teapot

In OpenGL, this can be done using a model-view matrix, which describes the relative transformation between the model (teapot) and the camera. There are several ways to understand this (and we'll go over this later in the course), but the easiest way is to remember that the OpenGL camera is centered at [0,0,0], with x-axis (i.e. [1,0,0]) pointing to the right, y-axis ([0,1,0]) pointing up and z-axis ([0,0,1]) pointing *behind* the camera. The last bit is slightly counter-intuitive, but it is like that: the camera is looking in the direction of the *negative* z-axis, i.e. [0,0,-1].

The *modelView* matrix can be understood as the (homogeneous) matrix which is used to transform the coordinates of your object into the camera space. If you set this matrix to identity, you will see nothing because the camera is exactly inside the teapot. So the provided framework translates the teapot -5 units along the z-axis, so the teapot is nicely visible to the OpenGL camera. There are many more things happening after the transformation (perspective projection etc.), but we'll worry about those later. For now, your task is to modify the function *updateModelViewMatirx* in such a way that the teapot starts to rotate on the screen about the y-axis. You may want to use the function *getTime* to query the current system time.

3 Extra Credit (up to 20 bonus points at instructor's discretion)

When you're finished with Task 2, you might notice the rotation axis is not perfectly aligned with the teapot. This is because the teapot rotates about its center of mass, which also accounts for the handle and the spout. The animation would be more visually pleasing if we make the teapot rotate about the center of its body *only*. You can also make the teapot stand still when the program runs, and gradually spin it in a smooth transition, as if the teapot was rotated by physically-realistic torque (force/torque always needs time to translate into velocity, it never happens on a dime; no matter how strong your car brakes are, you have to apply them *before* the car is in contact with a wall).

If you are curious what else we can do with normals, you can also experiment with basic OpenGL shading models (i.e. how colors of the vertices are interpolated during rasterization). Default shading model is *GL_SMOOTH* which is an implementation of Gouraud shading. There's also a *GL_FLAT* shading model which you can set using the *glShadeModel* function. You can also try to change the teapot's color. A particularly creative real-time rendering and animation of the Utah teapot will be rewarded with extra credit.

4 Submission

When you're finished with Tasks 1 and 2 (and possibly the optional Task 3), you'll need to submit the following:

- Source code (you should only modify main.cpp so that's all we need)
- PDF document describing what you did, with several screenshots; if you used any textbooks or online resources that may have inspired your way of thinking about the assignment, you must references these resources in this document
- The PDF document should also include a link (URL) to a video which you need to upload on your YouTube channel. The video will be a screen capture of your final rotating teapot. There are many free screen capture utilities that can help you with this.

Please pack the source code and the PDF document to a single ZIP file named Lastname_Firstname_HW1.zip

Please submit this ZIP file via Canvas.