

COMP3201 – Computer Graphics

Module 2: Transformations and Scene Creation

2.7 Surface of Revolution

Another technique for producing a 3D object consists of drawing a profile and then rotating this shape about an axis in discrete steps. This results in a “discrete” approximation to a surface of revolution – a true surface of revolution requires the shape to be defined in parameterised form.

At this stage we will follow the discrete approach. Start by drawing a shape, for example the profile of a lamp.



We now want to position this profile at n equi-spaced angles around the y -axis. One approach is to calculate the transformed vertices directly using the transformation matrix

$$M_i = \begin{pmatrix} \cos \theta_i & 0 & \sin \theta_i & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta_i & 0 & \cos \theta_i & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad \theta_i = \frac{2\pi i}{n}, \quad i = 0, 1, \dots, n-1.$$

Supposing that the profile is stored in vertices v_0, \dots, v_{N-1} , then we can compute all the points required to draw the 3D representation of the lamp by constructing

$$M_1(v_0, \dots, v_{N-1}), \dots, M_{n-1}(v_0, \dots, v_{N-1}).$$

Note that

$$M_i \begin{pmatrix} x_j \\ y_j \\ z_j \\ 1 \end{pmatrix} = \begin{pmatrix} \cos \theta_i & 0 & \sin \theta_i & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta_i & 0 & \cos \theta_i & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_j \\ y_j \\ z_j \\ 1 \end{pmatrix} = \begin{pmatrix} x_j \cos \theta_i + \sin \theta_i z_j \\ y_j \\ -\sin \theta_i x_j + \cos \theta_i z_j \\ 1 \end{pmatrix} = \begin{pmatrix} x_j \cos \theta_i \\ y_j \\ x_j \sin \theta_i \\ 1 \end{pmatrix}.$$

Another approach is to use **glRotate*** to generate the transformation. There can be problems with round-off errors, so to avoid this, the profile should be rotated the full angle from its start point rather than rotating $2\pi/n$ incrementally.

The following code will exhibit problems with lack of depth-buffering, due to the order that the profile lines are drawn in. This can be seen if different segments of the lamp are drawn in different colours.

```
/* sweepvol.c */
/* Purpose: Construct a shape and sweep it around the y-axis */

#include "GL/glut.h"    /* includes gl.h */
#include "math.h"

#define KEY_ESC 27
#define MAXOUTLINES 30.0 /* maximum number of

/* Global Definitions */
struct point3D{
    float x;
    float y;
    float z;
};

/* This contains the vertex data for the lamp outline */
struct point3D v[30] = {
    {0.0, 0.0, 0.0},
    {0.1, 0.0, 0.0},
    {0.2, 0.0, 0.0},
    {0.3, 0.0, 0.0},
    {0.4, 0.0, 0.0},
    {0.5, 0.0, 0.0},
    {0.5, 0.3, 0.0},
    {0.5, 0.4, 0.0},
    {0.4, 0.55, 0.0},
    {0.3, 0.7, 0.0},
    {0.2, 0.8, 0.0},
    {0.2, 1.0, 0.0},
    {0.3, 1.0, 0.0},
    {0.4, 1.0, 0.0},
    {0.5, 1.0, 0.0},
    {0.6, 1.0, 0.0},
    {0.7, 1.0, 0.0},
    {0.8, 1.0, 0.0},
    {0.9, 1.0, 0.0},
    {1.0, 1.0, 0.0},
    {0.9, 1.05, 0.0},
    {0.8, 1.1, 0.0},
    {0.7, 1.15, 0.0},
    {0.6, 1.2, 0.0},
    {0.5, 1.25, 0.0},
    {0.4, 1.3, 0.0},
    {0.3, 1.35, 0.0},
    {0.2, 1.4, 0.0},
    {0.1, 1.45, 0.0},
    {0.0, 1.5, 0.0}
};
```

```

/* draw outline of lamp */
GLvoid drawOutline( GLvoid )
{
    int loop;

    glBegin(GL_LINE_STRIP);
    for( loop=0; loop<30; ++loop)
    {
        glVertex3f( v[loop].x, v[loop].y, v[loop].z );
    }
    glEnd();
}

/* Sweep the outline around the y axis */
GLvoid sweepOutline( GLvoid )
{
    GLint loop;
    glClear( GL_COLOR_BUFFER_BIT );

    for( loop = 0; loop < MAXOUTLINES ; ++loop )
    {
        float frac = (float)loop / MAXOUTLINES; /* frac will run
from 0 to a bit less than 1 */
        glLoadIdentity();
        glRotatef( frac * 360, 0.0, 1.0, 0.0 );
        glColor3f(1.0 - frac, 0.0, frac); /* change the colour to
show up the separate outlines */
        drawOutline();
    }
    glFlush();
}

GLvoid init( GLvoid )
{
    glClearColor( 1.0, 1.0, 1.0, 1.0);
    glMatrixMode( GL_PROJECTION );
    glLoadIdentity();
    glOrtho(-2.0, 2.0, -2.0, 2.0, -2.0, 2.0);
    glMatrixMode( GL_MODELVIEW );
    glLineWidth(3.0);
}

GLvoid keyboard( GLubyte key, GLint x, GLint y)
{
    switch (key) {
        case KEY_ESC: /* exit when escape key is pressed */
            exit(0);
    }
}

int main( int argc, char *argv[] )
{
    glutInit( &argc, argv );
    glutCreateWindow( argv[0] );

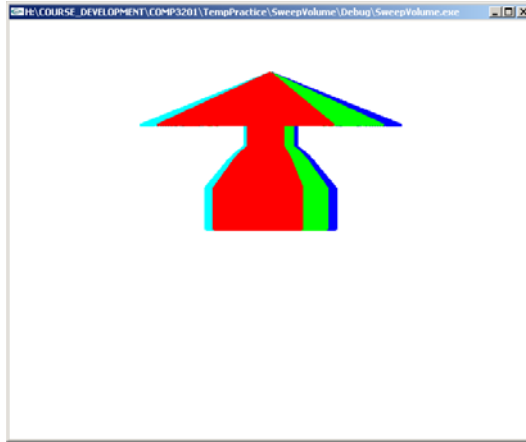
    init();

    glutKeyboardFunc( keyboard );
    glutDisplayFunc( sweepOutline );

    glutMainLoop();
}

```

```
}    return 0;  
}
```



Depth Buffering is covered in the next subsection.