Modeling with GLU and GLUT

Prerequisites

An understanding of the simple modeling with polygon primitives.

Introduction

Modeling with polygons alone would require you to write many standard graphics elements that are so common, any reasonable graphics system should include them. OpenGL includes the OpenGL Utility Library, GLU, with many useful functions, and most releases of OpenGL also include the OpenGL Utility Toolkit, GLUT. We saw in the first module that GLUT includes window management functions, and both GLU and GLUT include a number of built-in graphical elements that you can use. This module describes a number of these elements.

Definitions

The primitives that GLU provides to you are defined with some parameters that define the primitive and some that define the resolution with which the primitive is modeled. The former are element-specific and will be described when we describe each primitive, but the latter are general and are described here.

Each GLU primitive is declared as a GLUquadric and is allocated with the function

```c
GLUquadric* gluNewQuadric( void )
```

Each is a surface of revolution defined with the z-axis as the rotation axis. Each is modeled in terms of subdivisions around the z-axis, called slices, and subdivisions along the z-axis, called stacks. Figure 1 shows an example of a typical pre-built quadric, a GLUT wireframe sphere, modeled with a small number of slices and stacks so you can see the basis of this definition.

![Figure 1: A GLUT wireframe sphere with 10 slices and 10 stacks](image-url)
Below we describe the GLU primitives by listing the function prototype for each; more details may be found in the GLU section of your OpenGL manual.

**GLU cylinder:**

```c
void gluCylinder( GLUquadric* quad, GLdouble base, GLdouble top, GLdouble height, GLint slices, GLint stacks )
```

- `quad` identifies the quadrics object you previously created with `gluNewQuadric`
- `base` is the radius of the cylinder at \( z = 0 \), the base of the cylinder
- `top` is the radius of the cylinder at \( z = \text{height} \), and
- `height` is the height of the cylinder.

**GLU disk:**

The GLU disk is different from the other GLU primitives because it is two-dimensional, lying entirely within the X-Y plane. Thus instead of being defined in terms of stacks, the second granularity parameter is loops, the number of concentric rings that define the disk.

```c
void gluDisk( GLUquadric* quad, GLdouble inner, GLdouble outer, GLint slices, GLint loops )
```

- `quad` identifies the quadrics object you previously created with `gluNewQuadric`
- `inner` is the inner radius of the disk (may be 0).
- `outer` is the outer radius of the disk.

**GLU sphere:**

```c
void gluSphere( GLUquadric* quad, GLdouble radius, GLint slices, GLint stacks )
```

- `quad` identifies the quadrics object you previously created with `gluNewQuadric`
- `radius` is the radius of the sphere.

The GLUT primitives are more complex than the GLU primitives, including both solid and wireframe versions of many interesting solids. They include a cone, a cube, a dodecahedron (12-sided regular polyhedron), an icosahedron (20-sided regular polyhedron), an octahedron (8-sided regular polyhedron), a sphere, a teapot (the canonical Utah teapot, an icon of computer graphics), a tetrahedron (4-sided regular polyhedron), and a torus (doughnut). Each has a canonical position and orientation, typically centered at the origin and within a standard volume and, if it has an axis, the axis is aligned with the z-axis. As with the GLU standard primitives, some of the GLUT primitives allow you to specify the granularity of the primitive’s modeling, but some do not; the example program will show you the differences. If you have GLUT with your OpenGL, you should check the GLUT manuals for the details.

**Some examples**

Our example for this module is quite simple, initially displaying a medium-quality GLU sphere and using a menu to let you select another GLU or GLUT primitive, to zoom in or out on that primitive, and to increase or decrease the quality of the modeling when that is possible. Menus are provided by the GLUT toolkit and will be discussed later in the module on event-driven programming, but the example is much more interesting if we allow you to see most of the primitives in one function.

The heart of the example is the following code that responds to a menu selection to change the primitive you are displaying, change the resolution on your primitive, or to zoom in or
zoom out on your primitive. This code depends on the menu, so the fundamentals of the menu definition are also shown, though this is getting somewhat ahead of ourselves.

```c
// menu functions in main()
glutCreateMenu(options_menu);
glutAddMenuEntry("GLU sphere", 1);
glutAddMenuEntry("GLU cylinder", 2);
glutAddMenuEntry("GLUT dodecahedron", 3);
glutAddMenuEntry("GLUT torus", 4);
glutAddMenuEntry("GLUT teapot", 5);
glutAddMenuEntry("More detail", 6);
glutAddMenuEntry("Less detail", 7);
glutAddMenuEntry("Move in", 8);
glutAddMenuEntry("Move out", 9);
glutAttachMenuName(GLUT_RIGHT_BUTTON, "Selections");

// function to be called to handle mouse inputs
void options_menu(int input)
{
    if ((input >= 1) && (input <= 5))
    {
        selectedObject = input;
    }
    else
    {
        if (input == 6) resolution *= 2; // higher resolution
        if (input == 7) resolution = (resolution+1)/2; // lower resolution
        if (input == 8) distance -= 1.0; // move in
        if (input == 9) distance += 1.0; // move out
    }
glutPostRedisplay();
}

// function to manage the display that responds to the menu options
void display( void )
{
    GLUquadric *myQuad;
    GLdouble radius = 1.0;
    GLint slices, stacks;
    GLint nsides, rings;
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(60.0,1.0,1.0,30.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    // eye point center of view up
    gluLookAt(distance,distance,distance,0.0,0.0,0.0,0.0,1.0,0.0);
    switch (selectedObject) {
    case (1): {
        myQuad=gluNewQuadric();
        slices = stacks = resolution;
        gluSphere( myQuad, radius, slices, stacks );
        break;
    }
    case (2): {
        myQuad=gluNewQuadric();
    }
    ```
slices = stacks = resolution;
gluCylinder( myQuad, 1.0, 1.0, 1.0, slices, stacks );
break;
}
case (3): {
  glutSolidDodecahedron(); break;
}
case (4): {
  nsides = rings = resolution;
  glutSolidTorus( 1.0, 2.0, nsides, rings);
  break;
}
case (5): {
  glutSolidTeapot(2.0); break;
}
}

A word to the wise...

One of the differences between student programming and professional programming is that students are often asked to create applications or tools for the sake of learning creation, not for the sake of creating working, useful things. The graphics primitives that are the subject of the first section of this module are the kind of tools that students are often asked to use, because they require more analysis of fundamental geometry and are good learning tools. However, working programmers developing real applications will often find it useful to use pre-constructed templates and tools such as the GLU or GLUT graphics primitives. You are encouraged to use the GLU and GLUT primitives whenever they can save you time and effort in your work, and when you cannot use them, you are encouraged to create your own primitives in a way that will let you re-use them as your own library and will let you share them with others.

Code examples

* graphicsPrimitivesMenu.c