# 3D Graphics Programming Tools

## OpenGL 3D Drawing

Dr. Xianhui Cherry Che

x.che@qmul.ac.uk

# Learning Objectives

- Grasp 3D capabilities from extending the existing 2D knowledge and skills
- Draw a basic 3D shape using a case study
- Learn pre-built 3D objects from OpenGL

# Topics

- Draw a Cube
- GLUT Objects

# 3D Applications

- In OpenGL, two-dimensional applications are a special case of three-dimensional graphics
- Going to 3D
  - Not much changes
  - Use **`glVertex3*( )`**
  - Have to worry about the order in which polygons are drawn or use hidden-surface removal
  - Polygons should be simple, convex, flat





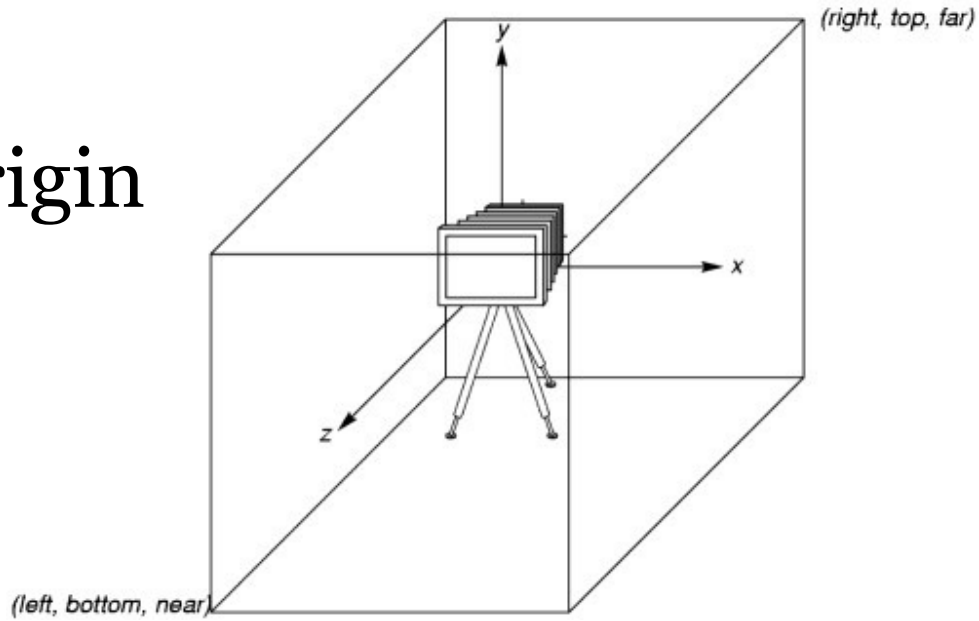


3

# OpenGL Camera

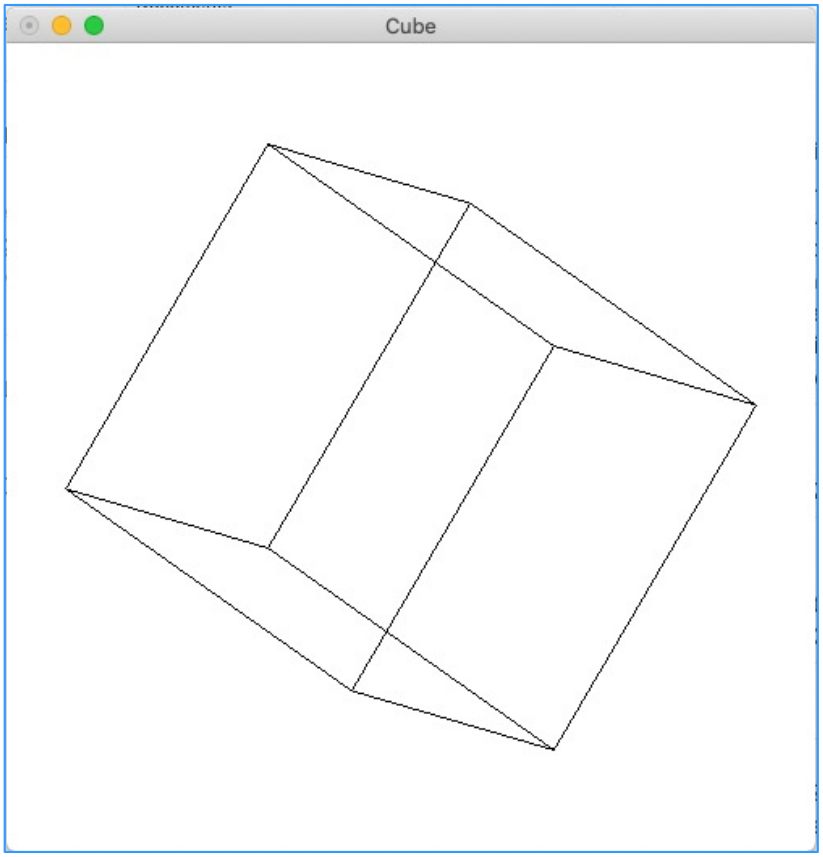
- OpenGL places a camera at the origin in object space pointing in the negative $z$ direction

- Remember: the default viewing volume is a box centered at the origin with a side of length 2



# Draw a Cube

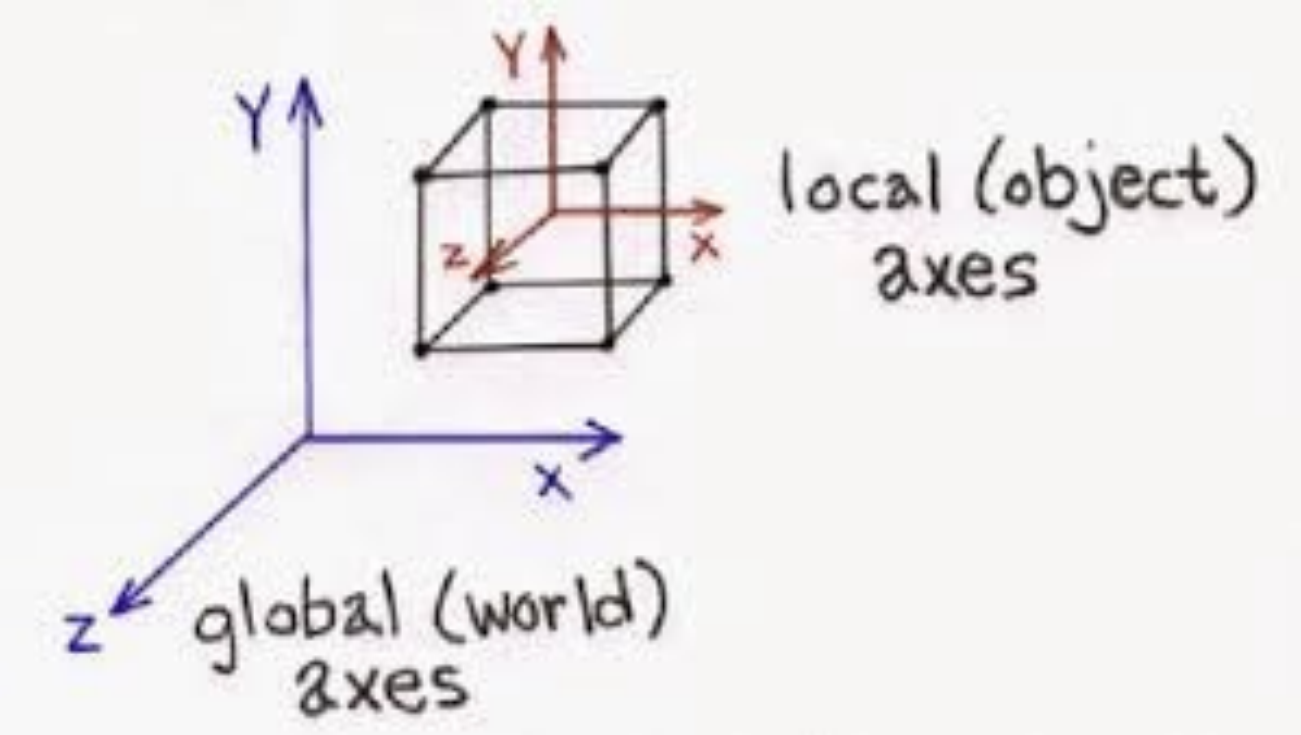
- Learn how to draw a basic 3D shape



# Draw a Cube – Part 1

Define vertices:

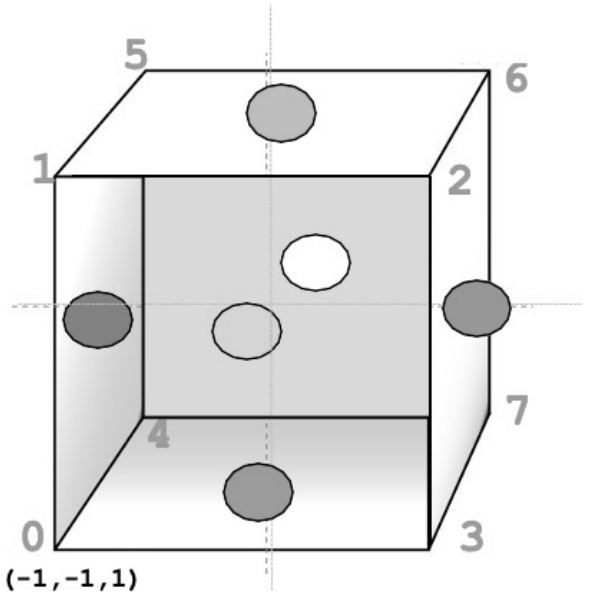
```
GLfloat CubeVertices[][3] = {{-1.0,-1.0,1.0}, {-1.0,1.0,1.0},
{1.0,1.0,1.0}, {1.0,-1.0,1.0}, {-1.0,-1.0,-1.0}, {-1.0,1.0,-
1.0}, {1.0,1.0,-1.0}, {1.0,-1.0,-1.0}};
```



# Draw a Cube – Part 2

```
void a3dpolygon(GLfloat vertices[][3],int a,int b,int c,int d)
{
    glBegin(GL_LINE_LOOP);
    glVertex3fv(vertices[a]);
    glVertex3fv(vertices[b]);
    glVertex3fv(vertices[c]);
    glVertex3fv(vertices[d]);
    glEnd();
}
```

← Draw a square




```
void cube(){
    a3dpolygon(CubeVertices, 0,3,2,1);
    a3dpolygon(CubeVertices, 2,3,7,6);
    a3dpolygon(CubeVertices, 3,0,4,7);
    a3dpolygon(CubeVertices, 1,2,6,5);
    a3dpolygon(CubeVertices, 4,5,6,7);
    a3dpolygon(CubeVertices, 5,4,0,1);
}
```

← Draw 6 facets

# Draw a Cube – Part 3

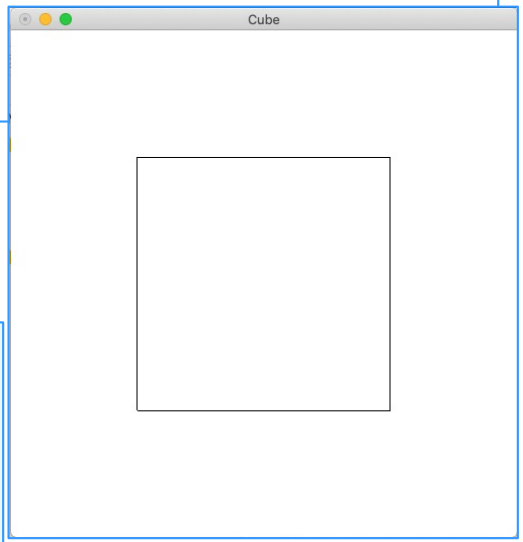
```
static GLfloat theta[] = {45.0,45.0,45.0};

void display(){
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glRotatef(theta[0], 1.0, 0.0, 0.0);
    glRotatef(theta[1], 0.0, 1.0, 0.0);
    glRotatef(theta[2], 0.0, 0.0, 1.0);
    cube();
    glFlush();
}
```

← Rotation angles

← Rotate the cube

*Without rotation, the view would be like this* →




**glRotate** produces a rotation of angle degrees. The current matrix is multiplied by a rotation matrix with the product replacing the current matrix.

# Draw a Cube – Part 4

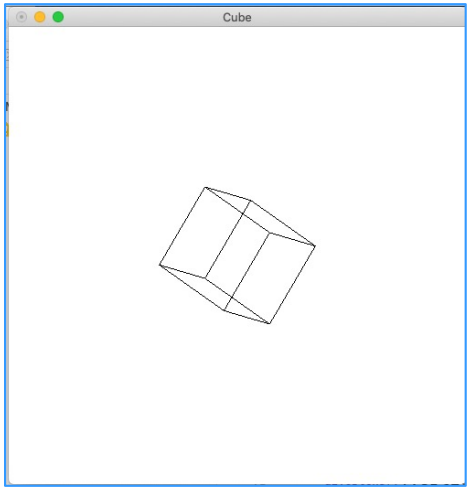
```
GLfloat x = 0.5;

void init(void) {
    int i, j;
    glClearColor(1.0, 1.0, 1.0, 1.0);
    glColor3f (0.0, 0.0, 0.0);
    for (j = 0; j < 3; j++)
        for (i = 0; i < 8; i++)
        CubeVertices[i][j] = CubeVertices[i][j]*x;
}
```

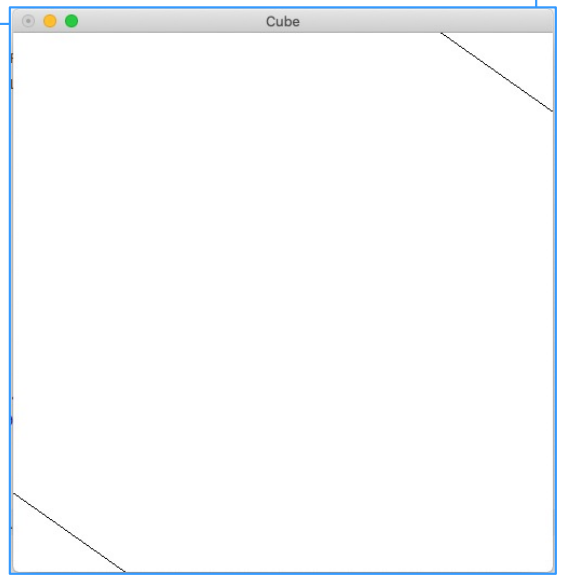
← A scaling factor

← Scale the vertices

*Without scaling the cube, it would be too big to view:* →




← *With a scaling factor of 0.2*



# Draw a Cube – Part 5

```
int main(int argc, char** argv){
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Cube");
    glutDisplayFunc(display);
    init();
    glEnable(GL_DEPTH_TEST);
    glutMainLoop();
}
```

The algorithm uses an extra buffer, the z-buffer, to store depth information as geometry travels down the pipeline.

It is cleared in the display callback:

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
```

# Draw a Cube – Full Program

```
#include <GLUT/glut.h>
GLfloat X = 0.5;
static GLfloat theta[] = {45.0,45.0,45.0};
GLfloat CubeVertices[][3] = {{-1.0,-1.0,1.0},{-1.0,1.0,1.0}, {1.0,1.0,1.0}, {1.0,-1.0,1.0},
{-1.0,-1.0,-1.0}, {-1.0,1.0,-1.0}, {1.0,1.0,-1.0},{1.0,-1.0,-1.0}};

void a3dpolygon(GLfloat vertices[][3],int a,int b,int c,int d) {
    glBegin(GL_LINE_LOOP);
    glVertex3fv(vertices[a]);
    glVertex3fv(vertices[b]);
    glVertex3fv(vertices[c]);
    glVertex3fv(vertices[d]);
    glEnd();
}
void cube(){
    a3dpolygon(CubeVertices, 0,3,2,1);
    a3dpolygon(CubeVertices, 2,3,7,6);
    a3dpolygon(CubeVertices, 3,0,4,7);
    a3dpolygon(CubeVertices, 1,2,6,5);
    a3dpolygon(CubeVertices, 4,5,6,7);
    a3dpolygon(CubeVertices, 5,4,0,1);
}
void display(){
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glRotatef(theta[0], 1.0, 0.0, 0.0);
    glRotatef(theta[1], 0.0, 1.0, 0.0);
    glRotatef(theta[2], 0.0, 0.0, 1.0);
    cube();
    glFlush();
}
```

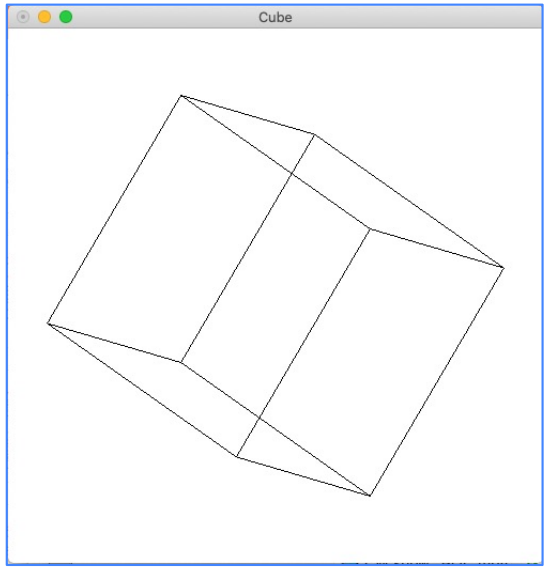
```
void init(void) {
    int i, j;
    glClearColor(1.0, 1.0, 1.0, 1.0);
    glColor3f (0.0, 0.0, 0.0);
    for (j = 0; j < 3; j++)
        for (i = 0; i < 8; i++)
        CubeVertices[i][j] = CubeVertices[i][j]*X;
}
int main(int argc, char** argv){
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB |
GLUT_DEPTH);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Cube");
    glutDisplayFunc(display);
    init();
    glEnable(GL_DEPTH_TEST);
    glutMainLoop();
}
```

# Topics

- Draw a Cube
- GLUT Objects

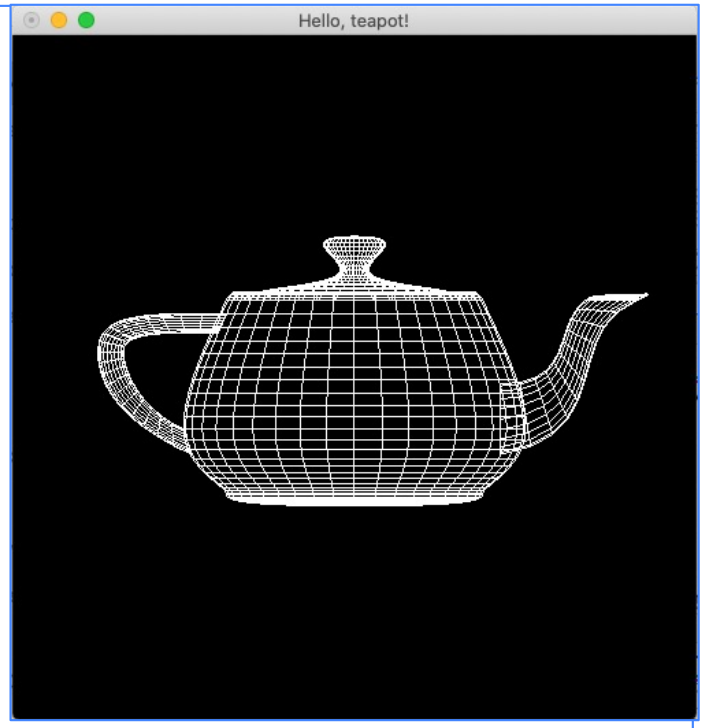
# Draw a Cube – An Easier Way!

```
#include <GLUT/glut.h>
static GLfloat theta[] = {45.0,45.0,45.0};
void display(){
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glRotatef(theta[0], 1.0, 0.0, 0.0);
    glRotatef(theta[1], 0.0, 1.0, 0.0);
    glRotatef(theta[2], 0.0, 0.0, 1.0);
    glutWireCube(1.0);          glutSolidCube();
    glFlush();}
void init(void) {
    glClearColor(1.0, 1.0, 1.0, 1.0);
    glColor3f (0.0, 0.0, 0.0);}
int main(int argc, char** argv){
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Cube");
    glutDisplayFunc(display);
    init();
    glEnable(GL_DEPTH_TEST);
    glutMainLoop();}
```



# Teapot ☺

```
#include <GLUT/glut.h>
void display () {
    glClear(GL_COLOR_BUFFER_BIT);
    glutWireTeapot(0.5);
    glFlush();
}
int main ( int argc, char * argv[] ) {
    glutInit(&argc,argv);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(0,0);
    glutInitDisplayMode(GLUT_RGB);
    glutCreateWindow("hello, teapot!");
    glutDisplayFunc(display);
    glutMainLoop();
}
```



# GLUT Objects

**11 Geometric Object Rendering**

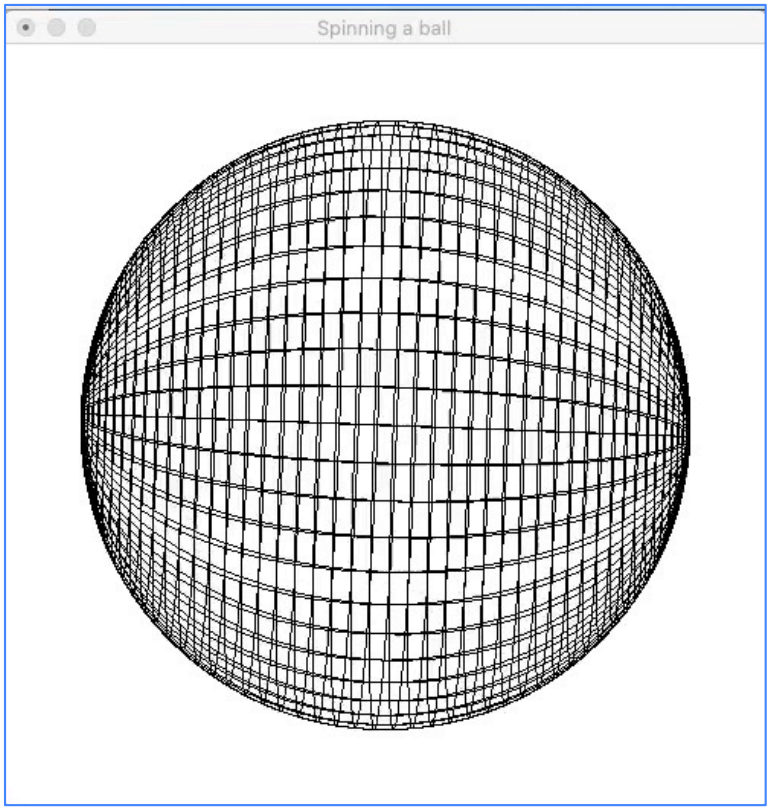


*Official reference document available at:*

*QMPlus: Background Reading – GLUT Documentation or*
https://www.opengl.org/resources/libraries/glut/spec3/spec3.html

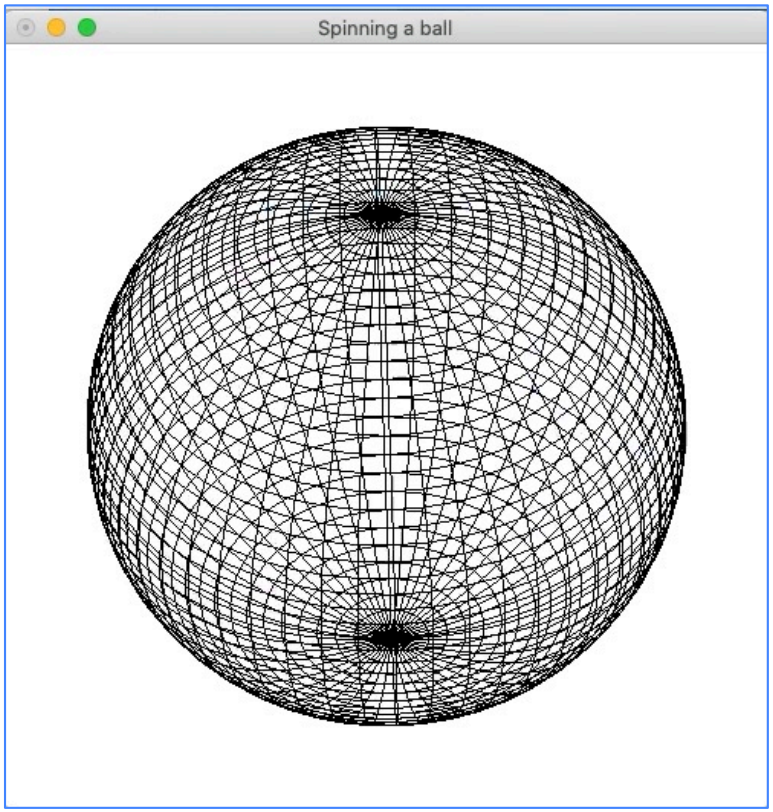
# Spin a Ball

```
#include <GLUT/glut.h>
GLfloat x = 45.0;
GLfloat y = 0;
GLfloat z = 45.0;
void idle() {
    y++;
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glRotatef(x, 1.0, 0.0, 0.0);
    glRotatef(y, 0.0, 1.0, 0.0);
    glRotatef(z, 0.0, 0.0, 1.0);
    glutPostRedisplay();
}
void display(){
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glutWireSphere(0.8, 50, 50);
    glutSwapBuffers ();
}
void init(void) {
    glClearColor(1.0, 1.0, 1.0, 1.0);
    glColor3f (0.0, 0.0, 0.0);
}
int main(int argc, char** argv){
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Spinning a ball");
    init();
    glutDisplayFunc(display);
    glutIdleFunc(idle);
    glEnable(GL_DEPTH_TEST);
    glutMainLoop();
}
```



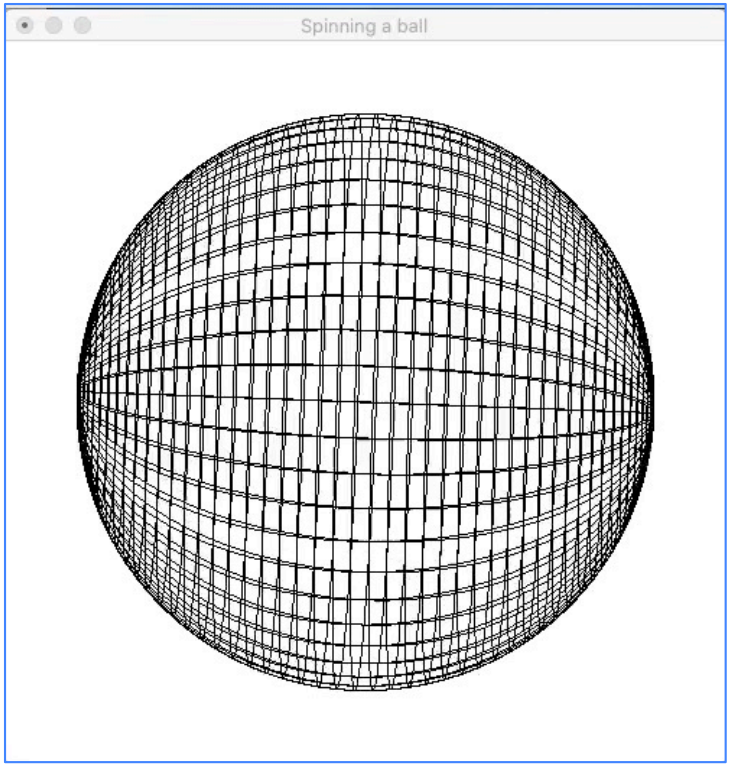

# Spin a Ball – Zoomed Out View

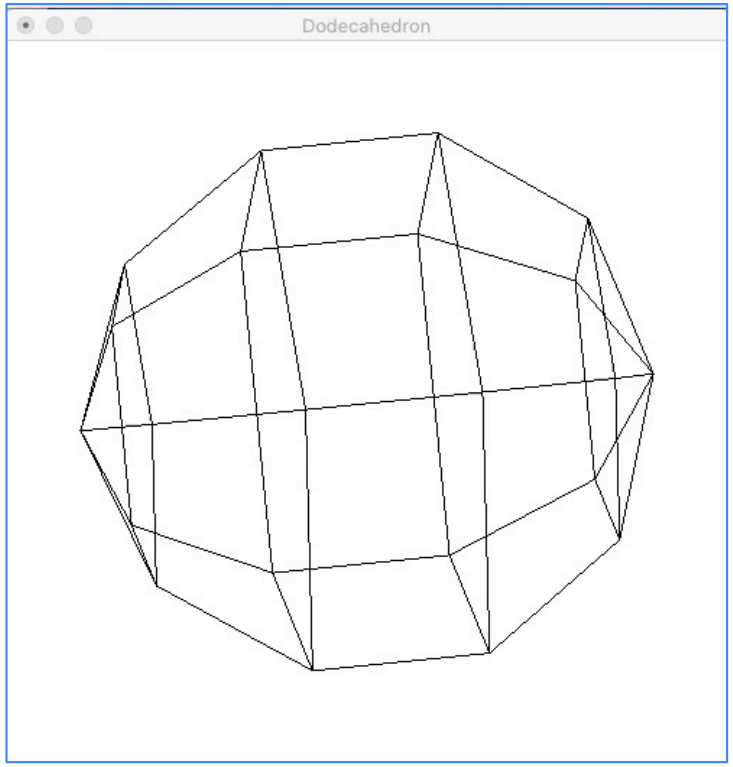
```
#include <GLUT/glut.h>
GLfloat x = 45.0;
GLfloat y = 0;
GLfloat z = 45.0;
GLdouble zoom = 1.0;
void idle() {
    y++;
    zoom += 0.02;
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluOrtho2D (-zoom, zoom, -zoom, zoom);
    glRotatef(x, 1.0, 0.0, 0.0);
    glRotatef(y, 0.0, 1.0, 0.0);
    glRotatef(z, 0.0, 0.0, 1.0);
    glutPostRedisplay();}
void display(){
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glutWireSphere(0.8, 50, 50);
    glutSwapBuffers ();}
void init(void) {
    glClearColor(1.0, 1.0, 1.0, 1.0);
    glColor3f (0.0, 0.0, 0.0);}
int main(int argc, char** argv){
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Spinning a ball");
    init();
    glutDisplayFunc(display);
    glutIdleFunc(idle);
    glEnable(GL_DEPTH_TEST);
    glutMainLoop();
}
```




# Curved Surface

- slices – The number of subdivisions around the Z axis (similar to longitude).
- Stacks – The number of subdivisions along the Z axis (similar to latitude).



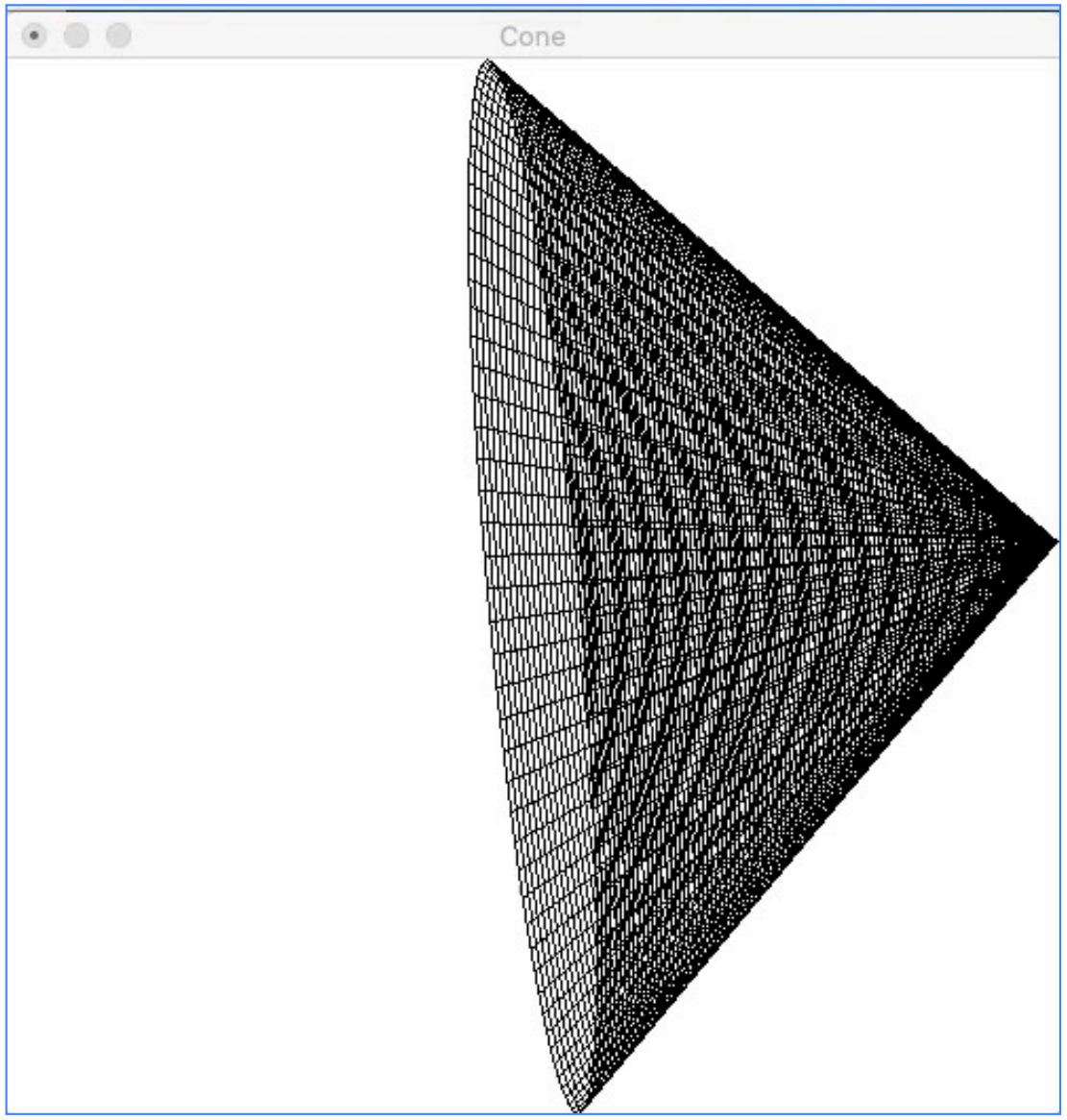

```
glutWireSphere(0.8, 50, 50);
```
(50 slices, 50 stacks)

```
glutWireSphere(0.8, 5, 5);
```
(5 slices, 5 stacks)

# GLUT Object – Cone

Specification:

- radius
- height
- slices
- stacks

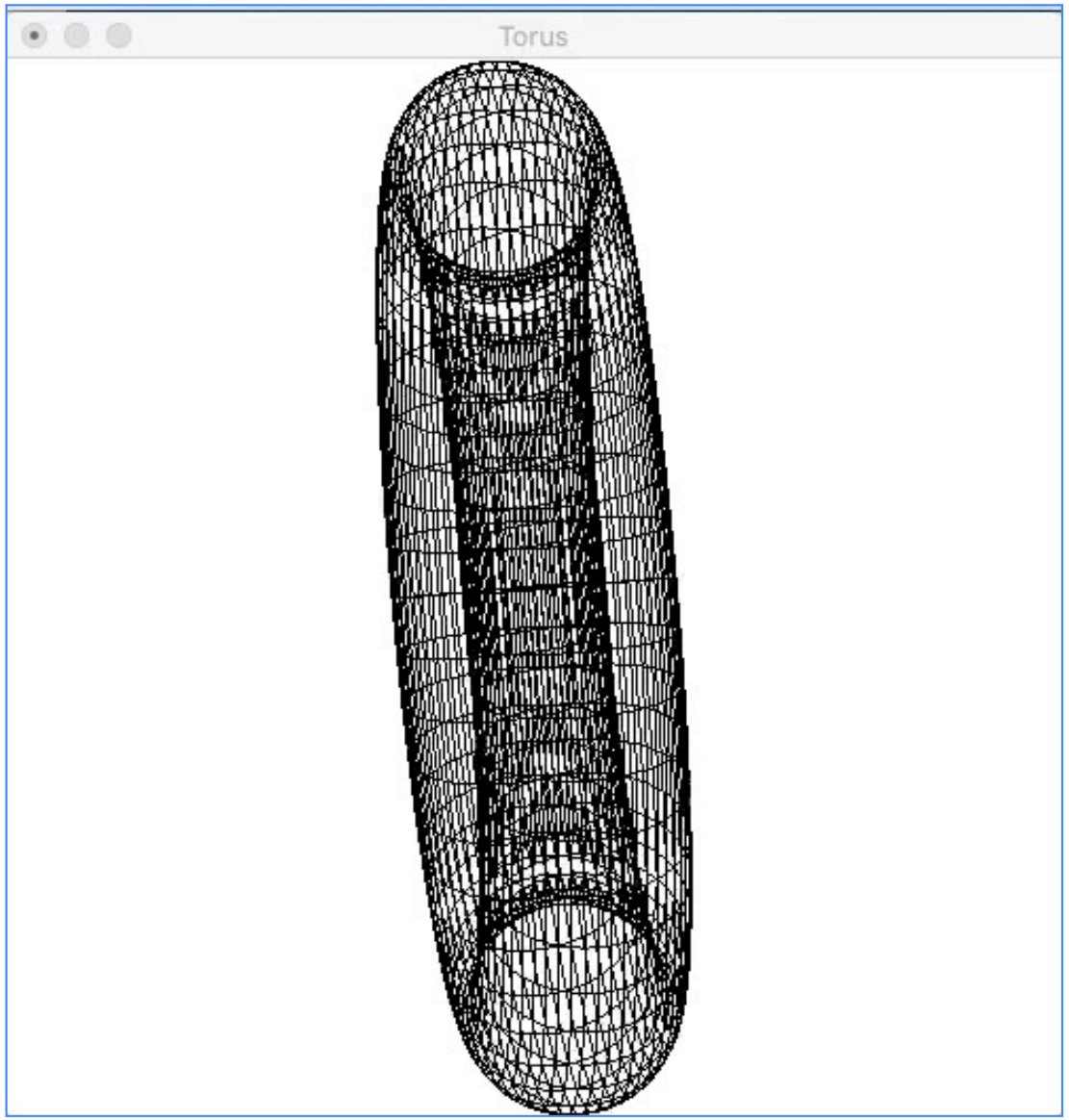



```
glutWireCone(1, 1, 100, 100);
```

# GLUT Object – Torus

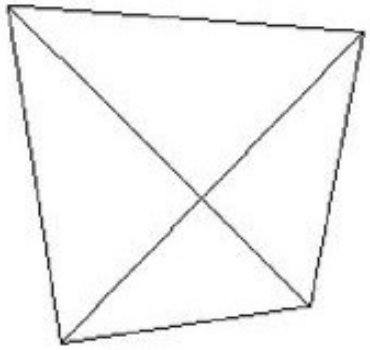
Specification:

- Inner radius
- Outer radius
- Sides
- Rings

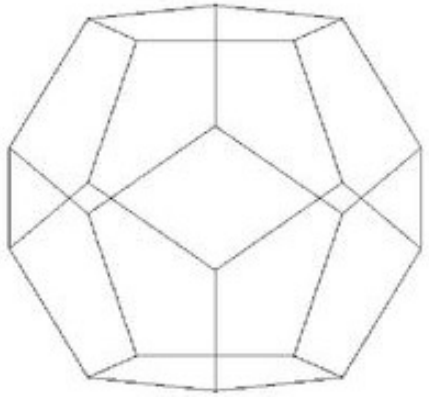



```
glutWireTorus(0.2, 0.8, 50, 50);
```

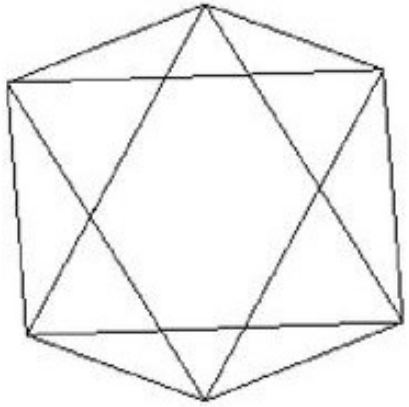
# GLUT Objects – Platonic



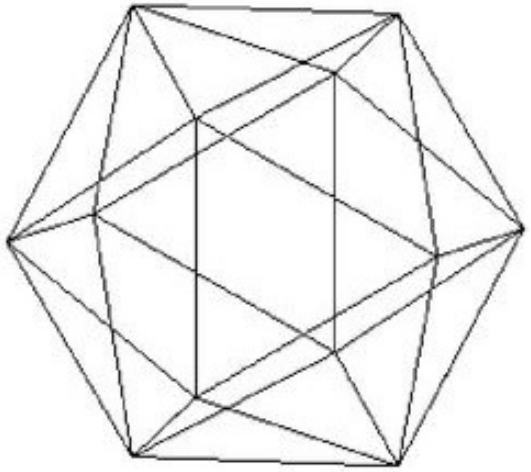
`glutWireTetrahedron()`



`glutWireDodecahedron()`



`glutWireOctahedron()`



`glutWireIcosahedron()`

# Questions?

x.che@qmul.ac.uk