Hochiminh city University of Technology
Faculty of Computer Science and Engineering

COMPUTER GRAPHICS

CHAPTER 03:

3D Object & Mesh

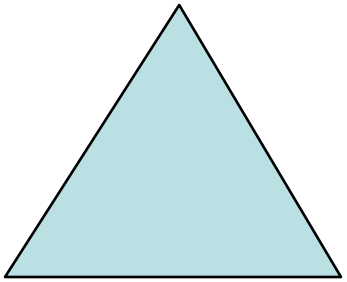
Trần Giang Sơn
tgson@hcmut.edu.vn

OUTLINE

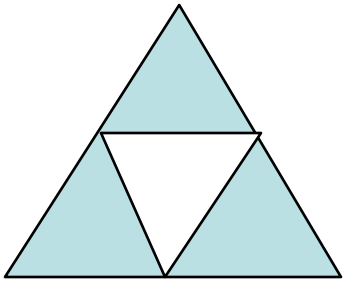
- Draw Sierpinski gasket by recursion
- Draw 3D Sierpinski gasket
- Hidden Surface
- Modeling Sphere
- Data Structure

The Sierpinski Gasket

- Start with a triangle



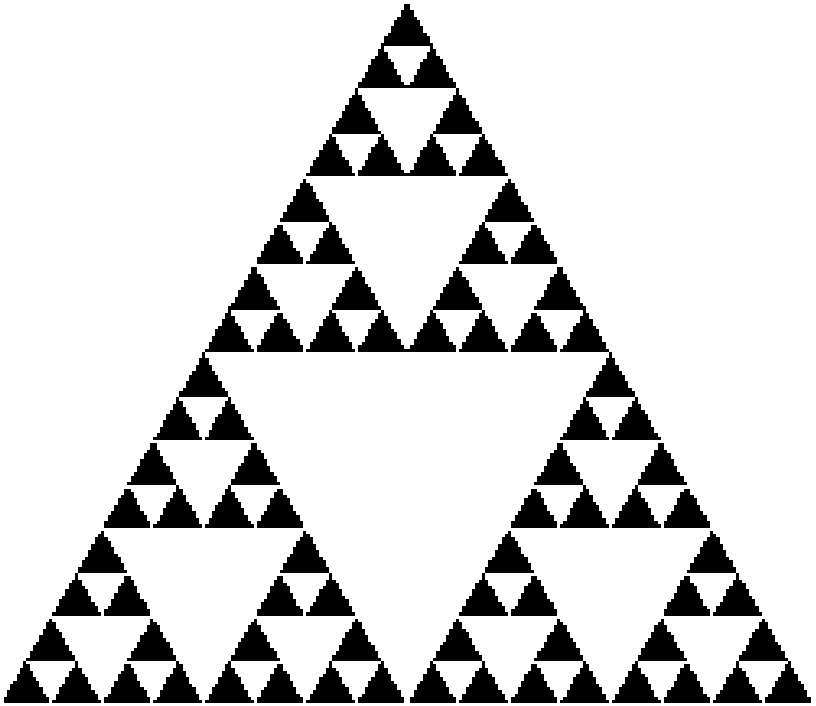
- Connect bisectors of sides and remove central triangle



- Repeat

The Sierpinski Gasket

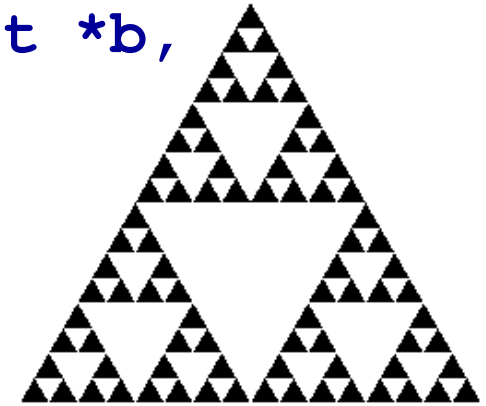
- Five subdivisions



The Sierpinski Gasket

```
GLfloat v[3][2]={{-1.0, -0.58},
                 {1.0, -0.58}, {0.0, 1.15}};
int n;

void triangle( GLfloat *a, GLfloat *b, GLfloat *c)
/* display one triangle  */
{
      glVertex2fv(a);
      glVertex2fv(b);
      glVertex2fv(c);
}
```

The Sierpinski Gasket



```
void divide_triangle(GLfloat *a, GLfloat *b,
     GLfloat *c, int m)
{
    point2 v0, v1, v2;
    int j;
    if(m>0){
        for(j=0; j<2; j++) v0[j]=(a[j]+b[j])/2;
        for(j=0; j<2; j++) v1[j]=(a[j]+c[j])/2;
        for(j=0; j<2; j++) v2[j]=(b[j]+c[j])/2;
        divide_triangle(a, v0, v1, m-1);
        divide_triangle(c, v1, v2, m-1);
        divide_triangle(b, v2, v0, m-1);
    }
    else(triangle(a,b,c));
}
```

The Sierpinski Gasket

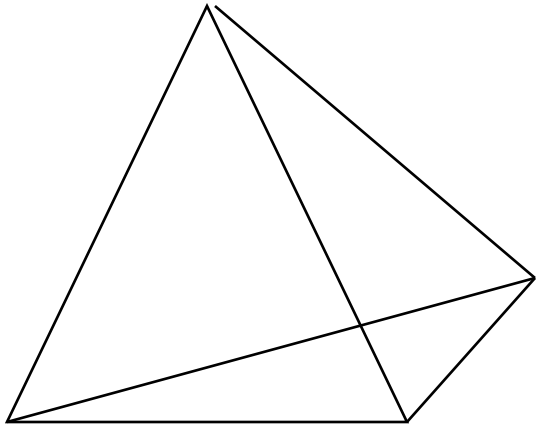
```
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_TRIANGLES);
       divide_triangle(v[0], v[1], v[2], n);
    glEnd();
    glFlush();
}

void myinit()
{
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-2.0, 2.0, -2.0, 2.0);
    glMatrixMode(GL_MODELVIEW);
    glClearColor (1.0, 1.0, 1.0,1.0)
    glColor3f(0.0,0.0,0.0);
}
```

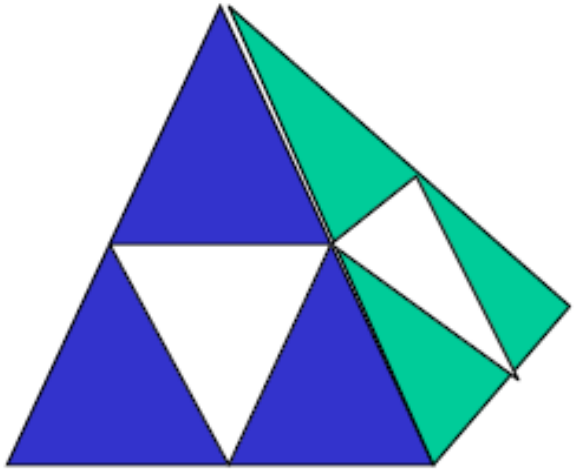
The Sierpinski Gasket

```
int main(int argc, char **argv)
{
    n=4;
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutCreateWindow("2D Gasket");
    glutDisplayFunc(display);
    myinit();
    glutMainLoop();
}
```

The Sierpinski Gasket

- We can subdivide each of the four faces





- Appears as if we remove a solid tetrahedron from the center leaving four smaller tetrahedra

The Sierpinski Gasket

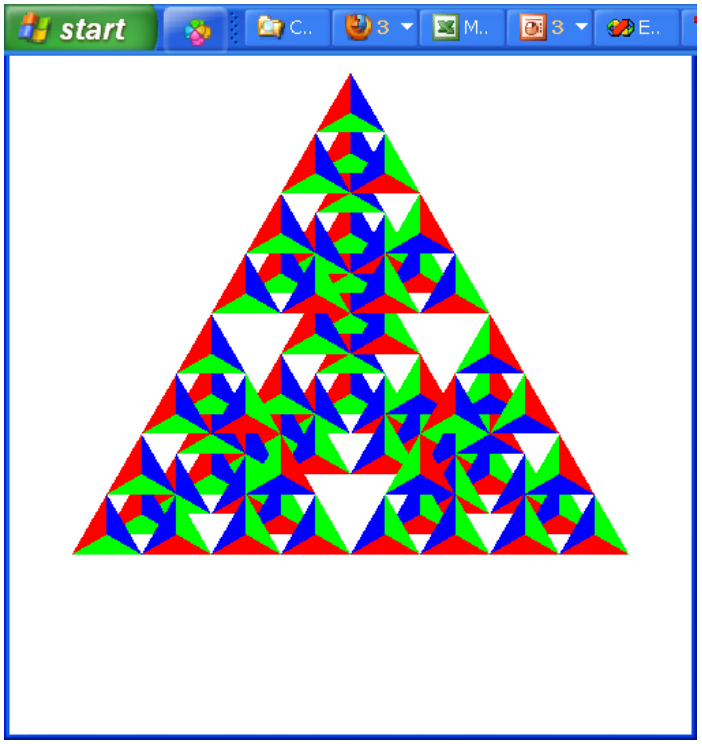
```
void triangle( GLfloat *a, GLfloat *b, GLfloat *c){
  glVertex3fv(a);
  glVertex3fv(b);
  glVertex3fv(c);
}
void tetra(GLfloat *a, GLfloat *b, GLfloat *c, GLfloat *d){
   glColor3fv(colors[0]);
   triangle(b, d, c);
   glColor3fv(colors[1]);
   triangle(a, b, c);
   glColor3fv(colors[2]);
   triangle(a, c, d);
   glColor3fv(colors[3]);
   triangle(a, d, b);
}
```

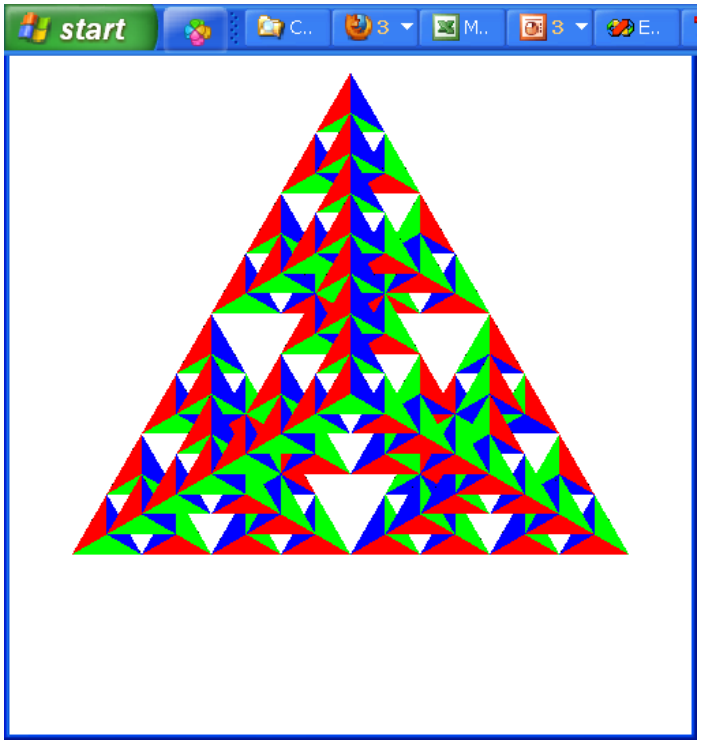
The Sierpinski Gasket

```
void divide_tetra(GLfloat *a, GLfloat *b, GLfloat *c, GLfloat *d, int m){
    GLfloat mid[6][3];
    int j;
    if(m>0)    {
        for(j=0; j<3; j++) mid[0][j]=(a[j]+b[j])/2;
        ……….
        divide_tetra(a, mid[0], mid[1], mid[2], m-1);
        ……….
    }
    else(tetra(a,b,c,d)); /* draw tetrahedron at end of recursion */
}
```

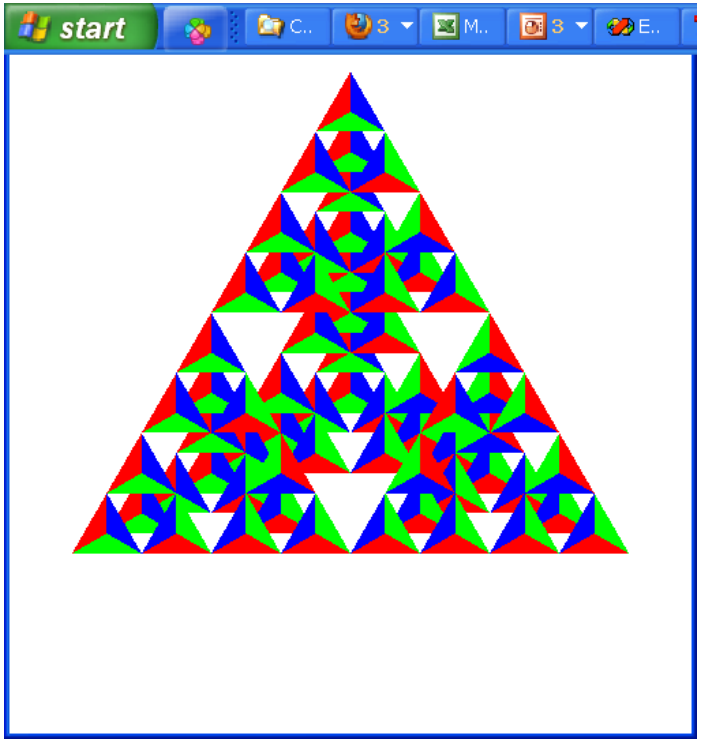
The Sierpinski Gasket

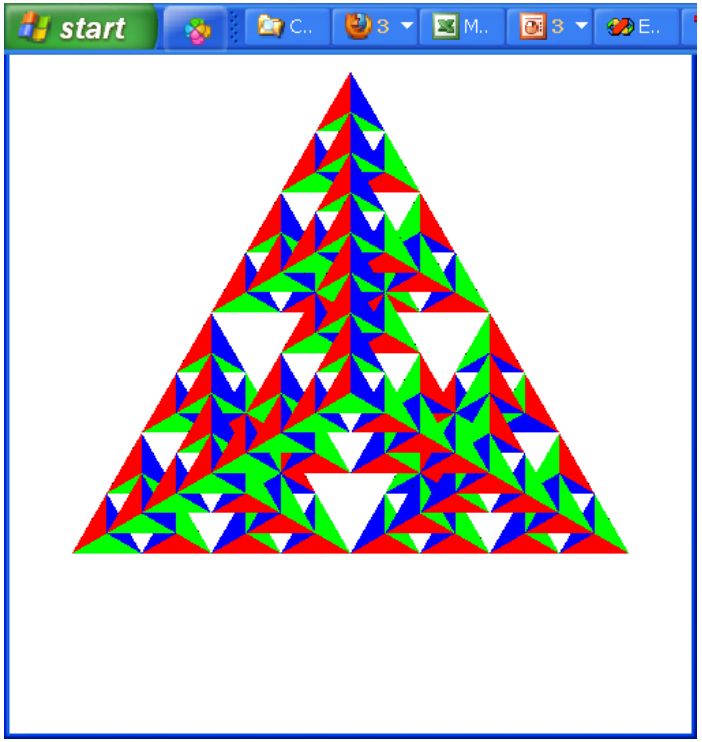
- Because the triangles are drawn in the order they are defined in the program, the front triangles are not always rendered in front of triangles behind them





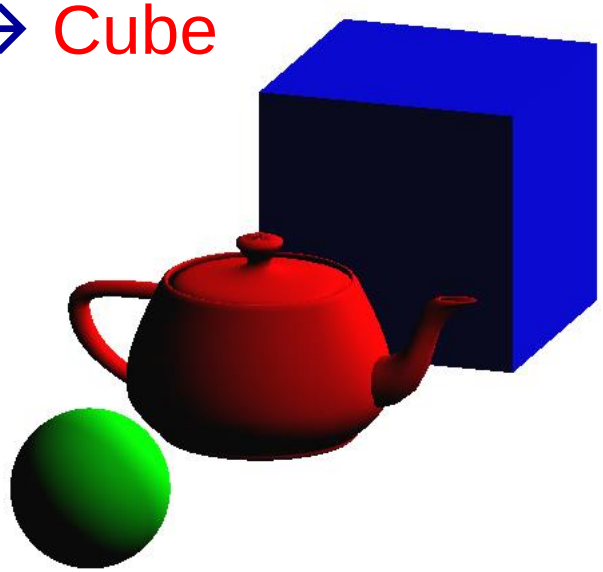
Hidden-Surface Removal





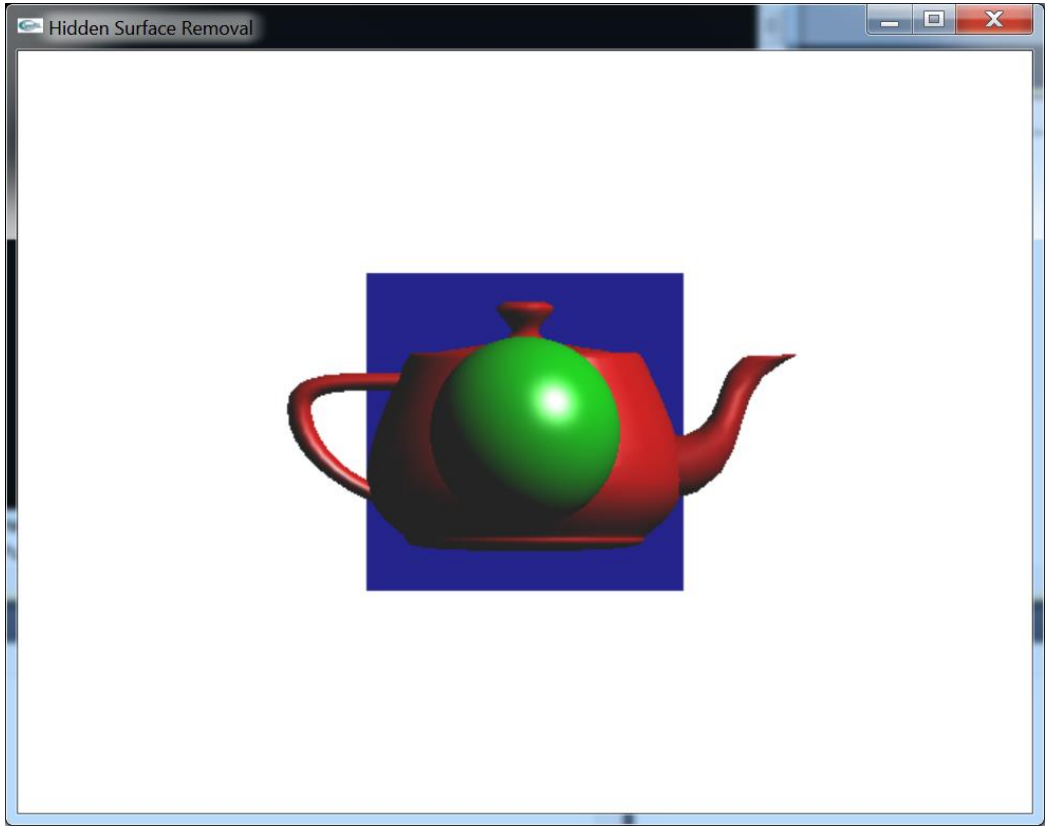
Hidden-Surface Removal

- Camera : (0, 0, 20)
- Teapot : (0, 0, -1)
- Sphere: (0, 0, 1)
- Cube: (0, 0, -3)
- Correct Order: Camera→Sphere→Teapot→Cube
- Draw Order : Teapot→Sphere → Cube



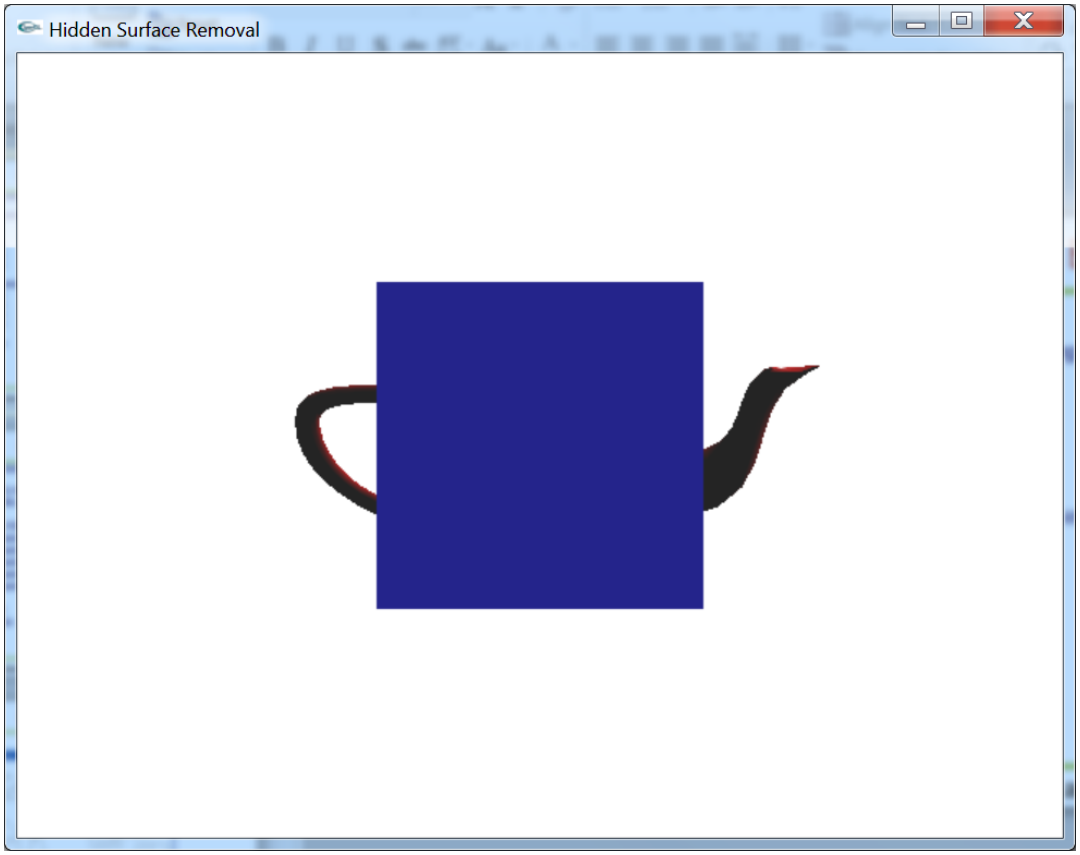
Hidden-Surface Removal

- Use Depth Buffer



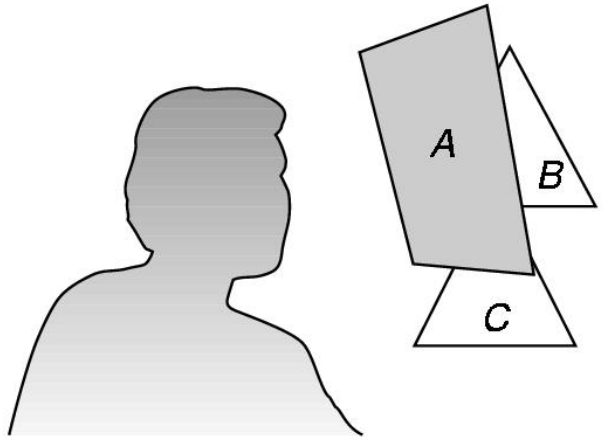
Hidden-Surface Removal

- Don’t use depth buffer



Hidden-Surface Removal

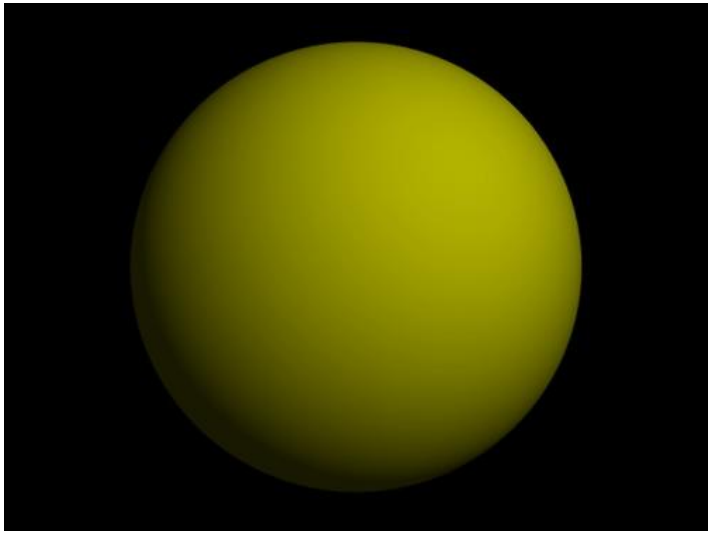
- We want to see only those surfaces in front of other surfaces
- OpenGL uses a *hidden-surface* method called the *z*-buffer algorithm that saves depth information as objects are rendered so that only the front objects appear in the image

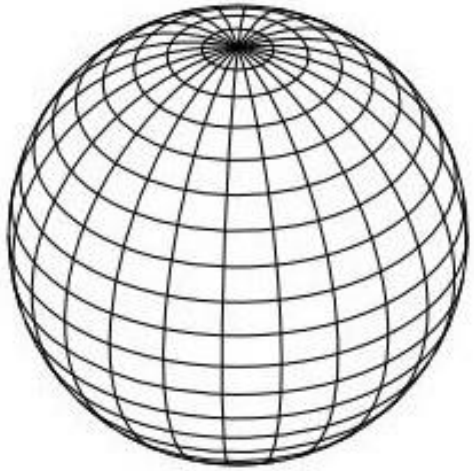


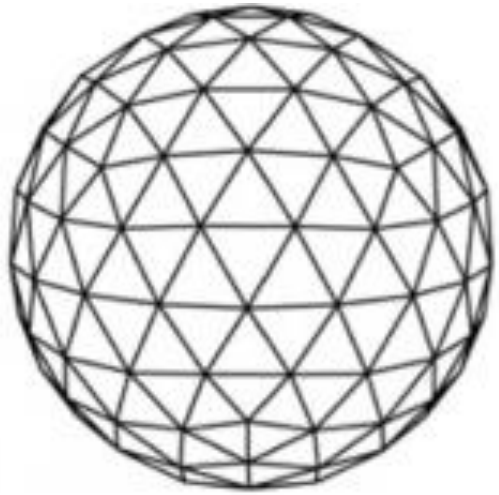
Hidden-Surface Removal

- The algorithm uses an extra buffer, the z-buffer, to store depth information as geometry travels down the pipeline
- It must be
  - Requested in **`main()`**
    - **`glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH)`**
  - Enabled
    - **`glEnable(GL_DEPTH_TEST)`**
  - Cleared in the display callback
    - **`glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)`**

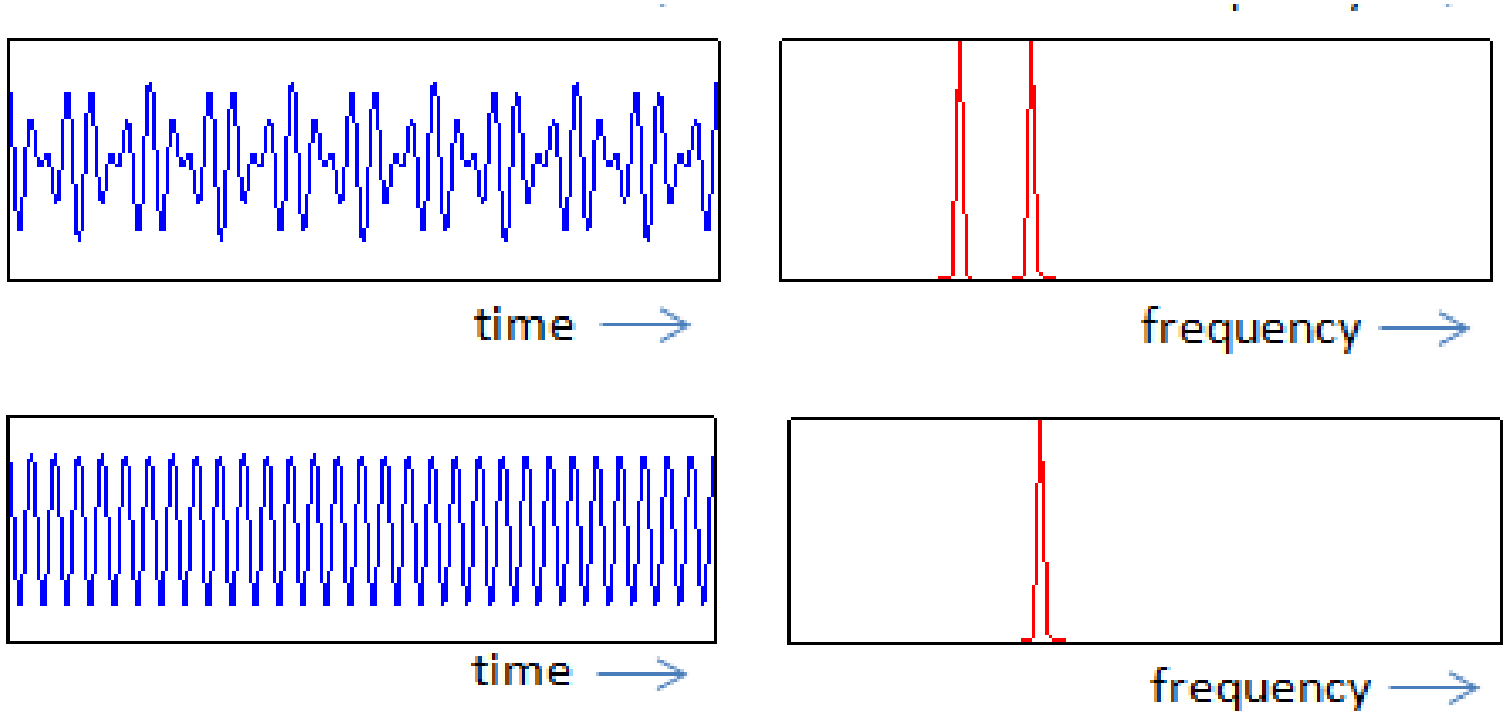
Modeling Sphere





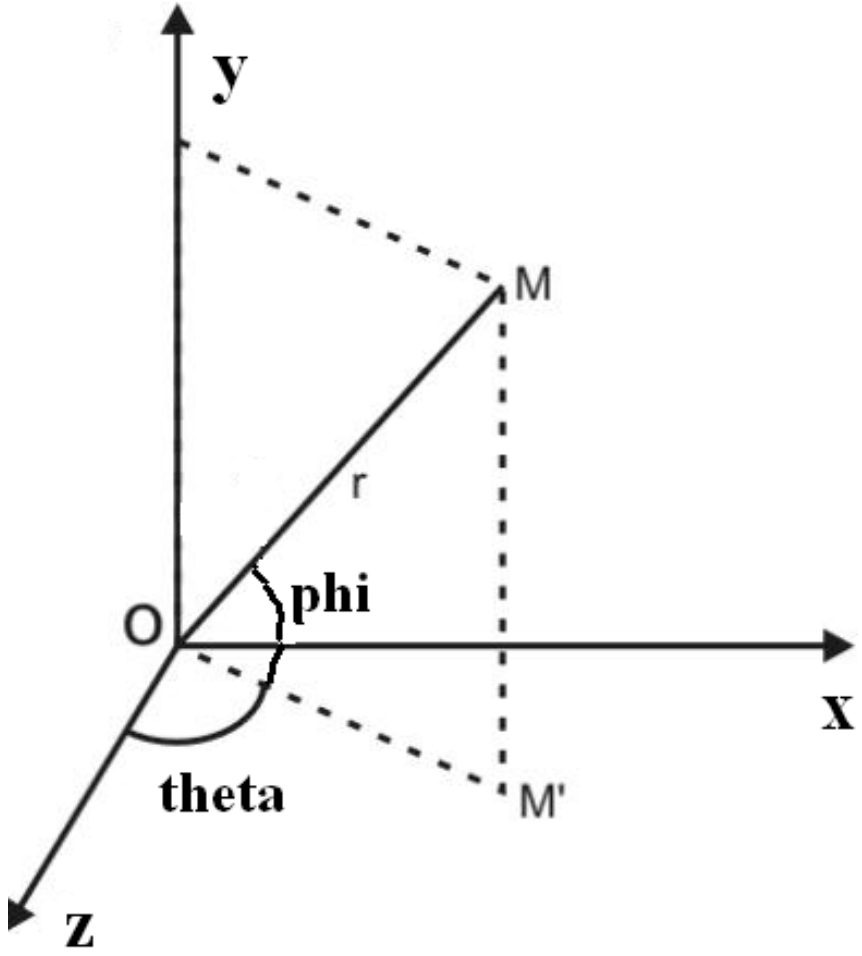


Modeling Sphere

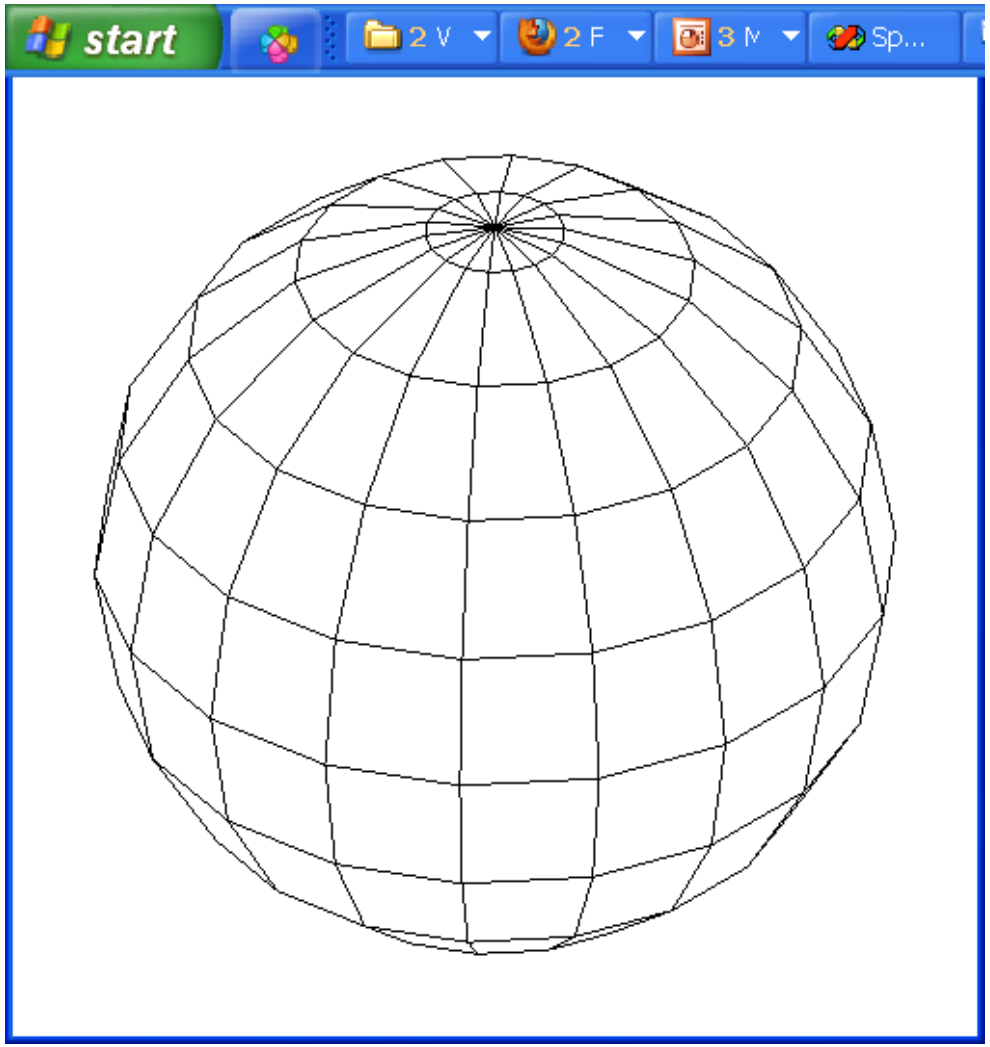


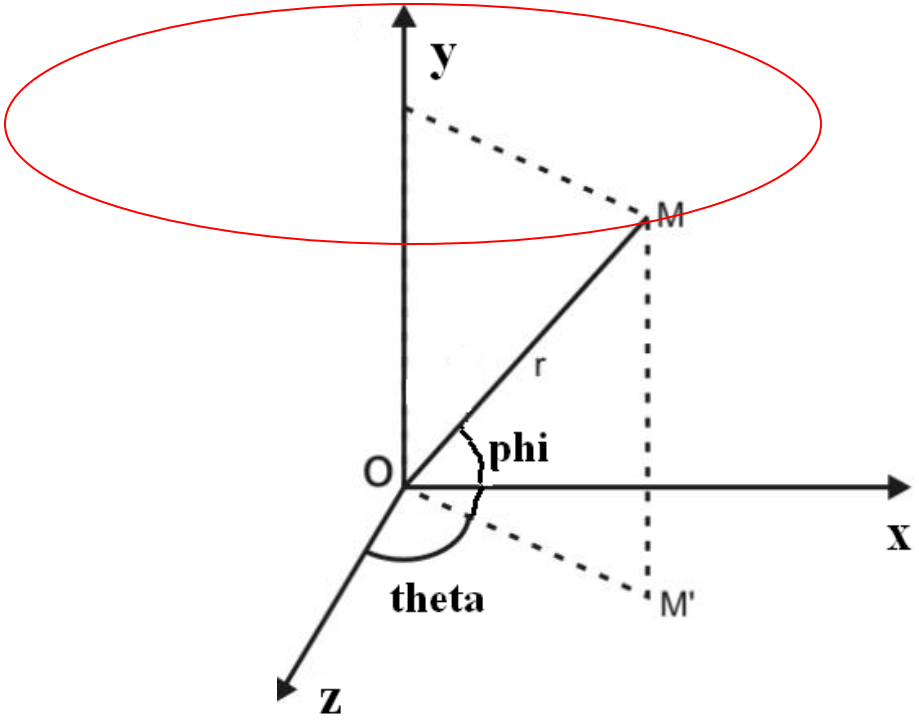
Modeling Sphere

- **Spherical coordinate system**
  - x = r*sin(theta)*cos(phi);
  - z = r*cos(theta)*cos(phi);
  - y = r*sin(phi);



Modeling Sphere



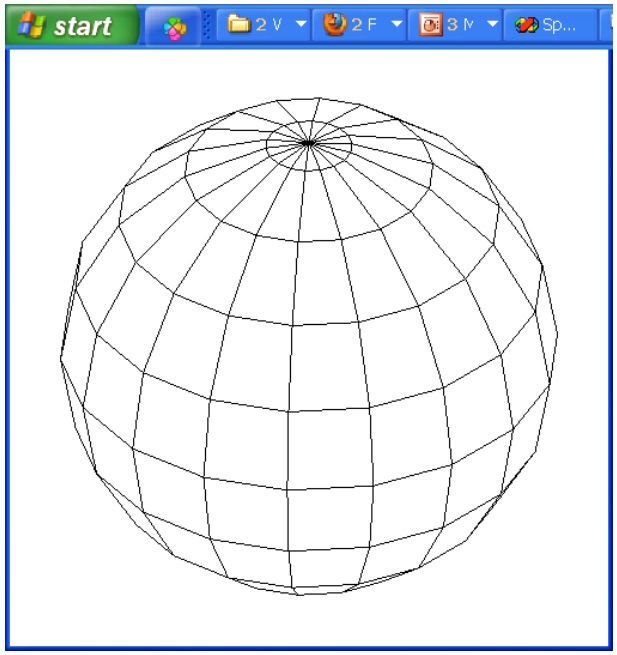


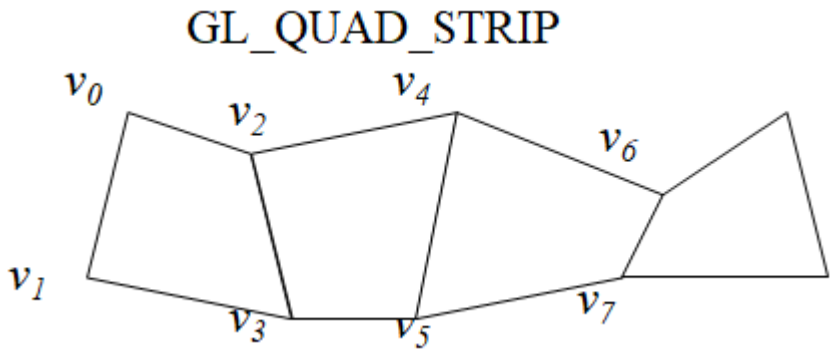
Phi : -90 → 90

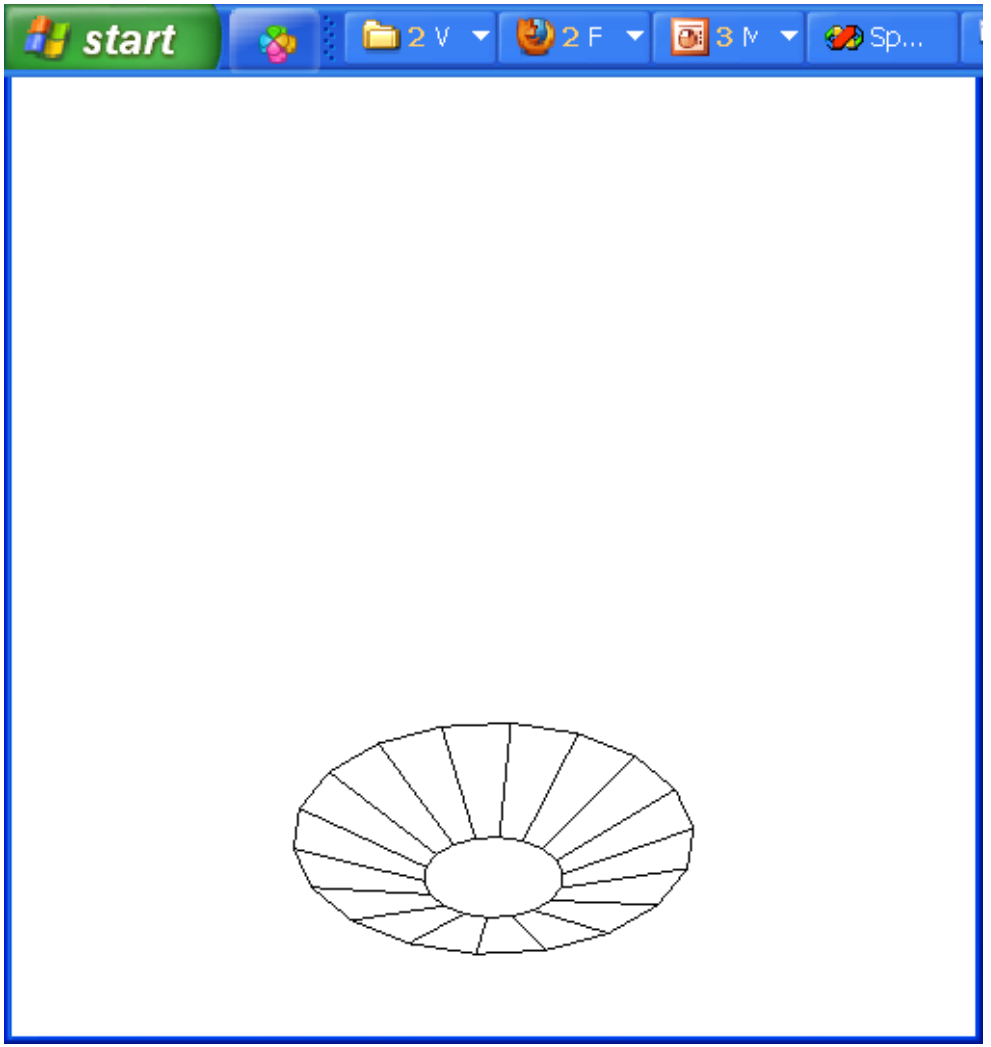
Theta: 0 → 360

Theta: -180 → 180

Modeling Sphere

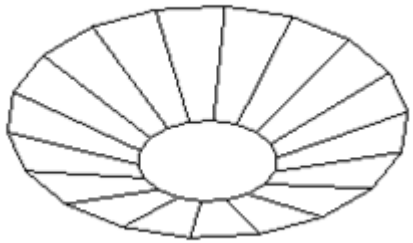






Modeling Sphere

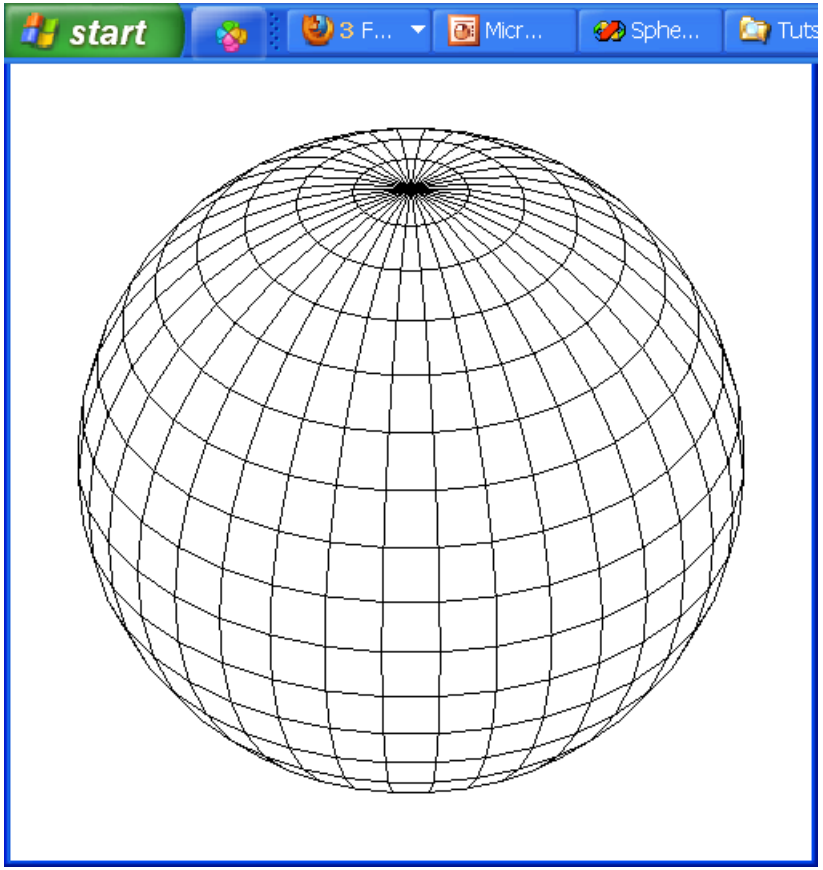
```
for(float phi = -80; phi<=80; phi+=20){
        phir = c*phi;
        phir20 = c*(phi+20);
        glBegin(GL_QUAD_STRIP);
        for(float theta = -180; theta<=180; theta+=20)        {
                thetar = c*theta;
                x = sin(thetar)*cos(phir); z = cos(thetar)*cos(phir);
                y = sin(phir);
                glVertex3d(x, y, z);
                x = sin(thetar)*cos(phir20);z = cos(thetar)*cos(phir20);
                y = sin(phir20);
                glVertex3d(x, y, z);
        }
        glEnd();
    }
```



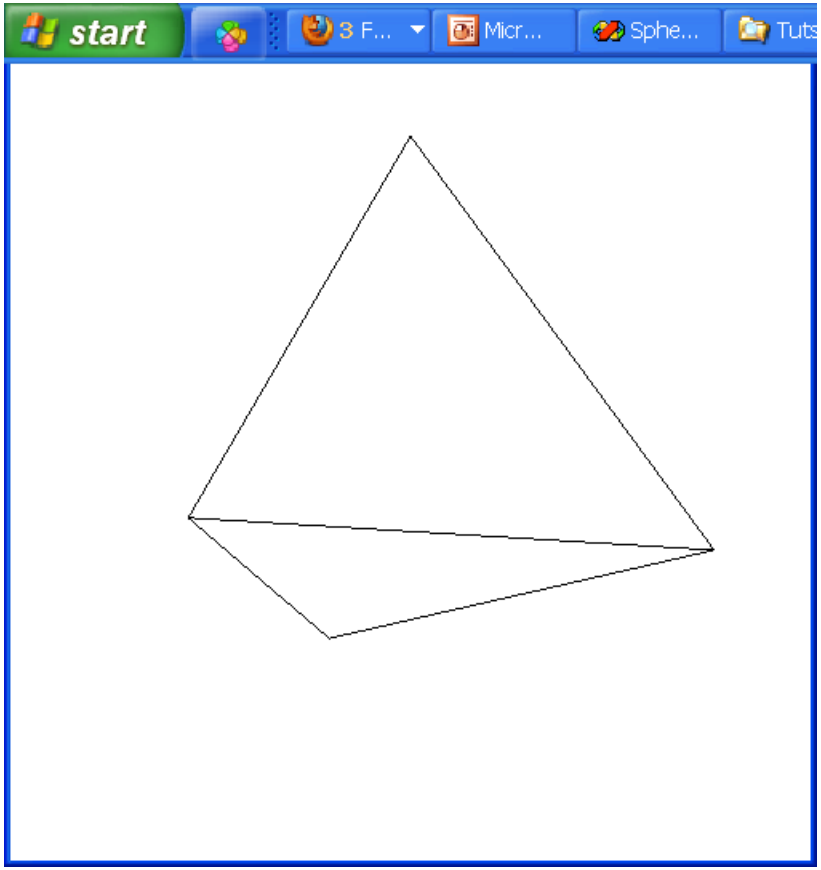
Modeling Sphere

```
glBegin(GL_TRIANGLE_FAN);
        glVertex3d(0, 1, 0);
        c80 = c*80;
        y = sin(c80);
        for(float theta = 180; theta>=-180; theta-=20)
        {
                thetar = c*theta;
                x = sin(thetar)*cos(c80);
                z = cos(thetar)*cos(c80);
                glVertex3d(x, y, z);
        }
glEnd();
```

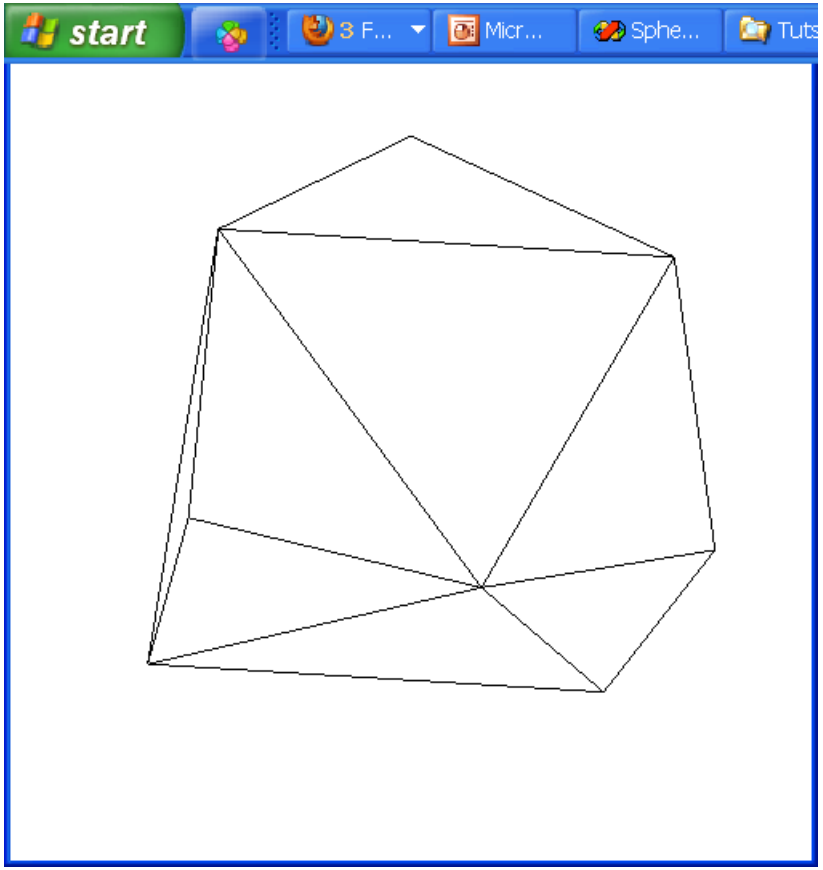
Modeling Sphere



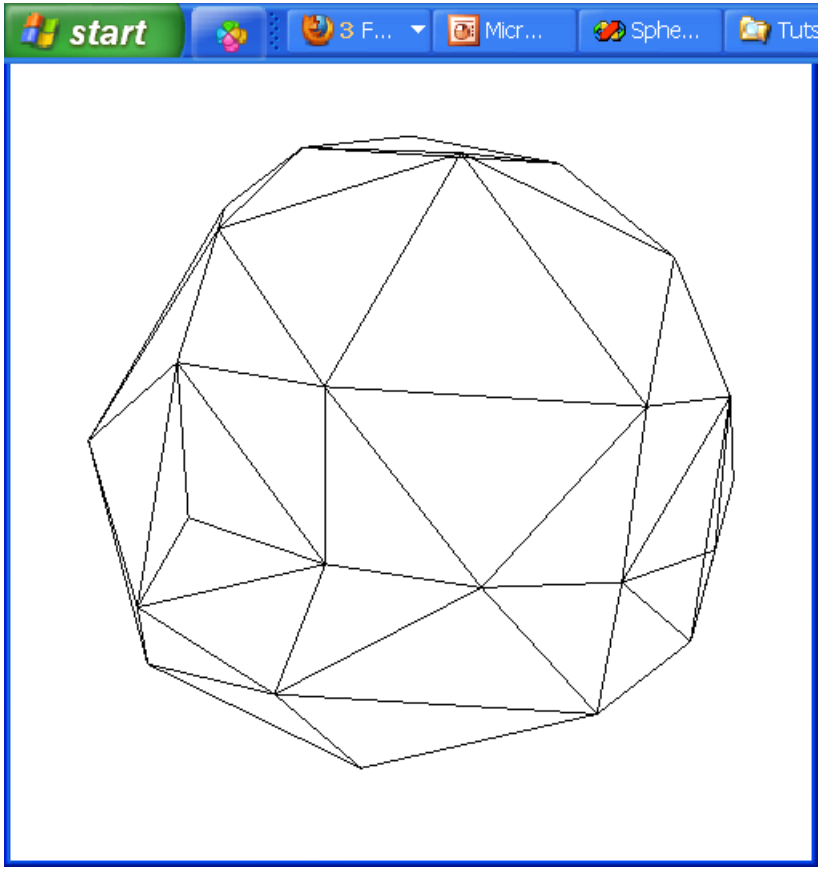
Modeling Sphere



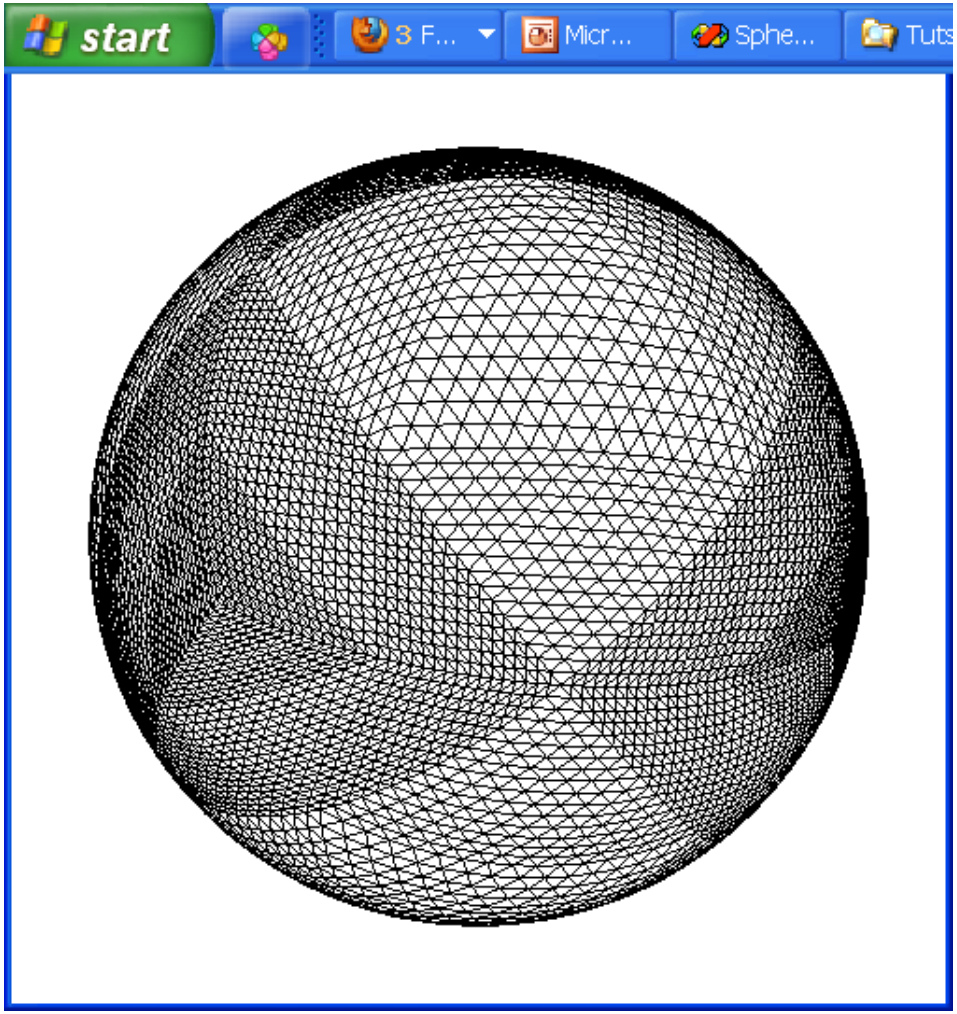
Modeling Sphere



Modeling Sphere

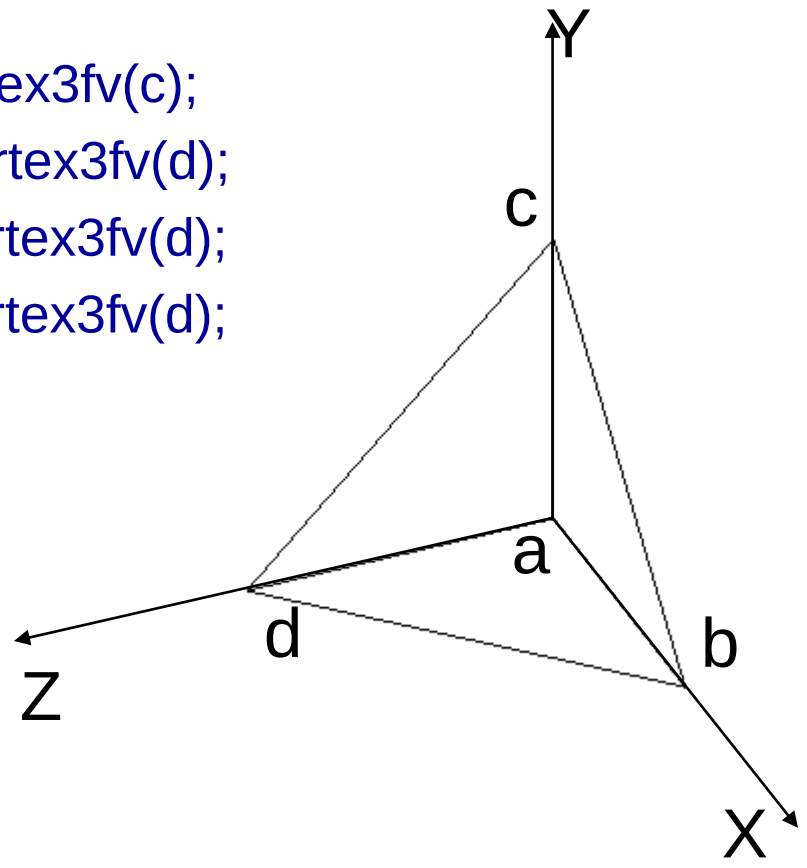


Modeling Sphere



Data Structure

```
GLfloat a[3] = { 0, 0, 0 },b[3] = { 1, 0, 0 };
GLfloat c[3] = { 0, 1, 0 },d[3] = { 0, 0, 1 };
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
glBegin(GL_TRIANGLES);
        glVertex3fv(a);glVertex3fv(b);glVertex3fv(c);
        glVertex3fv(a); glVertex3fv(b); glVertex3fv(d);
        glVertex3fv(a); glVertex3fv(c); glVertex3fv(d);
        glVertex3fv(b); glVertex3fv(c); glVertex3fv(d);
glEnd();
```



Data Structure

```
struct Face {
        int             numVerts;
        Point3D         *pointArr;
};
class Mesh  {
        int             numFaces;
        Face            *faceArr;
        void            DrawWireframe() ;
        void            DrawColor();
};
```

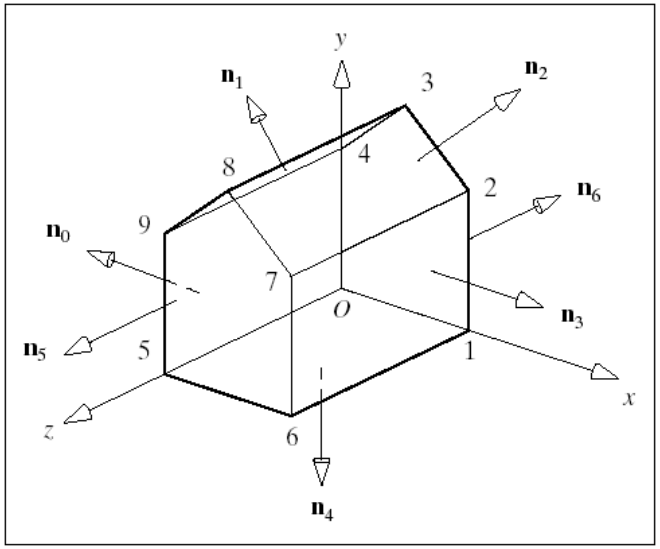
Modeling Shapes with Polygonal Meshes

- **Defining a Polygonal Mesh**

  - A more efficient approach uses three separate lists : a vertex list, a normal list, and a face list

  - The three lists work together : The vertex list contains locational or geometric information, the normal list contains orientation information, and the face list contains connectivity or topological information.

Modeling Shapes with Polygonal Meshes



face	vertices	associated normal
0 (left)	0,5,9,4	0,0,0,0
1 (roof left)	3,4,9,8	1,1,1,1
2 (roof right)	2,3,8,7	2,2,2,2
3 (right)	1,2,7,6	3,3,3,3
4 (bottom)	0,1,6,5	4,4,4,4
5 (front)	5,6,7,8,9	5,5,5,5,5
6 (back)	0,4,3,2,1	6,6,6,6,6

vertex	x	y	z
0	0	0	0
1	1	0	0
2	1	1	0
3	0.5	1.5	0
4	0	1	0
5	0	0	1
6	1	0	1
7	1	1	1
8	0.5	1.5	1
9	0	1	1

normal	n_x	n_y	n_z
0	−1	0	0
1	−0.707	0.707	0
2	1	0	0
3	0	−1	0
4	0	0	1
5	0	0	−1
6	0.707	0.707	0

Modeling Shapes with Polygonal Meshes

```
class VertexID{
    public:
        int   vertIndex; //index of this vertex in the vertex list
        int   normIndex; // index of this vertex's normal
};
class Face{
    public:
      int           nVerts; // number of vertice in this face
      VertexID*  vert; // the list of vertex and normal index
      Face() { nVerts = 0; vert = NULL; }
      ~Face() { delete[] vert; nVerts = 0; }
 };
```

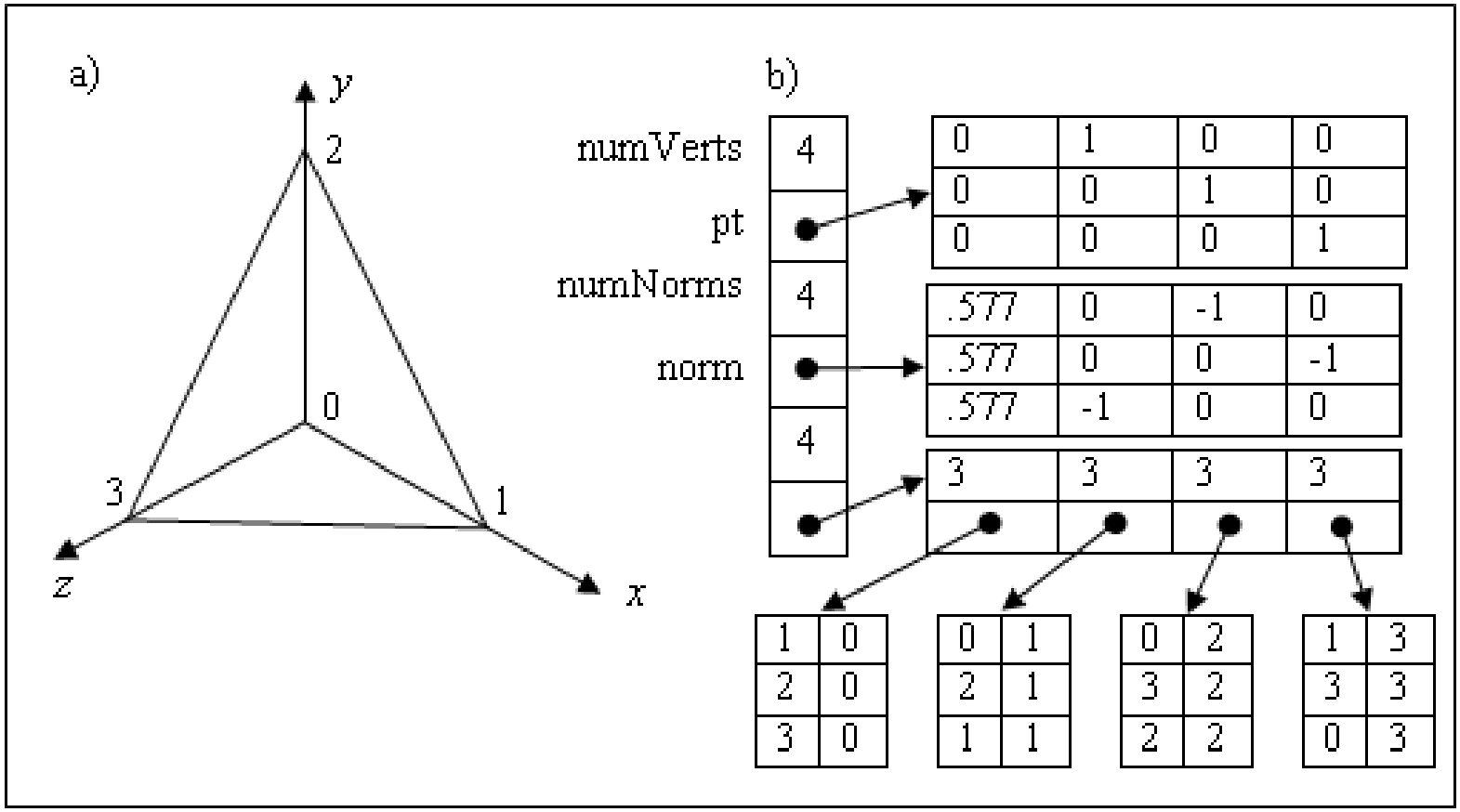
Modeling Shapes with Polygonal Meshes

```
class Mesh {
    private:
        int         numVerts; // number of vertices in the mesh
        Point3*     pt;   // array of 3D vertices
        int         numNormals; // number of normal vectors for the mesh
        Vector3*    norm; // array of normals
        int         numFaces; // number of faces in the mesh
        Face*       face; // array of face data
        // ... others to be added later
     public:
        Mesh();
        ~Mesh();
      // ... others
};
```

Modeling Shapes with Polygonal Meshes

```
void Mesh::DrawWireframe(){
    glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
    for (int f = 0; f < numFaces; f++) {
        glBegin(GL_POLYGON);
        for (int v = 0; v < face[f].nVerts; v++){
            int             iv = face[f].vert[v].vertIndex;
            glVertex3f(pt[iv].x, pt[iv].y, pt[iv].z);
        }
        glEnd();
    }
}
```

Modeling Shapes with Polygonal Meshes



Modeling Shapes with Polygonal Meshes

```
void Mesh::CreateTetrahedron()
{
    numVerts=4;
    pt = new Point3[numVerts];
    pt[0].set(0, 0, 0);
    pt[1].set(1, 0, 0);
    pt[2].set(0, 1, 0);
    pt[3].set(0, 0, 1);
```

Modeling Shapes with Polygonal Meshes

```
numFaces= 4;
face = new Face[numFaces];

face[0].nVerts = 3;
face[0].vert = new VertexID[face[0].nVerts];
face[0].vert[0].vertIndex = 1;
face[0].vert[1].vertIndex = 2;
face[0].vert[2].vertIndex = 3;
face[0].vert[0].normIndex = 0;
face[0].vert[1].normIndex = 0;
face[0].vert[2].normIndex = 0;
```