CS 480/680 Computer Graphics

Spring 1999 - Dr. George Bebis

Programming Assignment 4

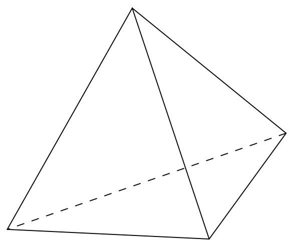
Due date: 5/4/99

In this assignment, you will generate realistic images of a sphere. Both GLU and GLUT include functions for drawing a sphere but your are **not** supposed to use them in your program. To draw a sphere, you will write your own function which will be based on a very powerful technique called *recursive subdivision*.

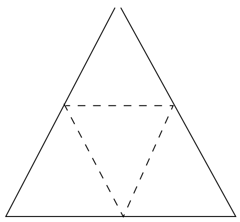
1. Recursive Subdivision

This technique is used very often in computer graphics to approximate surfaces to any desired degree of accuracy using polygons. To approximate a sphere using this approach, we first approximate a sphere using a tetrahedron (see figure). The regular tetrahedron is composed of four equilateral triangles, determined by the vertices $(0, 0, 1)$, $(0, 2\sqrt{2}/3, -1/3)$, $(-\sqrt{6}/3, -\sqrt{2}/3, -1/3)$, and $(\sqrt{6}/3, -\sqrt{2}/3, -1/3)$. To draw the tetrahedron, write a function which draws one face, by specifying the vertices of the face (*void triangle(v1,v2,v3)*), and then write another function which draws all four faces (*void tetrahedron(void)*). The order of the vertices per face should *obey the right-hand rule*.

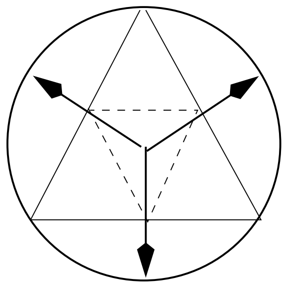
To get a closer approximation of the sphere, you must subdivide each face of the tetrahedron into smaller triangles. To do this, you can simply find the bisectors of the sides of the triangle. Then, connect the bisectors together to generate four new equilateral triangles (see figure). After a face has been subdivided as has just been described, the four new triangles will still be in the same plane as the original triangle. You can move the new vertices that you created by bisection to the unit sphere my normalizing each bisected vertex (i.e., normalize the vector connecting the center of the sphere to the point of bisection, see figure).



tetrahedron



subdivision



normalization

The subdivision process should be repeated as many times as desired to generate successively closer approximations of the sphere. You can implement this idea using recursion. Suppose that the subdivision process needs to be repeated n times (i.e., n is the depth of recursion). First, you need to make *tetrahedron()* depend on the depth of recursion by adding n to its argument list. Within *tetrahedron()*, you need to call not *triangle()* but another function (e.g., *divide_triangle(v1,v2,v3,n)*) which basically subdivides each face of the original tetrahedron n times. The function *divide_triangle()* needs to call itself $n-1$ times. When it reaches the last stage of recursion, then it needs to call *triangle()*.

2. Shading the sphere

To shade the sphere constructed by following the above methodology, you must assign normals to the vertices of the triangles to be drawn. For *flat shading*, you need to associate the normal with the first vertex of the triangle only (OpenGL does not consider the normals of the other vertices). To find the normal, use the cross-product and then normalize the result (we have discussed this in class).

For *smooth* shading, you need to assign to each vertex the corresponding normal (i.e., find the vector from the center of the sphere to the vertex and normalize it). Using *glShadeModel(GL_SMOOTH)*, OpenGL will compute the average normal per vertex for smoother shading.

3. Functionality

Your program needs to have the following functionality:

3.1 Input File

The input to your program will be a file containing information about the object in the scene (sphere), its material properties, and light source. Each line of the file may contain a single keyword followed by some values. The general format is:

```
GLOBAL_AMBIENT r g b

SPHERE
RADIUS radius
POSITION x y z
AMBIENT r g b
DIFFUSE r g b
SPECULAR r g b shininess

LIGHT
POSITION x y z
AMBIENT r g b
DIFFUSE r g b
SPECULAR r g b
SPOT_DIRECTION x y z
SPOT_CUTOFF angle
SPOT_EXPONENT exponent
```

3.2 Light

Draw a very small object (e.g., a cube) at the location of the light source. Make this object as "emissive" material of an appropriate color for the light.

3.3 Changing light and material properties

Include a menu that allows the user to change the material properties of selected spheres and light properties of selected lights. For example, you could have a menu item as "*increase red diffuse component*".

3.4 Shading

Allow the user to switch back and forth between smooth and flat shading.

Extra work for graduate students only

Include multiple light sources. Allow the user select a light source in order to: (i) translate it and/or (ii) change the light orientation (for spotlights) interactively.