CMSC 435 / 634 Introduction to Computer Graphics

Project Assignment 1: Simple Scene from Primitives

Goals of this project

Get hands-on experience with OpenGL; learn simple geometry primitives construction through ***geometry tessellation***.

Introduction

In this assignment you will model a simple scene that uses several primitives. Your job is to

- design a primitives library that draws the following primitives in the wireframe form: plane, cube, sphere, cylinder, and cone. These primitives must be ***tessellated***. Using glu and glut routines is not allowed. Your code must handle the level of details in the tessellation.
- use your primitives library to construct an interesting scene. Your scene must include a robot standing on the floor constructed using your primitives. Note that your robot must use all primitives. For those that are not used, place them on the ground plane.

The window layout and camera control are provided, as well as the beginnings of simple yet sophisticated object manipulation. You will write the code that constructs primitives for all the shapes in the scene and manipulates their positions.

Requirements:

Computer Graphics often deals with 3D scenes. The catch however is that your computer screen can only display 2D images. Therefore, there needs to be a way to convert a 3D scene to something that can be viewed in 2D screen. A common method which we will use in this and later assignments, is by composing a scene using only triangles, and then projecting those triangles to the screen, drawing each one sequentially. Triangles in computer graphics are the simplest geometric surface units, so all other surfaces can be reduced to triangles.

In this assignment, you will be implementing the portion of this project that pertains to **tessellating** objects. In another words, you will be breaking up the "standard shapes – the primitives - plane, box, sphere, disk, and cone, into a lot of triangles, that when put together, look as much like the desired shapes as possible. Flat-faced objects will be pretty simple and will come out looking just like the actual shape. On the other hand, curved surfaces won't look *exactly* like the real thing (our eyes can be easily fooled). It is possible to get a better approximation of a curved surface using more triangles, but keep in mind that more to draw means more to compute, and a major motivation behind tessellating objects is to simply the process of displaying them (we will learn the graphics pipeline later).

One of the most exciting aspects of this assignment is that the "building-block" nature of the assignments. In other words, you won't just hand in your code and expect never to user it again, but rather you can expect to see the fruits of your labor from this assignment even in the final one (or so we hope!). This of course means that careful planning and a good design are paramount (you knew that!).

What you must do is write the routines that, given a primitive, will computer the actual three-dimensional triangles needed to simulate the objects. You will be using OpenGL commands glVertex3f or glVertex4f to draw your shapes. You only need to tessellate five objects: a plane, a cube, a sphere, a cylinder, and a cone.

The tessellation values will take on different meanings depending on the object you are tessellating. For the radically symmetric shapes (sphere, cylinder, and cone), the first parameter should represent the number of "stacks", and the second should be the number of "slices." For the cube, it really only make sense to utilize one of the tessellation values, which would be proportional to the amount one of the faces is subdivided. Slice lines are like longitude and stack lines are like latitude.

The actual details of the tessellation are left up to you. However, when you draw triangles, making sure the three coordinates are drawn in the counter-clock wise manner viewed from the eye (camera).

Shape Specification:

Now when we say "tessellation some shape," you're going to need a lot more information than just tessellation parameters. Where a shape is found in the scene as well as size and orientation are important in writing the good, consistent tesselators that you will need for later assignments. To simply matters and eliminate lot of special cases, a trick that is often followed is to deal with a shape only at some set location, and mathematically transform (scale / rotate / translate) the shape to meet the demands of a particular scene. Here are specifications for the shapes you will be tessellating (all are centered at the origin):

- **Plane** – the plane has the unit length edges. Hence, it goes from -0.5 to 0.5 along x and y axes.
- **Cube** – the cube also has the unit length edges. Hence, it goes from -0.5 to 0.5 along all three axes.
- **Cylinder** – the cylinder has a height of one unit, and is one unit in diameter. The Y axis passes vertically through the center; the ends are parallel to the XZ plane. So the extends are once again -0.5 to 0.5 along all axes.
- **Cone** – the cone also fits within a unit cube, and is similar to the cylinder but with the top (the end of the cylinder at Y=0.5) pinched to a point.
- **Sphere** – the sphere is centered at the origin, and has a radius of 0.5.

Scene Specification:

Your scene will use the primitives you have created. To construct the robot you will need to call the following routines:

glTranslatef; glScalef, glBegin, glEnd;

Support Code:

A Makefile is provided and all the modifications that need to be made to it are for you to add the name of your object files and source files. The support code comes prepackaged to draw a single triangle, which should help you get started on the right track.

643 only:

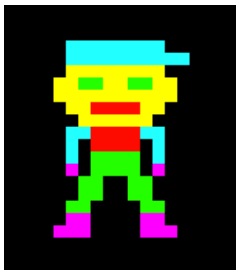
Besides those shape primitives, you will also build one more special shape: a torus and a geodesic sphere.

Extra credits

For those taking 435, for 5 points, implement a torus primitive and for another 5 points, implement a geodesic sphere.

For 3 credits, create a second scene that can be anything of your interest and again it must use all primitives – *be creative!*

For 10 points, build a pixel primitive that takes an input array as an image representation and turn that into pixel groups drawn on the screen (in the screen coordinates). For example, the LittleMario.txt will create an image on the right on the screen.



The key OpenGL functions you are to use include *glDrewPixels* and *glPixelZoom*.

Scene overview

This section gives an overview of how the scene is constructed; there are more details in the following section about the primitives.

Primitives library overview

This section describes how the primitives should be designed.

Strategy

Incremental development will probably result in the most efficient use of your time. For example, first try to get your program to draw a triangle. Use the move function

provided in the code to move the triangle to understand how the *Move* function works. Once the triangle is done, add other primitives. Once you've got the primitives, make the scene interesting by moving objects around using the Keyboard routine.

What to turn in

Source code only by email to TA. Please do not include any .o files. Please include:

- A README with your handin containing basic information about your design decisions and any known bugs or extra credit;
- How to compile and run your code as if you are telling a colleague that is to continue the development.