Assignment 1: Shapes

Comp175: Computer Graphics – Spring 2014

Algorithm due: **Monday February 3rd** at 11:59pm
Assignment due: **Monday February 10th** at 11:59pm

1 Introduction

Computer graphics often deals with images depicting a view of a 3-dimensional scene. The catch, however, is that a monitor can only display 2-dimensional images. Our task? Convert a 3-dimensional scene into something that can be viewed in 2 dimensions. A common method, which we will use in this and later assignments, is to compose a scene using only triangles, and then sequentially projecting and drawing those triangles onto the screen. In this projection, triangles hidden behind other triangles simply aren't drawn. Triangles are the simplest geometric surface unit, and all other surfaces can be reduced to triangles or *tessellated*.

In this assignment, you will be writing the *tessellation* portion of this process. That is, you will be breaking up the "standard" shapes – or *primitives* – into a lot of triangles that, when put together, look as much like the desired shape as possible. Flat objects will be pretty simple, and will come out looking just like the actual shape. Curved surfaces, on the other hand, won't look *exactly* like the real thing. The more triangles you use, the closer the approximation will mimic the actual object, but keep in mind that more to draw means more to compute, and a major motivation behind tessellating objects is to simplify and speed up the process of displaying them.

2 Demo

The demo presents you with a list of shapes from which to choose, the option of selecting either a solid or wireframe view, and some spinners to allow you to modify how finely tessellated the object is. Rotation and zooming are also done via spinners.

Notice that if you use really high tessellation values (i.e. using lots and lots of tiny triangles), even shapes with curved surfaces look really good. On the other hand, you may notice that the time it takes to draw the shape is roughly proportional to the number of triangles used in the tessellation. If you try to rotate a finely tessellated shape, you will probably notice a slowdown in the refresh speed. The problem becomes even more apparently if there are multiple detailed objects in the scene, as there will be later in the course. You may also notice the other end of this spectrum; it is sometimes desirable to set a minimum allowable tessellation value, as a value too low will cause you to lose determining features of the object.

3 Requirements

The wireframe/solid transition as well as the shape's orientation is all taken care of by the support code; you don't need to worry about that. What you must do, however, is write the routines that, given a shape and two tessellation values, compute the vertices, edges, and normals of the triangles needed to approximate the shape in 3D. You will be using the OpenGL commands `glNormal3f` and `glVertex3f` to draw your shapes. You are required to tesselate four objects: a `cube`, a `cylinder`, a `cone`, a `sphere`, and another geometric shape of your choosing (in the Demo, we implemented a `torus`, but you can be more imaginative than that).

The tessellation values will take on different meanings depending on the object you are tessellating. For the radially symmetric shapes (`sphere`, `cylinder`, and `cone`), the first parameter should represent the number of *stacks*, and the second should be the number of *slices*. Take a look at the demo if you have trouble visualizing how the parameters affect the shapes. Slice lines are like longitude, and stack lines are like latitude.

Note: *The tessellation parameters only make sense for values greater than a certain minimum value, depending on the shape and which parameter it is. As you might imagine, if you allow the parameter for the slice of a cylinder to go below 3, your cylinder is going to be flat. To avoid behavior such as this, in your implementation of this assignment you should bound the parameters at appropriate values to avoid unsightly or improper results.*

The actual details of the tessellation are left up to

you (including both how to generate the triangles and the surface normals for them). Surface normals are vectors that are *normal*, or perpendicular, to a surface. They are used for lighting calculations and shading, as you will see for yourself later in the semester. It is possible to generate a normal for each triangle (which uses less memory) or for every vertex (this produces better shading, making your shape look more curved). It is only possible to use normals per-triangle where the surface is flat. Keep in mind there are methods of tessellation that depart greatly from what is shown in the demo; it is up to you to find them. As for the shapes you will be tessellating, look in the Shape Specification section for details.

An important consideration when tessellating shapes is that whenever the user modifies one of the drawing parameters (i.e., the orientation, tessellation values, drawing style, etc.), you will need to redraw all the triangles that compose an object. Some of these adjustments change how the object is tessellated, but when they don't, you should not recompute all the triangles, but rather redraw the triangles you have already computed. In other words, you will need to keep track of all the triangles drawn for a particular shape. This organization should be done in a clean, object-oriented and extensible way. You will want to take some time to think about organizing your code. Remember that this code will be used in upcoming assignments other than Shapes.

4 Shape Specification

As you might imagine, you're going to need a lot more information than just tessellation parameters to successfully tessellate an object. The location of a shape, as well as its size and orientation are important in writing the good, consistent tessellators that you will need for later assignments. To simplify matters and eliminate lots of special cases, a convention that is often followed is to deal with a shape only at some set location (such as the origin), and mathematically move or distort the shape to meet the demands of a particular scene. Rather than force you to write completely generic tessellators for all the shapes, we encourage you to adopt this convention Here are the specifications for the shapes you will be tessellating (all are centered at the origin):

Cube The cube has 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 extents 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.

5 Support Code

You will be using the same support code as you used last time. Make a backup copy of your code before you start work on this project. The support code already contains triangle-drawing example code to get you started. See the OpenGL tutorials on the course website for additional reference material.

Begin by looking at the `Shape` class in `Shape.h`. Notice that this is an abstract class. For your assignment, you should extentiate (subclass) from the Shape class for each of the shapes that you implement (e.g., Cube.h, Sphere.h, etc.) In these classes, you'll need to fill in the `draw(...)` and `drawNormal(...)` methods. As before, you can get information from the GUI in the main `Assignment1.cpp` file. The relevant members include:

```
  OBJ_TYPE objType;
int segmentsX;
int segmentsY;
int wireframe;
int fill;
int normal;
```

The `segmentsX` variable will be used control the X tessellation parameter, and `segmentsY` will control the Y tessellation parameter. If your special shape requires a third parameter, add it at your own discretion. The `objType` parameter will select the shape that is to be drawn. Values include:

```
SHAPE_CUBE,
SHAPE_CONE,
SHAPE_SPHERE,
SHAPE_CYLINDER,
```

`SHAPE_SPECIAL1`,
`SHAPE_SPECIAL2`,
`SHAPE_SPECIAL3`

Note that if you choose to use
tt SHAPE_SPECIAL2 or
tt SHAPE_SPECIAL3, you will need to modify the GUI in the `main(...)` function.

In addition, although this is not required, we recommend that you design and implement appropriate data structures so that you don't recompute the same values at every frame. Here are some suggestions on how to do that:

- Construct a gigantic 1-dimensional array of all your point data, then carefully index into it when drawing your shapes.
- Construct a 2-dimensional array of all your point data: if you have N points, there are N elements in the first dimension. The second dimension would always be of magnitude 4 (since the points have 4 coordinates in our homogenous coordinate system, though the fourth coordinate will remain at "1" in this assignment)
- Use an STL structure to organize your points. A list or a deque would be the most obvious choices.
- There are many other options! It may help to build a simple wrapper class or struct for your point data. You may want to build up a list using the stl, then flatten it into an array after it's been finalized.

You will have to do something similar for your normals. If you have questions about your initial designs, please do come to office hours and ask for advice. Again, just to emphasize: you'll be living with this code for the entire semester, you don't want to have to struggle with bad design choices a few months down the line.

Note that the individual triangles must be passed to OpenGL in counter-clockwise order, or it will draw the faces backwards (and they will be invisible). This is a common cause of much agony, so be careful! Another common mistake that will create problems is using transformations.

6 How to Submit

Complete the algorithm portion of this assignment with your teammate. You may use a calculator or computer algebra system. All your answers should be given in simplest form. When a numerical answer is required, provide a reduced fraction (i.e. 1/3) or at least three decimal places (i.e. 0.333). Show all work.

For the project portion of this assignment, you are encouraged to discuss your strategies with your classmates. However, each team must turn in ORIGINAL WORK, and any collaboration or references outside of your team must be cited appropriately in the header of your submission.

Hand in the assignment using the following commands:

- Algorithm: `provide comp175 a1-alg`
- Project code: `provide comp175 a1`

7 FAQ

7.1 I am confused about this assignment. What am I supposed to do?

We understand that there is not much in the textbook with regard to tessellation. Keep it simple; the math in this assignment doesn't go beyond that which you learned in high school geometry, unless you attempt more complex shapes. A good approach to take if you are flustered is to try to place the triangle in the support code somewhere useful, such as on the side of the cube. Calculate, on paper, where the rest of the triangles will be placed, and think of a good way to parameterize their placement. If you're still feeling confused about shape parameterization, go to the library and get a college calculus textbook.

7.2 Some of my triangles just aren't appearing on the screen. What is wrong?

There are a few possibilities. The first is that you are simply drawing the triangles in the wrong place. The second is that you are specifying the coordinates of triangles in the wrong order. The third possibility is that you're calling `glVertex()` outside of a `glBegin()`/`glEnd()` block, in which case nothing will happen.