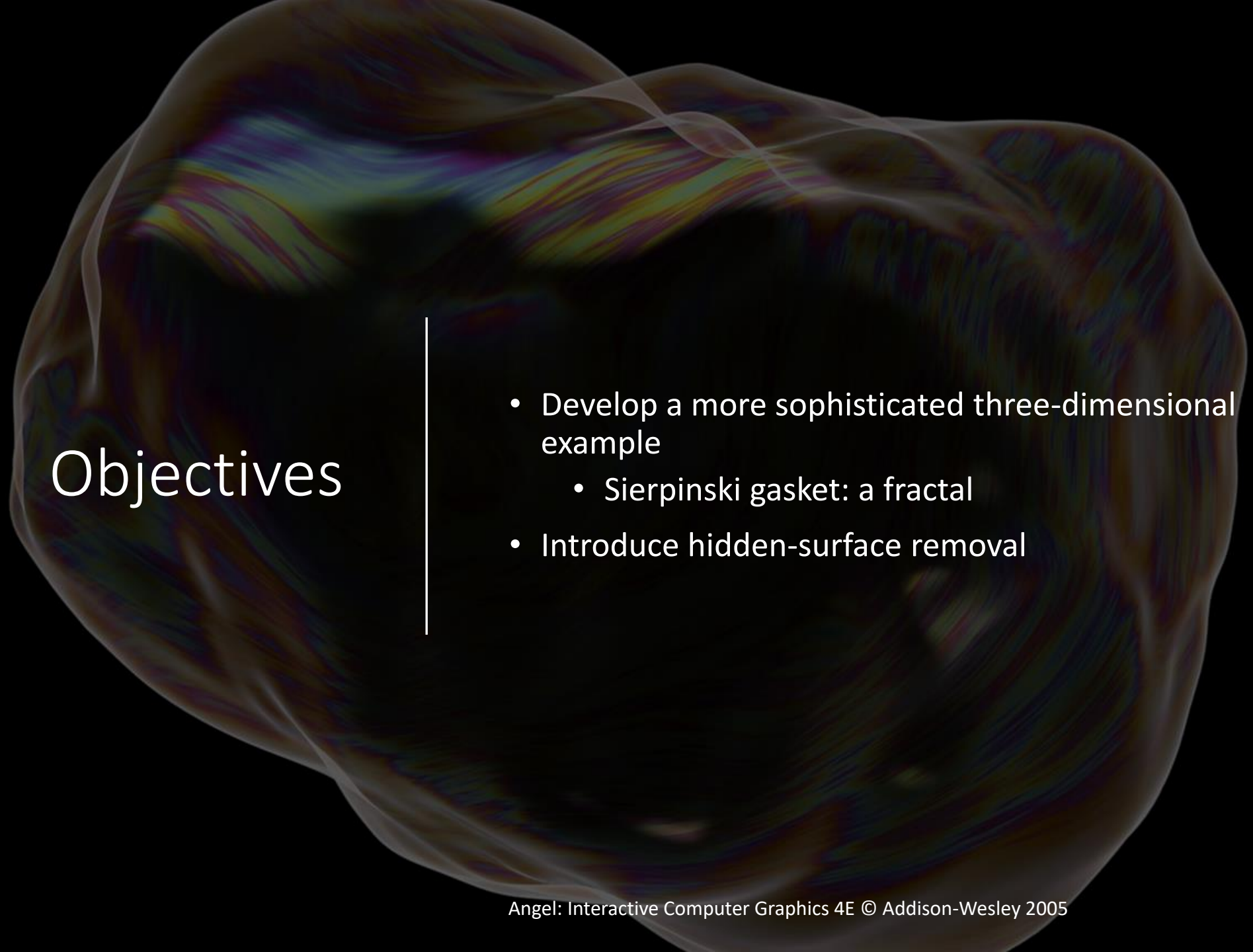
Programming with OpenGL Part 3b: Three Dimensions

Guided by Hamzah Asyrani Sulaiman



Objectives

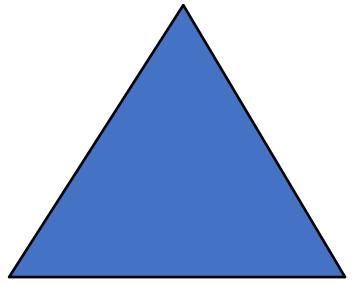
- Develop a more sophisticated three-dimensional example
  - Sierpinski gasket: a fractal
- Introduce hidden-surface removal

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

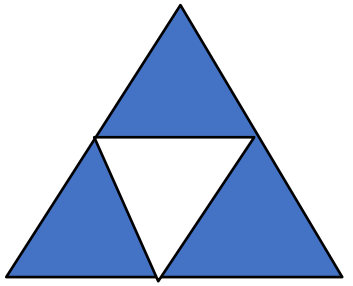
Sierpinski Gasket (2D)

Start with a triangle



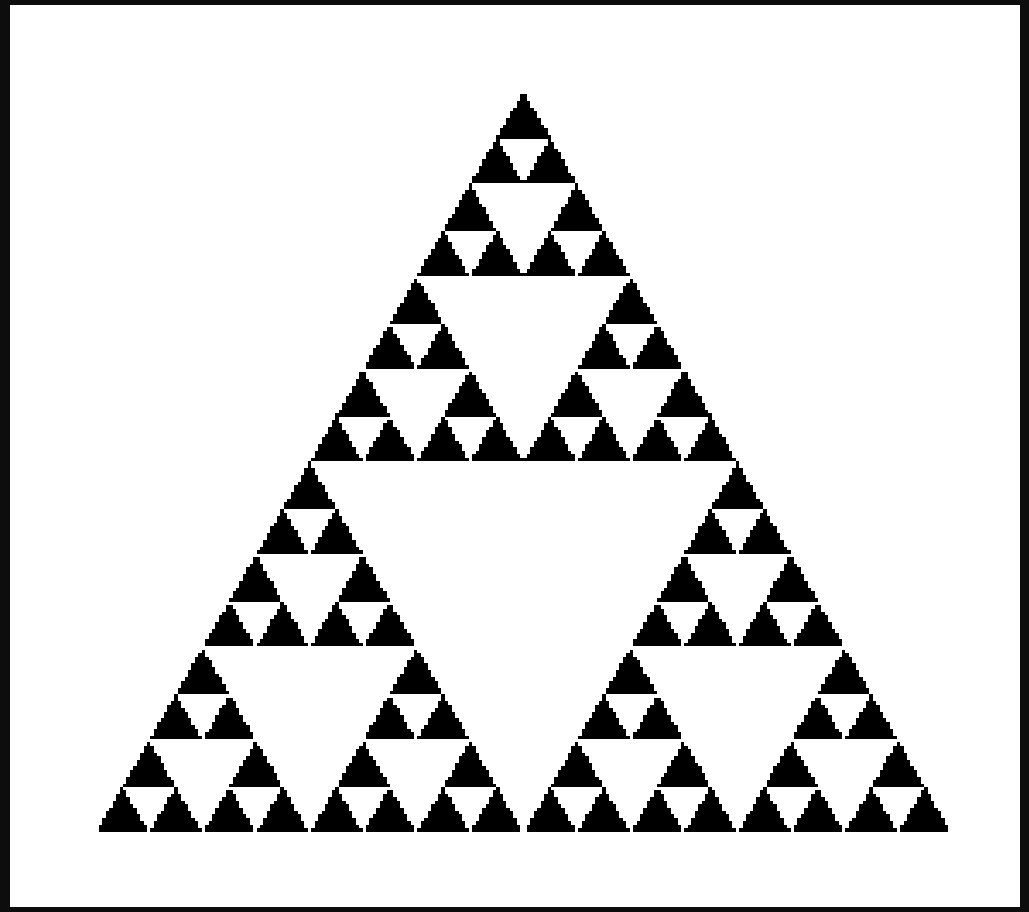
Connect bisectors of sides and remove central triangle

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

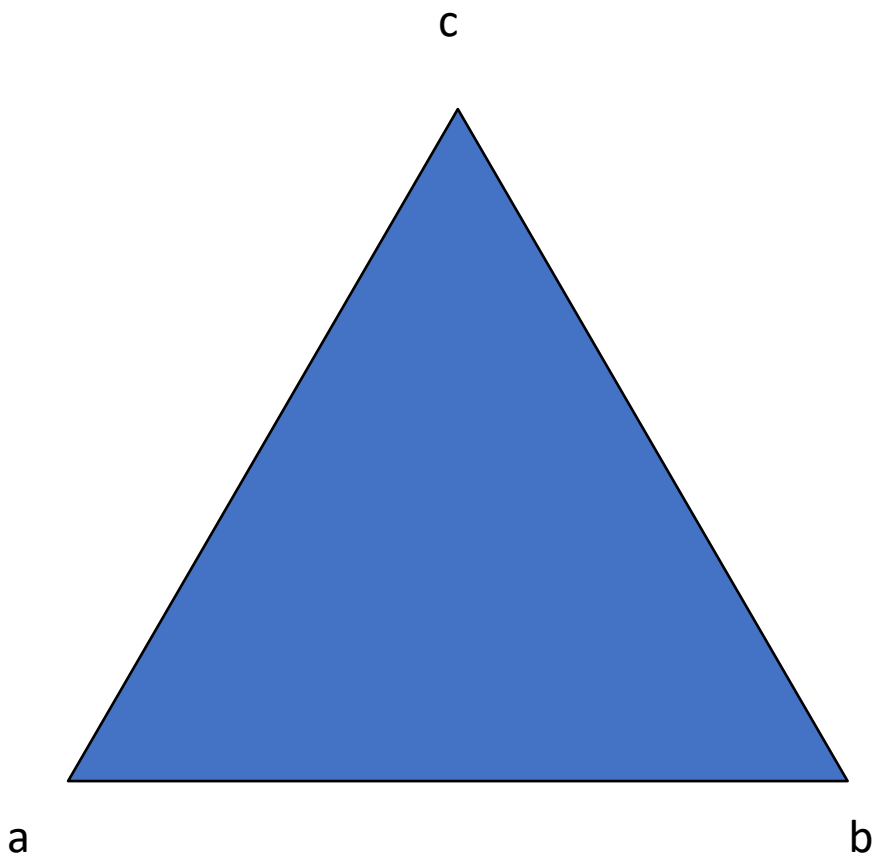
```
#include <stdlib.h>
#include <GL/glut.h>

/* initial triangle */

GLfloat v[3][2] = { {-1.0, -0.58},
   {1.0, -0.58}, {0.0, 1.15} };
GLfloat colors[4][3] = { { 1.0, 0.0, 0.0
}, { 0.0, 1.0, 0.0 },
{ 0.0, 0.0, 1.0 }, { 0.0, 0.0, 0.0 } };

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

Draw one triangle

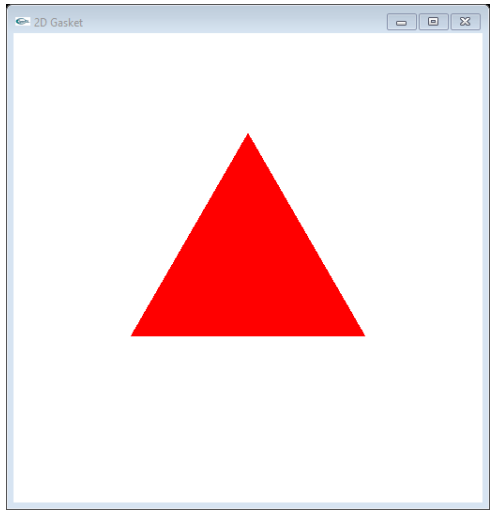
```
//a pointer to an array with three
elements.

    void triangle(GLfloat* va, GLfloat* vb,
    GLfloat* vc)
    {
        glColor3fv(colors[0]);
        glVertex2fv(va);
        glColor3fv(colors[1]);
        glVertex2fv(vb);
        glColor3fv(colors[3]);
        glVertex2fv(vc);
    }
```

glVertex — specify a vertex

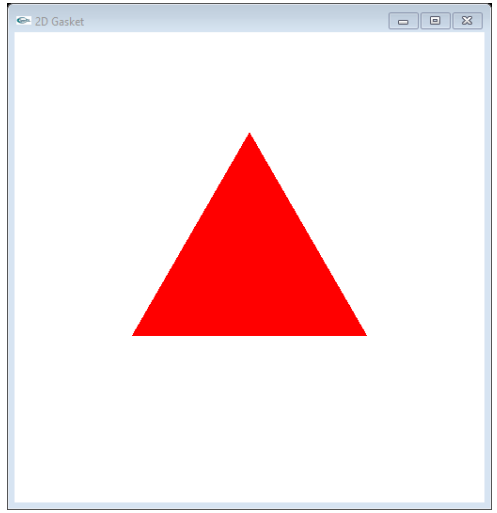
https://www.khronos.org/registry/OpenGL-Refpages/gl2.1/xhtml/glVertex.xml

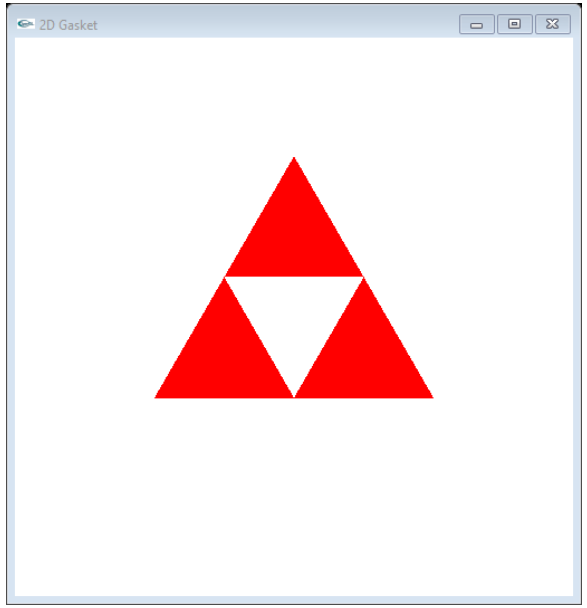

c
a
b

```
triangle(a, b, c)); // draw tetrahedron
```



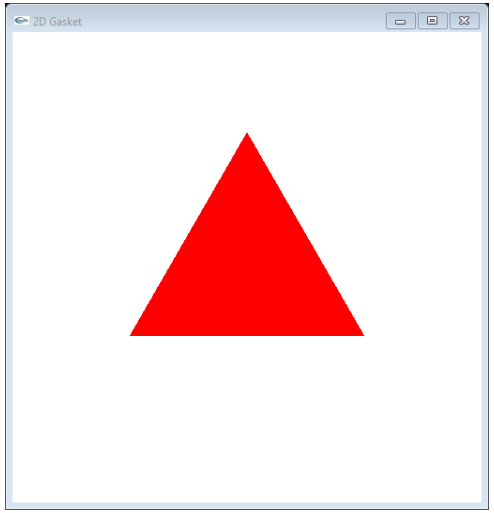
So how to turn from this

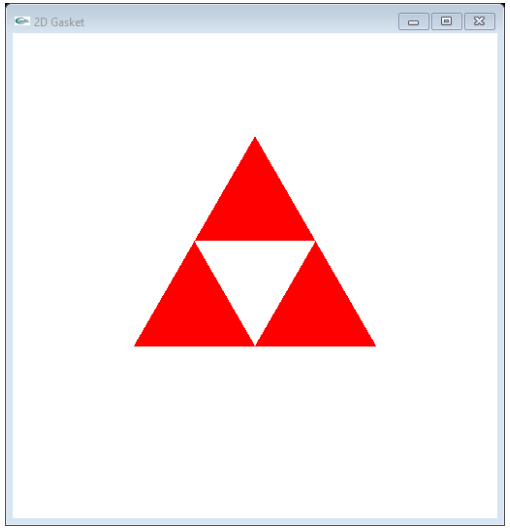




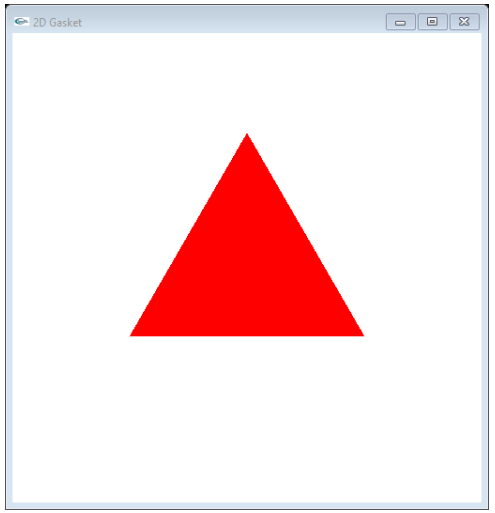
Into this?

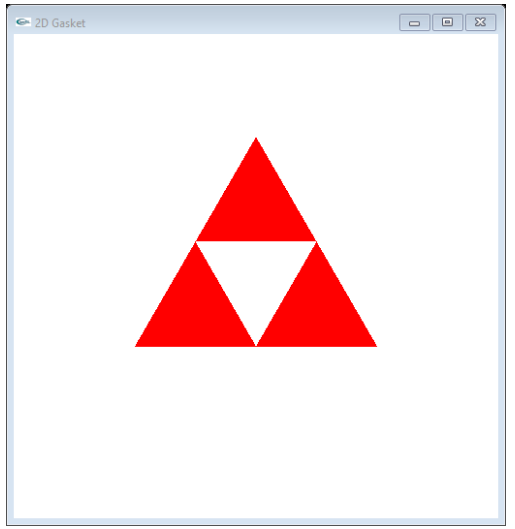
Get it?



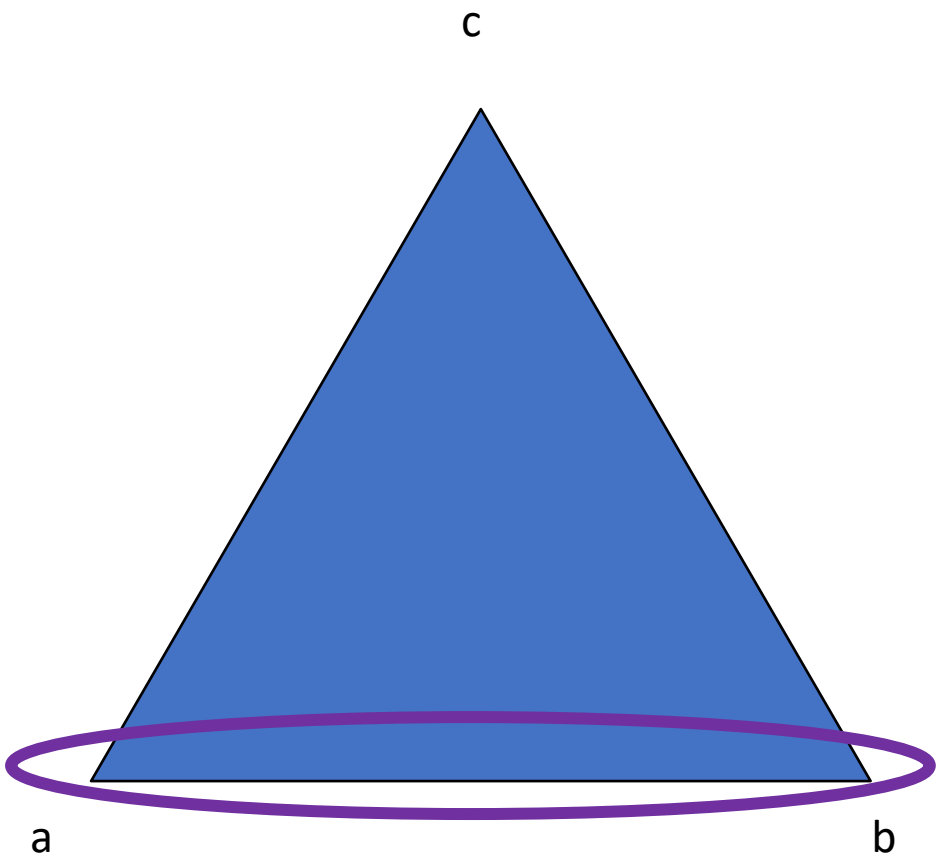


Hint – get the middle point first





FIRST FOR LOOP



How to code?

```
Divide between A and B
```

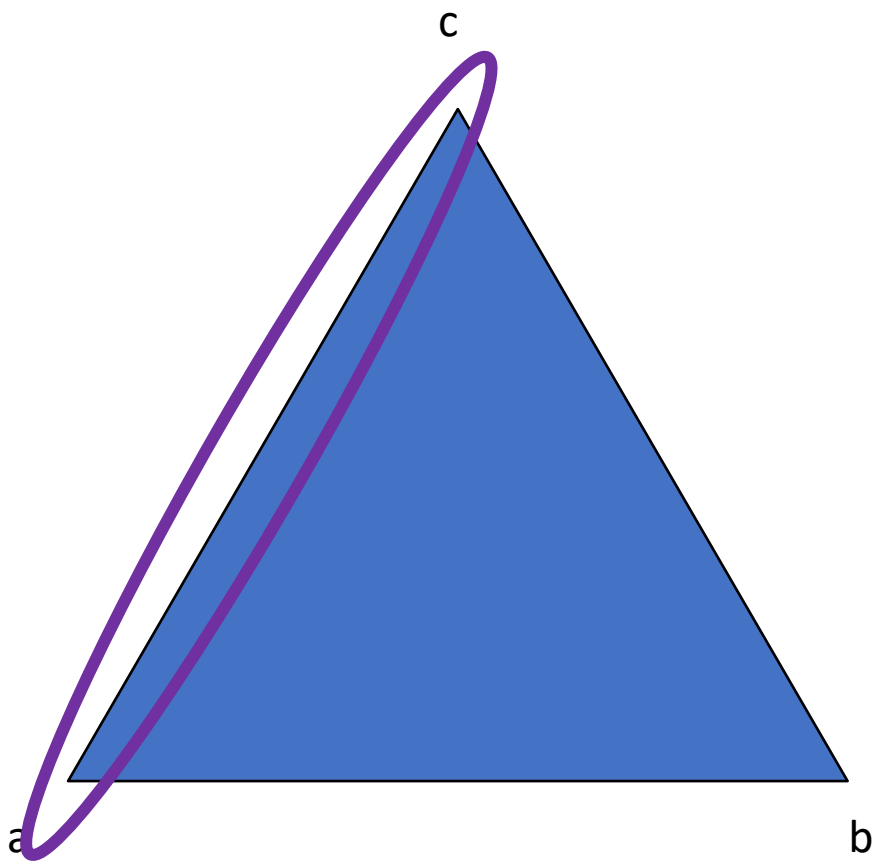
Get x and y

```
mid1 = (a_x + b_x) / 2;
mid2 = (a_y + b_y) / 2;
```

But not so cool enough!!

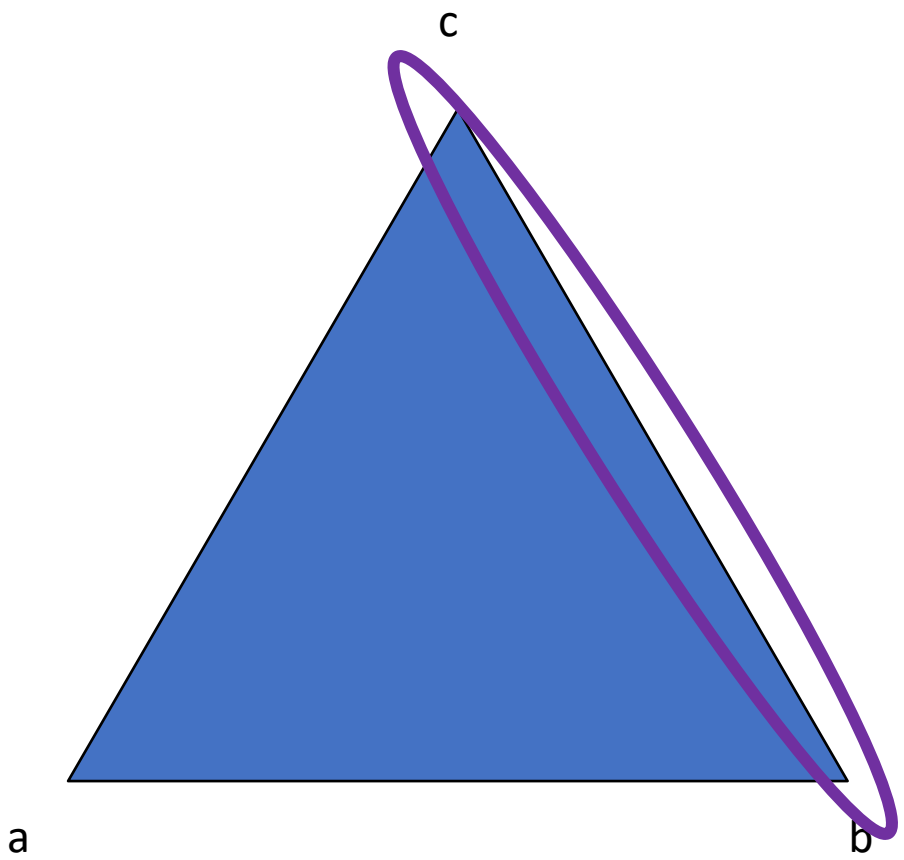
```
for (j = 0; j<2; j++)
{
 mid[0][j] = (a[j] + b[j]) / 2;
}
```

SECOND FOR LOOP



```
for (j = 0; j<2; j++)
{
mid[1][j] = (a[j] + c[j]) / 2;
}
```

THIRD FOR LOOP

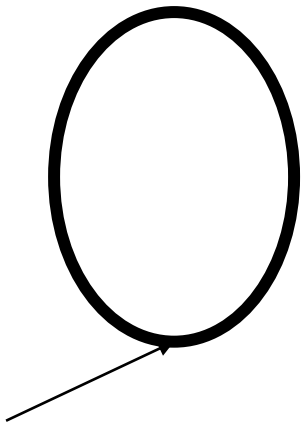


```
for (j = 0; j<2; j++)
{
mid[2][j] = (b[j] + c[j]) / 2;
}
```

Now divide it

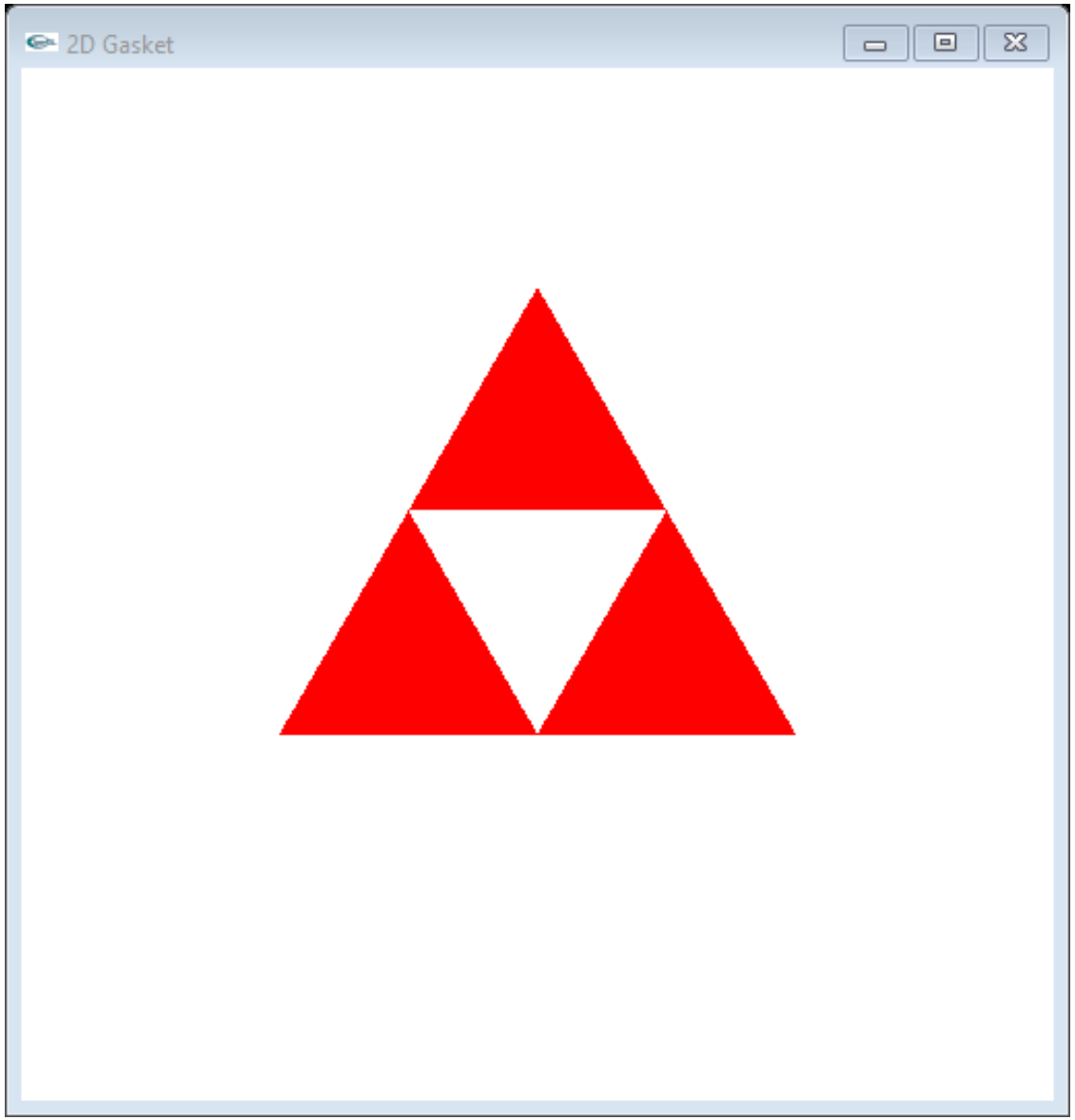
```
/* create tetrahedrons by subdivision */
divide_tetra(a, mid[0], mid[1], m - 1);
divide_tetra(c, mid[1], mid[2], m - 1);
divide_tetra(b, mid[2], mid[0], m - 1);
```

```
/* create tetrahedrons by subdivision */
divide_tetra(a, mid[0], mid[1], m - 1);
divide_tetra(c, mid[1], mid[2], m - 1);
divide_tetra(b, mid[2], mid[0], m - 1);
```



Deduct by -1 to enter the loop again

```
void divide_tetra(GLfloat *a, GLfloat *b, GLfloat *c, int m)
{
GLfloat mid[3][3];
int j;

if (m>0)
{
    /* compute six midpoints */
    for (j = 0; j<2; j++) mid[0][j] = (a[j] + b[j]) / 2;
    for (j = 0; j<2; j++) mid[1][j] = (a[j] + c[j]) / 2;
    for (j = 0; j<2; j++) mid[2][j] = (b[j] + c[j]) / 2;
    /* create tetrahedrons by subdivision */
    divide_tetra(a, mid[0], mid[1], m - 1);
    divide_tetra(c, mid[1], mid[2], m - 1);
    divide_tetra(b, mid[2], mid[0], m - 1);
}
else
    (tetra(a, b, c)); /* draw tetrahedron at end of recursion */
}
```


2D Gasket

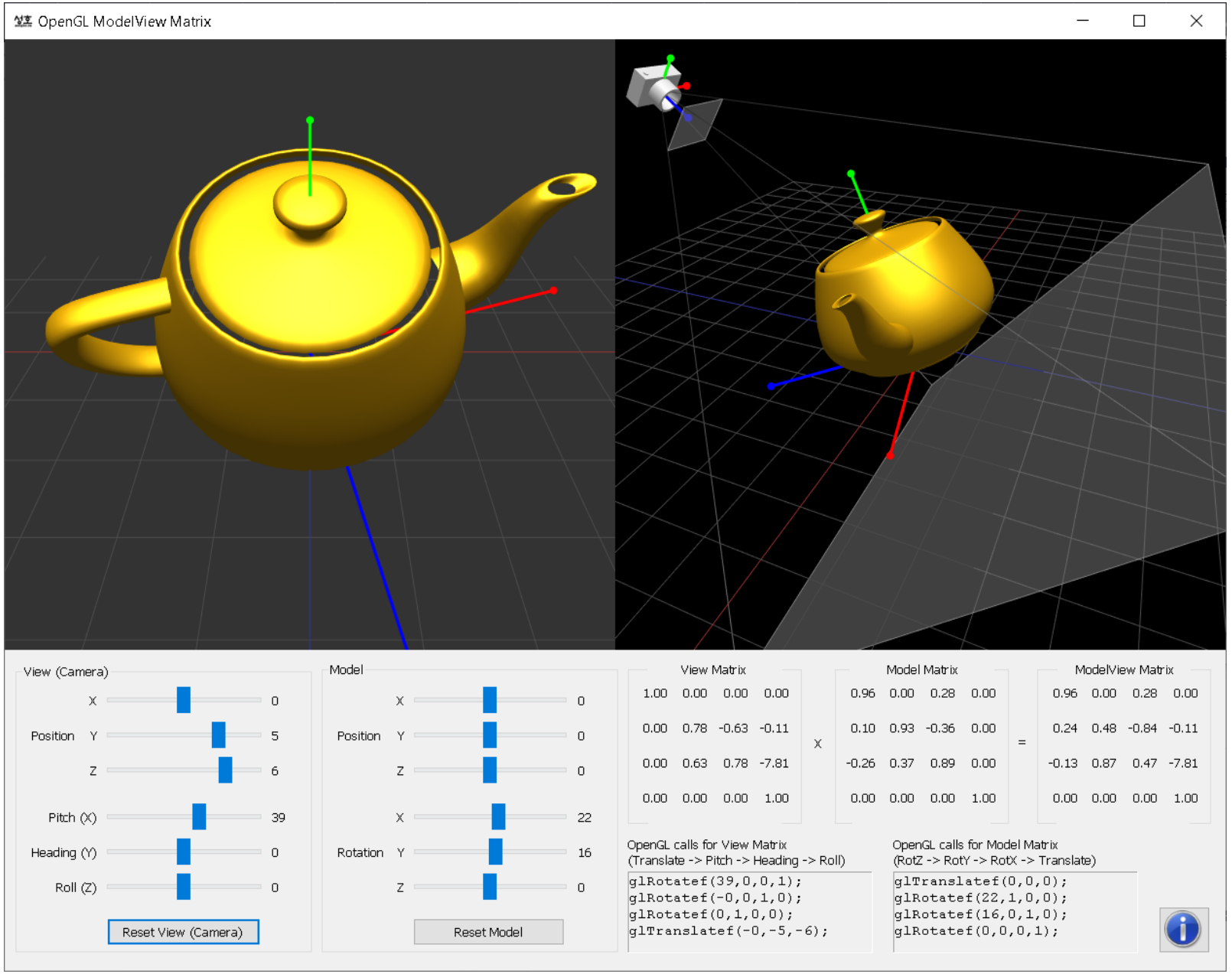
display and init Functions

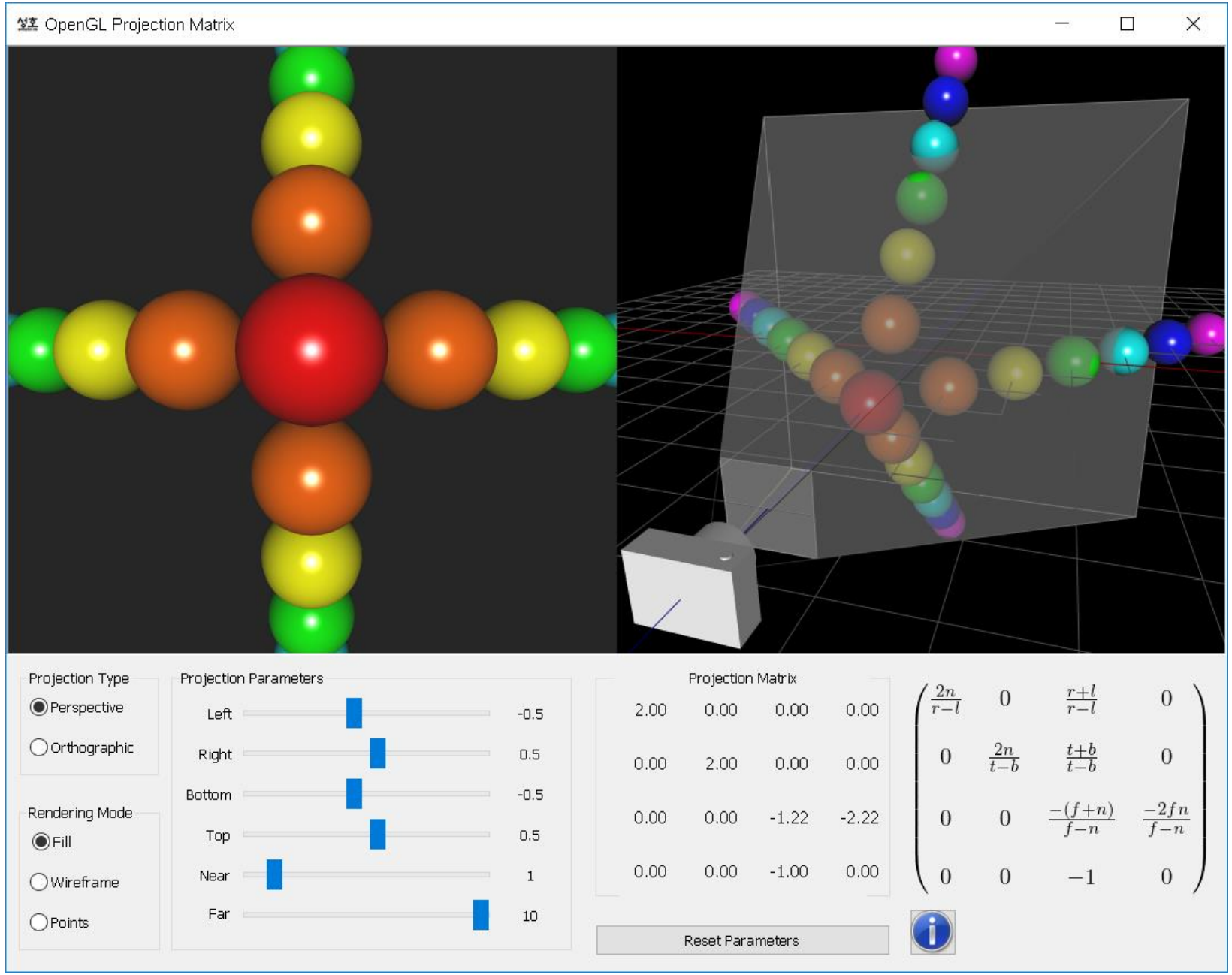
```
void display()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glBegin(GL_TRIANGLES);
    divide_triangle(v[0], v[1], v[2], n);
    glEnd();
    glFlush();
}
```

display and init Functions

```
void myReshape(int w, int h)
{
   glViewport(0, 0, w, h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   if (w <= h)
   glOrtho(-2.0, 2.0, -2.0 * (GLfloat)h / (GLfloat)w,
   2.0 * (GLfloat)h / (GLfloat)w, -10.0, 10.0);
   else
   glOrtho(-2.0 * (GLfloat)w / (GLfloat)h,
   2.0 * (GLfloat)w / (GLfloat)h, -2.0, 2.0, -10.0, 10.0);
   glMatrixMode(GL_MODELVIEW);
   glutPostRedisplay();
}
```

http://www.songho.ca/opengl/gl_transform.html


OpenGL ModelView Matrix
View (Camera)
X 0
Position Y 5
Z 6
Pitch (X) 39
Heading (Y) 0
Roll (Z) 0
Reset View (Camera)
Model
X 0
Position Y 0
Z 0
X 22
Rotation Y 16
Z 0
Reset Model
View Matrix
1.00 0.00 0.00 0.00
0.00 0.78 -0.63 -0.11
0.00 0.63 0.78 -7.81
0.00 0.00 0.00 1.00
X
Model Matrix
0.96 0.00 0.28 0.00
0.10 0.93 -0.36 0.00
-0.26 0.37 0.89 0.00
0.00 0.00 0.00 1.00
=
ModelView Matrix
0.96 0.00 0.28 0.00
0.24 0.48 -0.84 -0.11
-0.13 0.87 0.47 -7.81
0.00 0.00 0.00 1.00
OpenGL calls for View Matrix
(Translate -> Pitch -> Heading -> Roll)
glRotatef(39,0,0,1);
glRotatef(-0,0,1,0);
glRotatef(0,1,0,0);
glTranslatef(-0,-5,-6);
OpenGL calls for Model Matrix
(RotZ -> RotY -> RotX -> Translate)
glTranslatef(0,0,0);
glRotatef(22,1,0,0);
glRotatef(16,0,1,0);
glRotatef(0,0,0,1);


OpenGL Projection Matrix
Projection Type
Perspective
Orthographic
Rendering Mode
Fill
Wireframe
Points
Projection Parameters
Left
-0.5
Right
0.5
Bottom
-0.5
Top
0.5
Near
1
Far
10
Projection Matrix
2.00 0.00 0.00 0.00
0.00 2.00 0.00 0.00
0.00 0.00 -1.22 -2.22
0.00 0.00 -1.00 0.00
$\begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$
Reset Parameters

main Function

```
int main(int argc, char** argv)
{
   n = 1; /* or enter number of subdivision steps here */
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
   glutInitWindowSize(500, 500);
   glutCreateWindow("3D Gasket");
   glutReshapeFunc(myReshape);
   glutDisplayFunc(display);
   glEnable(GL_DEPTH_TEST);
   glClearColor(1.0, 1.0, 1.0, 1.0);
   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[4][3]
```

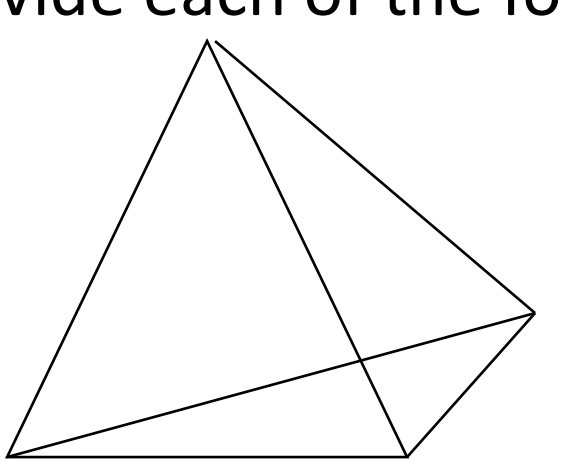
```
glVertex3f
```

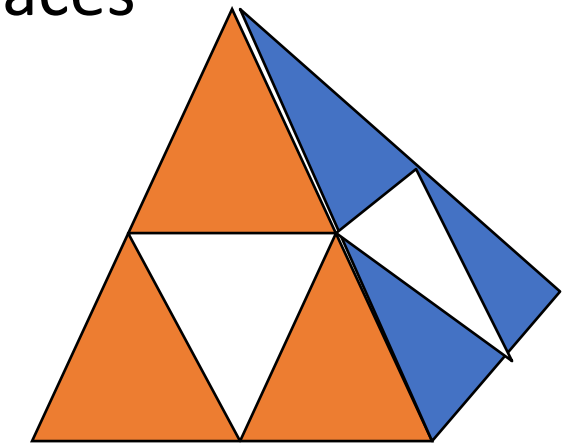
```
glOrtho
```

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

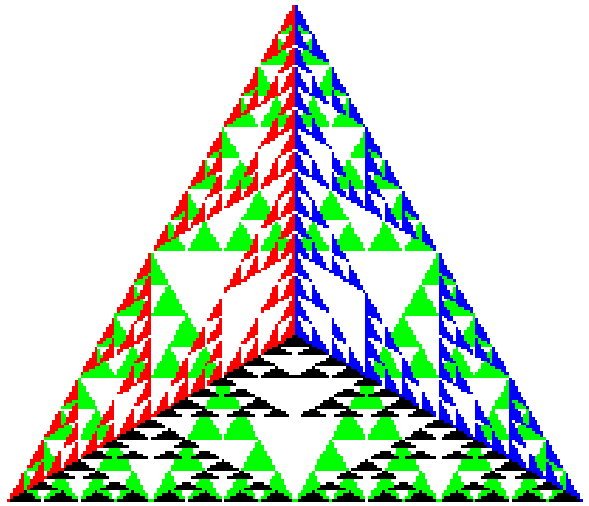
3D Gasket

- 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

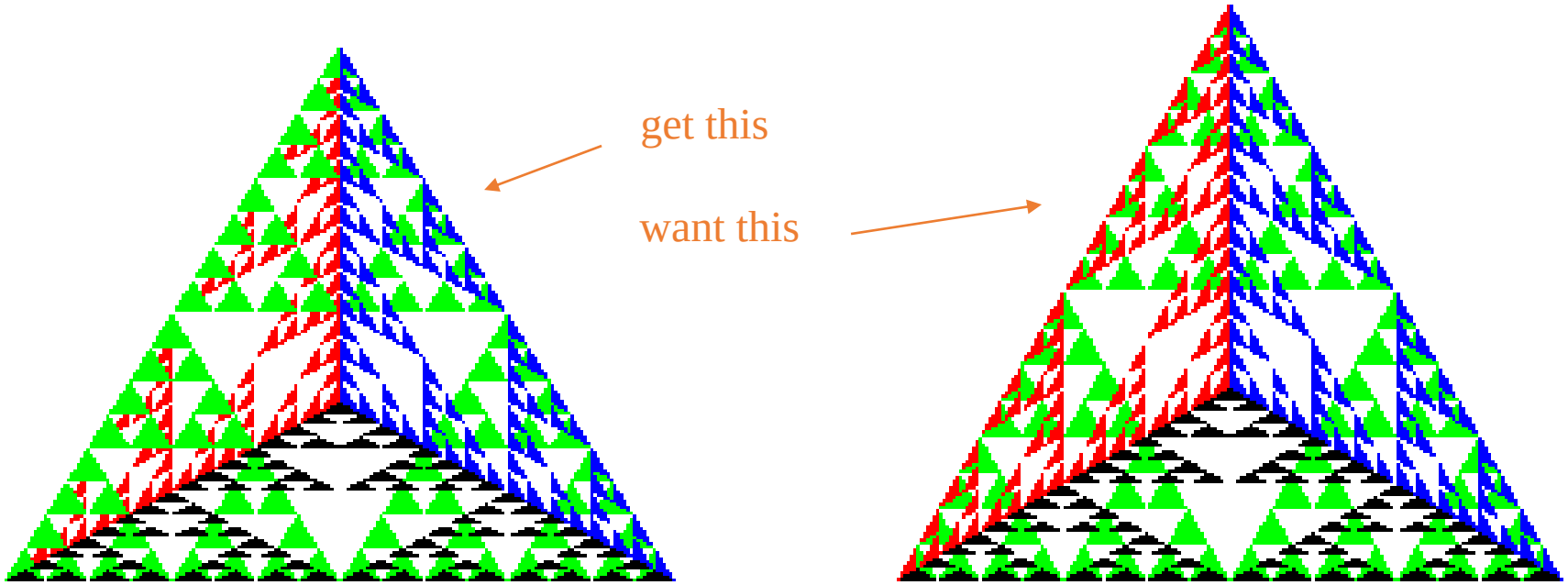
```
void divide_tetra(GLfloat *a, GLfloat *b, GLfloat
*c, GLfloat *d, int m)
{

GLfloat mid[6][3];
int j;
if (m>0)
{
/* compute six midpoints */

for (j = 0; j<3; j++) mid[0][j] = (a[j] + b[j]) / 2;
for (j = 0; j<3; j++) mid[1][j] = (a[j] + c[j]) / 2;
for (j = 0; j<3; j++) mid[2][j] = (a[j] + d[j]) / 2;
for (j = 0; j<3; j++) mid[3][j] = (b[j] + c[j]) / 2;
for (j = 0; j<3; j++) mid[4][j] = (c[j] + d[j]) / 2;
for (j = 0; j<3; j++) mid[5][j] = (b[j] + d[j]) / 2;

/* create 4 tetrahedrons by subdivision */

divide_tetra(a, mid[0], mid[1], mid[2], m - 1);
divide_tetra(mid[0], b, mid[3], mid[5], m - 1);
divide_tetra(mid[1], mid[3], c, mid[4], m - 1);
divide_tetra(mid[2], mid[4], d, mid[5], m - 1);

}
else(tetra(a, b, c, d)); /* draw tetrahedron at end
of recursion */
}
```

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);
}
```

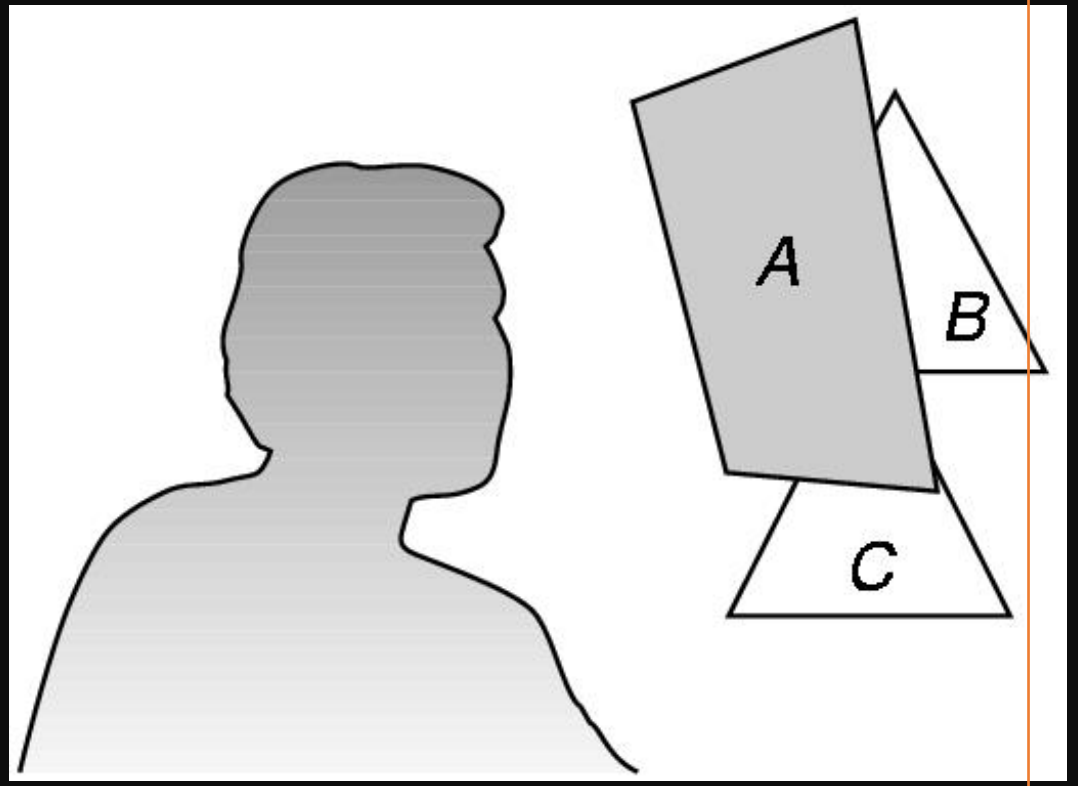
Almost Correct

- 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



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