////////////////////////////////////////////////////////////////////////////////////

// cylinder.cpp

////////////////////////////////////////////////////////////////////////////////////

#include <cmath>

#include <iostream>

#ifdef \_\_APPLE\_\_

# include <GLUT/glut.h>

#else

# include <GL/glut.h>

#endif

#define PI 3.14159265358979324

using namespace std;

// Globals.

static int p = 30; // Number of grid columns.

static int q = 20; // Number of grid rows

static float \*vertices = NULL; // Vertex array of the mapped sample on the cylinder.

static float Xangle = 150.0, Yangle = 60.0, Zangle = 0.0; // Angles to rotate the cylinder.

// Fuctions to map the grid vertex (u\_i,v\_j) to the mesh vertex (f(u\_i,v\_j), g(u\_i,v\_j), h(u\_i,v\_j)) on the cylinder.

float f(int i, int j)

{

return ( cos( (-1 + 2\*(float)i/p) \* PI ) );

}

float g(int i, int j)

{

return ( sin( (-1 + 2\*(float)i/p) \* PI ) );

}

float h(int i, int j)

{

return ( -1 + 2\*(float)j/q );

}

// Routine to fill the vertex array with co-ordinates of the mapped sample points.

void fillVertexArray(void)

{

int i, j, k;

k = 0;

for (j = 0; j <= q; j++)

for (i = 0; i <= p; i++)

{

vertices[k++] = f(i,j);

vertices[k++] = g(i,j);

vertices[k++] = h(i,j);

}

}

// Initialization routine.

void setup(void)

{

glEnableClientState(GL\_VERTEX\_ARRAY);

glClearColor(1.0, 1.0, 1.0, 0.0);

}

// Drawing routine.

void drawScene(void)

{

int i, j;

vertices = new float[3\*(p+1)\*(q+1)]; // Dynamic array allocation with new value of p and q.

glVertexPointer(3, GL\_FLOAT, 0, vertices);

glClear(GL\_COLOR\_BUFFER\_BIT);

glLoadIdentity();

gluLookAt (0.0, 0.0, 4.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_LINE);

glColor3f(0.0, 0.0, 0.0);

// Rotate scene.

glRotatef(Zangle, 0.0, 0.0, 1.0);

glRotatef(Yangle, 0.0, 1.0, 0.0);

glRotatef(Xangle, 1.0, 0.0, 0.0);

// Fill the vertex array.

fillVertexArray();

// Make the approximating triangular mesh.

for(j = 0; j < q; j++)

{

glBegin(GL\_TRIANGLE\_STRIP);

for(i = 0; i <= p; i++)

{

glArrayElement( (j+1)\*(p+1) + i );

glArrayElement( j\*(p+1) + i );

}

glEnd();

}

glutSwapBuffers();

}

// OpenGL window reshape routine.

void resize(int w, int h)

{

glViewport(0, 0, (GLsizei)w, (GLsizei)h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluPerspective(60.0, (float)w/(float)h, 1.0, 50.0);

glMatrixMode(GL\_MODELVIEW);

}

// Keyboard input processing routine.

void keyInput(unsigned char key, int x, int y)

{

switch(key)

{

case 27:

exit(0);

break;

case 'x':

Xangle += 5.0;

if (Xangle > 360.0) Xangle -= 360.0;

glutPostRedisplay();

break;

case 'X':

Xangle -= 5.0;

if (Xangle < 0.0) Xangle += 360.0;

glutPostRedisplay();

break;

case 'y':

Yangle += 5.0;

if (Yangle > 360.0) Yangle -= 360.0;

glutPostRedisplay();

break;

case 'Y':

Yangle -= 5.0;

if (Yangle < 0.0) Yangle += 360.0;

glutPostRedisplay();

break;

case 'z':

Zangle += 5.0;

if (Zangle > 360.0) Zangle -= 360.0;

glutPostRedisplay();

break;

case 'Z':

Zangle -= 5.0;

if (Zangle < 0.0) Zangle += 360.0;

glutPostRedisplay();

break;

default:

break;

}

}

// Callback routine for non-ASCII key entry.

void specialKeyInput(int key, int x, int y)

{

if (key == GLUT\_KEY\_LEFT) if (p > 3) p -= 1;

if (key == GLUT\_KEY\_RIGHT) p += 1;

if (key == GLUT\_KEY\_DOWN) if (q > 3) q -= 1;

if (key == GLUT\_KEY\_UP) q += 1;

glutPostRedisplay();

}

// Routine to output interaction instructions to the C++ window.

void printInteraction(void)

{

cout << "Interaction:" << endl;

cout << "Press left/right arrow keys to increase/decrease the number of grid columns." << endl

<< "Press up/down arrow keys to increase/decrease the number of grid rows." << endl

<< "Press x, X, y, Y, z, Z to turn the cylinder." << endl;

}

// Main routine.

int main(int argc, char \*\*argv)

{

printInteraction();

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);

glutInitWindowSize(500, 500);

glutInitWindowPosition(100, 100);

glutCreateWindow("cylinder.cpp");

setup();

glutDisplayFunc(drawScene);

glutReshapeFunc(resize);

glutKeyboardFunc(keyInput);

glutSpecialFunc(specialKeyInput);

glutMainLoop();

return 0;

}

///////////////////////////////////////////////////////////////////////////////////////

// hemisphere.cpp

///////////////////////////////////////////////////////////////////////////////////////

#include <cmath>

#include <iostream>

#ifdef \_\_APPLE\_\_

# include <GLUT/glut.h>

#else

# include <GL/glut.h>

#endif

#define PI 3.14159265

using namespace std;

// Globals.

static float R = 5.0; // Radius of hemisphere.

static int p = 6; // Number of longitudinal slices.

static int q = 4; // Number of latitudinal slices.

static float Xangle = 0.0, Yangle = 0.0, Zangle = 0.0; // Angles to rotate hemisphere.

// Drawing routine.

void drawScene(void)

{

int i, j;

glClear (GL\_COLOR\_BUFFER\_BIT);

glLoadIdentity();

// Command to push the hemisphere, which is drawn centered at the origin,

// into the viewing frustum.

glTranslatef(0.0, 0.0, -10.0);

// Commands to turn the hemisphere.

glRotatef(Zangle, 0.0, 0.0, 1.0);

glRotatef(Yangle, 0.0, 1.0, 0.0);

glRotatef(Xangle, 1.0, 0.0, 0.0);

glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_LINE);

glColor3f(0.0, 0.0, 0.0);

// Array of latitudinal triangle strips, each parallel to the equator, stacked one

// above the other from the equator to the north pole.

for(j = 0; j < q; j++)

{

// One latitudinal triangle strip.

glBegin(GL\_TRIANGLE\_STRIP);

for(i = 0; i <= p; i++)

{

glVertex3f( R \* cos( (float)(j+1)/q \* PI/2.0 ) \* cos( 2.0 \* (float)i/p \* PI ),

R \* sin( (float)(j+1)/q \* PI/2.0 ),

R \* cos( (float)(j+1)/q \* PI/2.0 ) \* sin( 2.0 \* (float)i/p \* PI ) );

glVertex3f( R \* cos( (float)j/q \* PI/2.0 ) \* cos( 2.0 \* (float)i/p \* PI ),

R \* sin( (float)j/q \* PI/2.0 ),

R \* cos( (float)j/q \* PI/2.0 ) \* sin( 2.0 \* (float)i/p \* PI ) );

}

glEnd();

}

glFlush();

}

// Initialization routine.

void setup(void)

{

glClearColor(1.0, 1.0, 1.0, 0.0);

}

// OpenGL window reshape routine.

void resize(int w, int h)

{

glViewport(0, 0, (GLsizei)w, (GLsizei)h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glFrustum(-5.0, 5.0, -5.0, 5.0, 5.0, 100.0);

// gluPerspective(90.0, 1.0, 5.0, 100.0);

// gluPerspective(90.0, (float)w/(float)h, 5.0, 100.0);

glMatrixMode(GL\_MODELVIEW);

}

// Keyboard input processing routine.

void keyInput(unsigned char key, int x, int y)

{

switch(key)

{

case 27:

exit(0);

break;

case 'P':

p += 1;

glutPostRedisplay();

break;

case 'p':

if (p > 3) p -= 1;

glutPostRedisplay();

break;

case 'Q':

q += 1;

glutPostRedisplay();

break;

case 'q':

if (q > 3) q -= 1;

glutPostRedisplay();

break;

case 'x':

Xangle += 5.0;

if (Xangle > 360.0) Xangle -= 360.0;

glutPostRedisplay();

break;

case 'X':

Xangle -= 5.0;

if (Xangle < 0.0) Xangle += 360.0;

glutPostRedisplay();

break;

case 'y':

Yangle += 5.0;

if (Yangle > 360.0) Yangle -= 360.0;

glutPostRedisplay();

break;

case 'Y':

Yangle -= 5.0;

if (Yangle < 0.0) Yangle += 360.0;

glutPostRedisplay();

break;

case 'z':

Zangle += 5.0;

if (Zangle > 360.0) Zangle -= 360.0;

glutPostRedisplay();

break;

case 'Z':

Zangle -= 5.0;

if (Zangle < 0.0) Zangle += 360.0;

glutPostRedisplay();

break;

default:

break;

}

}

// Routine to output interaction instructions to the C++ window.

void printInteraction(void)

{

cout << "Interaction:" << endl;

cout << "Press P/p to increase/decrease the number of longitudinal slices." << endl

<< "Press Q/q to increase/decrease the number of latitudinal slices." << endl

<< "Press x, X, y, Y, z, Z to turn the hemisphere." << endl;

}

// Main routine.

int main(int argc, char \*\*argv)

{

printInteraction();

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(500, 500);

glutInitWindowPosition(100, 100);

glutCreateWindow("hemisphere.cpp");

setup();

glutDisplayFunc(drawScene);

glutReshapeFunc(resize);

glutKeyboardFunc(keyInput);

glutMainLoop();

return 0;

}

////////////////////////////////////////////////////////////////////////////////////

// torus.cpp

////////////////////////////////////////////////////////////////////////////////////

#include <cmath>

#include <iostream>

#ifdef \_\_APPLE\_\_

# include <GLUT/glut.h>

#else

# include <GL/glut.h>

#endif

#define PI 3.14159265358979324

#define R 2.0

#define r 0.5

using namespace std;

// Globals.

static int p = 6; // Number of grid columns.

static int q = 4; // Number of grid rows

static float \*vertices = NULL; // Vertex array of the mapped sample on the torus.

static float Xangle = 150.0, Yangle = 0.0, Zangle = 0.0; // Angles to rotate the torus.

// Fuctions to map the grid vertex (u\_i,v\_j) to the mesh vertex (f(u\_i,v\_j), g(u\_i,v\_j), h(u\_i,v\_j)) on the torus.

float f(int i, int j)

{

return ( ( R + r \* cos( (-1 + 2\*(float)j/q) \* PI ) ) \* cos( (-1 + 2\*(float)i/p) \* PI ) );

}

float g(int i, int j)

{

return ( ( R + r \* cos( (-1 + 2\*(float)j/q) \* PI ) ) \* sin( (-1 + 2\*(float)i/p) \* PI ) );

}

float h(int i, int j)

{

return ( r \* sin( (-1 + 2\*(float)j/q) \* PI ) );

}

// Routine to fill the vertex array with co-ordinates of the mapped sample points.

void fillVertexArray(void)

{

int i, j, k;

k = 0;

for (j = 0; j <= q; j++)

for (i = 0; i <= p; i++)

{

vertices[k++] = f(i,j);

vertices[k++] = g(i,j);

vertices[k++] = h(i,j);

}

}

// Initialization routine.

void setup(void)

{

glEnableClientState(GL\_VERTEX\_ARRAY);

glClearColor(1.0, 1.0, 1.0, 0.0);

}

// Drawing routine.

void drawScene(void)

{

int i, j;

vertices = new float[3\*(p+1)\*(q+1)]; // Dynamic array allocation with new value of p and q.

glVertexPointer(3, GL\_FLOAT, 0, vertices);

glClear(GL\_COLOR\_BUFFER\_BIT);

glLoadIdentity();

gluLookAt (0.0, 0.0, 6.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_LINE);

glColor3f(0.0, 0.0, 0.0);

// Rotate scene.

glRotatef(Zangle, 0.0, 0.0, 1.0);

glRotatef(Yangle, 0.0, 1.0, 0.0);

glRotatef(Xangle, 1.0, 0.0, 0.0);

// Fill the vertex array.

fillVertexArray();

// Make the approximating triangular mesh.

for(j = 0; j < q; j++)

{

glBegin(GL\_TRIANGLE\_STRIP);

for(i = 0; i <= p; i++)

{

glArrayElement( (j+1)\*(p+1) + i );

glArrayElement( j\*(p+1) + i );

}

glEnd();

}

glutSwapBuffers();

}

// OpenGL window reshape routine.

void resize(int w, int h)

{

glViewport(0, 0, (GLsizei)w, (GLsizei)h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluPerspective(60.0, (float)w/(float)h, 1.0, 50.0);

glMatrixMode(GL\_MODELVIEW);

}

// Keyboard input processing routine.

void keyInput(unsigned char key, int x, int y)

{

switch(key)

{

case 27:

exit(0);

break;

case 'x':

Xangle += 5.0;

if (Xangle > 360.0) Xangle -= 360.0;

glutPostRedisplay();

break;

case 'X':

Xangle -= 5.0;

if (Xangle < 0.0) Xangle += 360.0;

glutPostRedisplay();

break;

case 'y':

Yangle += 5.0;

if (Yangle > 360.0) Yangle -= 360.0;

glutPostRedisplay();

break;

case 'Y':

Yangle -= 5.0;

if (Yangle < 0.0) Yangle += 360.0;

glutPostRedisplay();

break;

case 'z':

Zangle += 5.0;

if (Zangle > 360.0) Zangle -= 360.0;

glutPostRedisplay();

break;

case 'Z':

Zangle -= 5.0;

if (Zangle < 0.0) Zangle += 360.0;

glutPostRedisplay();

break;

default:

break;

}

}

// Callback routine for non-ASCII key entry.

void specialKeyInput(int key, int x, int y)

{

if (key == GLUT\_KEY\_LEFT) if (p > 3) p -= 1;

if (key == GLUT\_KEY\_RIGHT) p += 1;

if (key == GLUT\_KEY\_DOWN) if (q > 3) q -= 1;

if (key == GLUT\_KEY\_UP) q += 1;

glutPostRedisplay();

}

// Routine to output interaction instructions to the C++ window.

void printInteraction(void)

{

cout << "Interaction:" << endl;

cout << "Press left/right arrow keys to increase/decrease the number of grid columns." << endl

<< "Press up/down arrow keys to increase/decrease the number of grid rows." << endl

<< "Press x, X, y, Y, z, Z to turn the torus." << endl;

}

// Main routine.

int main(int argc, char \*\*argv)

{

printInteraction();

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);

glutInitWindowSize(500, 500);

glutInitWindowPosition(100, 100);

glutCreateWindow("torus.cpp");

setup();

glutDisplayFunc(drawScene);

glutReshapeFunc(resize);

glutKeyboardFunc(keyInput);

glutSpecialFunc(specialKeyInput);

glutMainLoop();

return 0;

}