Programming with OpenGL
Part 3: Three Dimensions

Ed Angel

Professor of Computer Science, Electrical and Computer Engineering, and Media Arts

University of New Mexico

Objectives

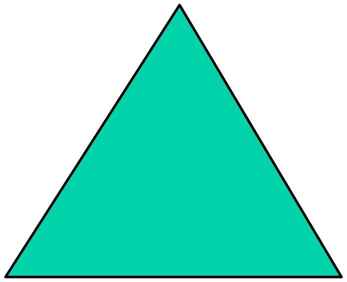
- Develop a more sophisticated three-dimensional example
  - Sierpinski gasket: a fractal
- Introduce hidden-surface removal

Three-dimensional Applications

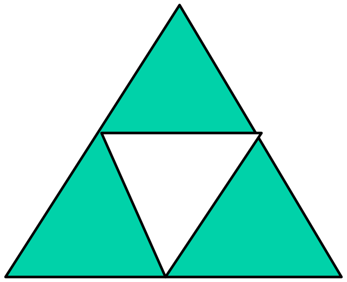
- In OpenGL, two-dimensional applications are a special case of three-dimensional graphics
- Going to 3D
  - Not much changes
  - Use **`glVertex3*( )`**
  - Have to worry about the order in which polygons are drawn or use hidden-surface removal
  - Polygons should be simple, convex, flat

Sierpinski Gasket (2D)

- Start with a triangle



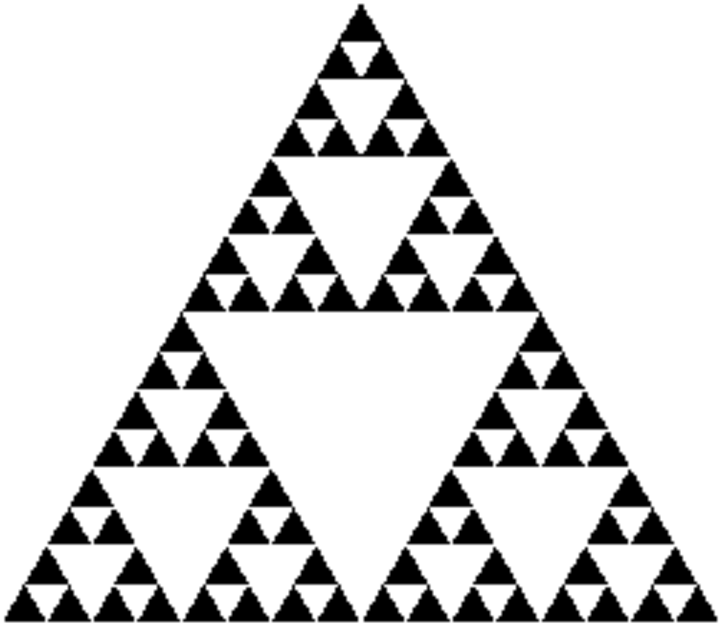
- Connect bisectors of sides and remove central triangle



- Repeat

Example

- Five subdivisions



The gasket as a fractal

- Consider the filled area (black) and the perimeter (the length of all the lines around the filled triangles)
- As we continue subdividing
  - the area goes to zero
  - but the perimeter goes to infinity
- This is not an ordinary geometric object
  - It is neither two- nor three-dimensional
- It is a *fractal* (fractional dimension) object

Gasket Program

```
#include <GL/glut.h>

/* initial triangle */

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

int n; /* number of recursive steps */
```

Draw one triangle

```
void triangle( GLfloat *a, GLfloat *b,
 GLfloat *c)

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

Triangle Subdivision

```
void divide_triangle(GLfloat *a, GLfloat *b, GLfloat *c,
 int m)
{
/* triangle subdivision using vertex numbers */
    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));
 /* draw triangle at end of recursion */
}
```

display and init Functions

```
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);
}
```

main Function

```
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();
}
```

Efficiency Note

By having the **glBegin** and **glEnd** in the display callback rather than in the function **triangle** and using **GL_TRIANGLES** rather than **GL_POLYGON** in **glBegin**, we call **glBegin** and **glEnd** only once for the entire gasket rather than once for each triangle

Moving to 3D

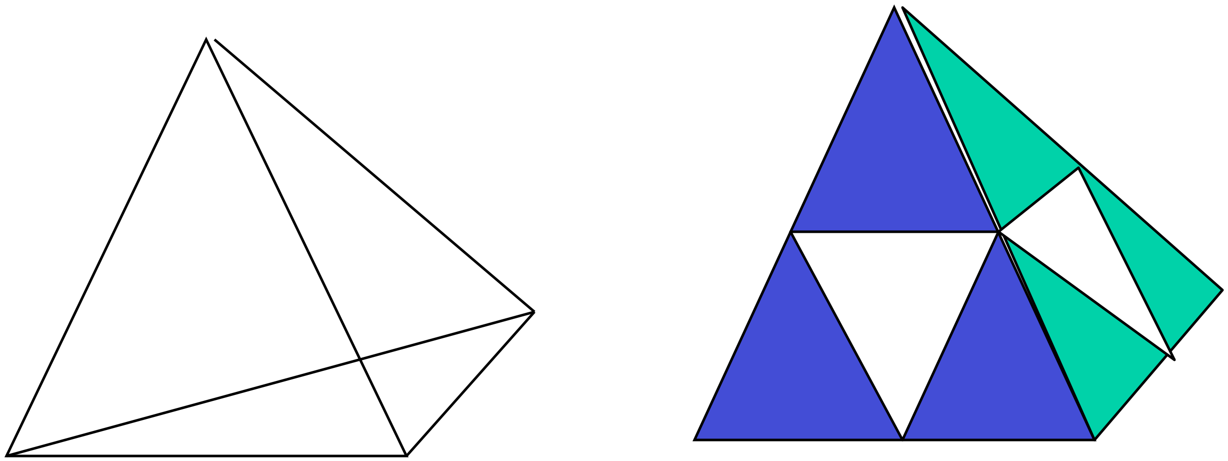
- We can easily make the program three-dimensional by using

```
GLfloat v[3][3]
glVertex3f
glOrtho
```

- But that would not be very interesting
- Instead, we can start with a tetrahedron

3D Gasket

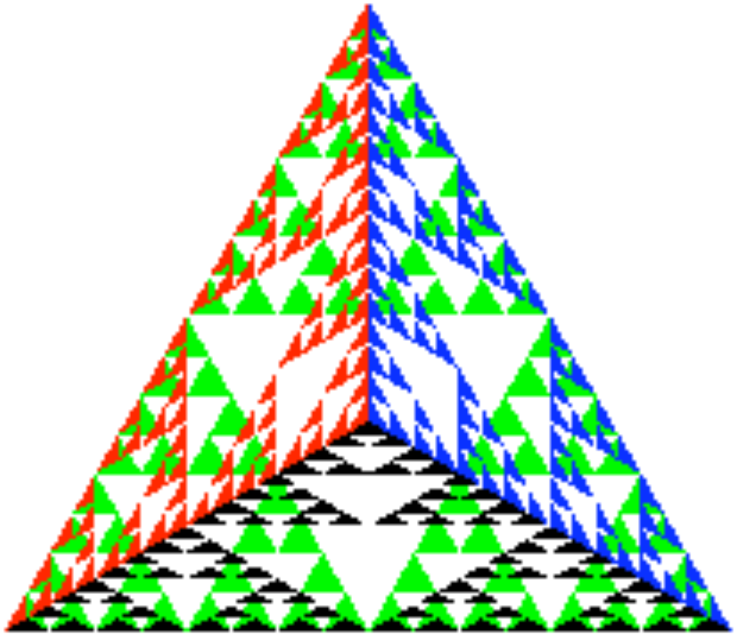
- We can subdivide each of the four faces



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

Example

after 5 iterations



triangle code

```
void triangle( GLfloat *a, GLfloat *b,
 GLfloat *c)
{
     glVertex3fv(a);
     glVertex3fv(b);
     glVertex3fv(c);
}
```

subdivision code

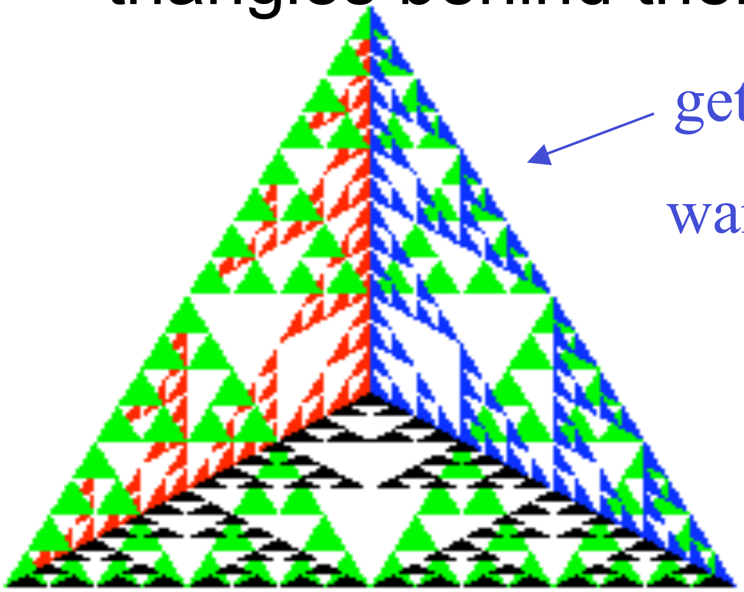
```
void divide_triangle(GLfloat *a, GLfloat *b,
 GLfloat *c, int m)
{
    GLfloat v1[3], v2[3], v3[3];
    int j;
    if(m>0)
    {
        for(j=0; j<3; j++) v1[j]=(a[j]+b[j])/2;
        for(j=0; j<3; j++) v2[j]=(a[j]+c[j])/2;
        for(j=0; j<3; j++) v3[j]=(b[j]+c[j])/2;
        divide_triangle(a, v1, v2, m-1);
        divide_triangle(c, v2, v3, m-1);
        divide_triangle(b, v3, v1, m-1);
    }
    else(triangle(a,b,c));
}
```

tetrahedron code

```
void tetrahedron( int m)
{
    glColor3f(1.0,0.0,0.0);
    divide_triangle(v[0], v[1], v[2], m);
    glColor3f(0.0,1.0,0.0);
    divide_triangle(v[3], v[2], v[1], m);
    glColor3f(0.0,0.0,1.0);
    divide_triangle(v[0], v[3], v[1], m);
    glColor3f(0.0,0.0,0.0);
    divide_triangle(v[0], v[2], v[3], m);
}
```

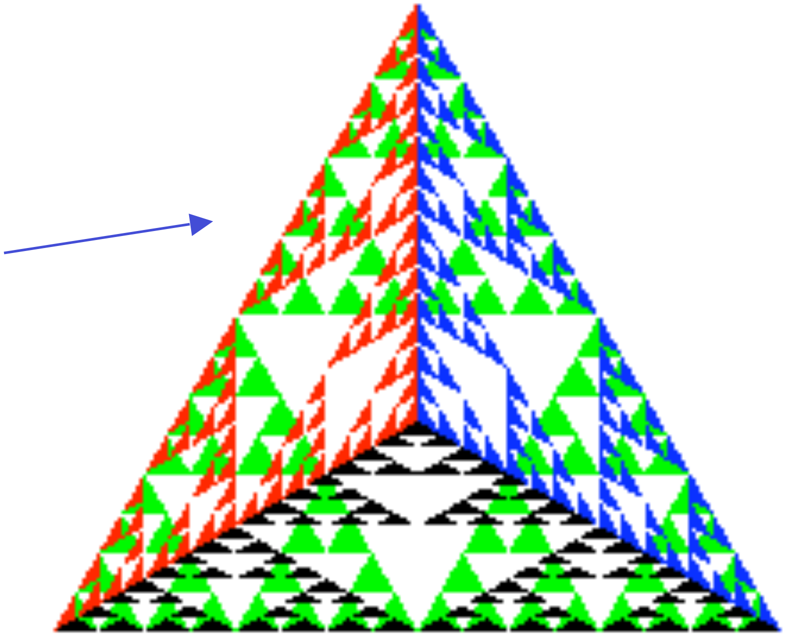
Almost Correct

- 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



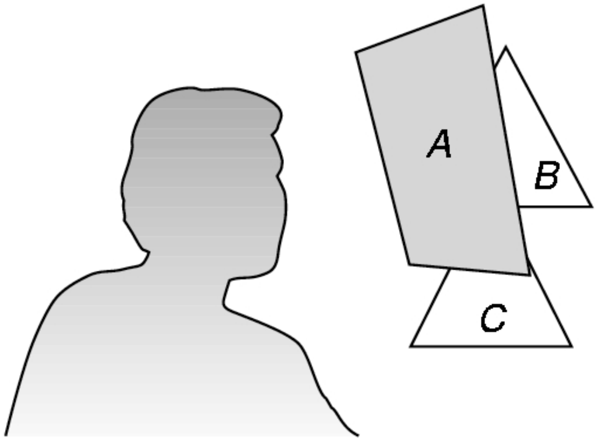
get this

want this



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



Using the z-buffer algorithm

- 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.c`**
    -
      ```
      glutInitDisplayMode
       (GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH)
      ```
  - Enabled in **`init.c`**
    -
      ```
      glEnable(GL_DEPTH_TEST)
      ```
  - Cleared in the display callback
    -
      ```
      glClear(GL_COLOR_BUFFER_BIT |
        GL_DEPTH_BUFFER_BIT)
      ```

Surface vs Volume Subdvision

- In our example, we divided the surface of each face
- We could also divide the volume using the same midpoints
- The midpoints define four smaller tetrahedrons, one for each vertex
- Keeping only these tetrahedrons removes a *volume* in the middle
- See text for code

Volume Subdivision