Quads, Quad Strips & Polygons

OpenGL

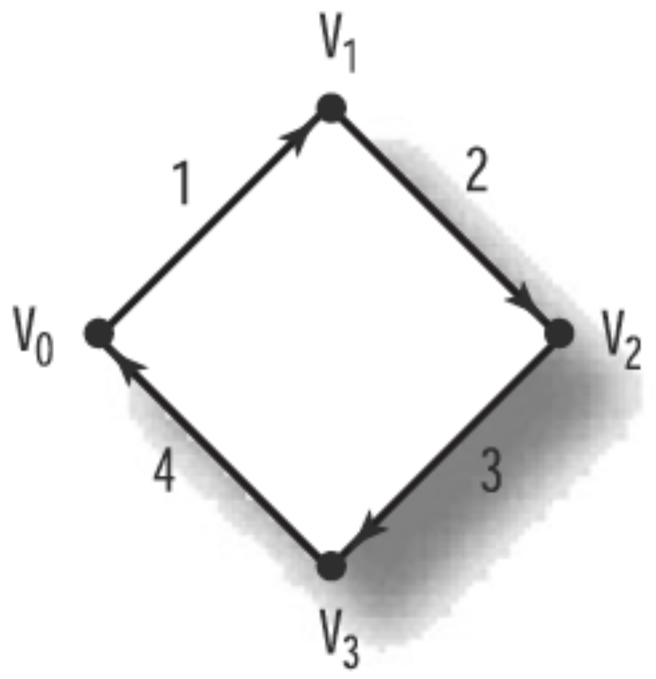
Learning Outcomes

- Have used Quadrilaterals, Quadrilateral strips and general purpose polygons.
- Understand the planar and convex rules for polygon construction
- Know how to turn off edges when rendering apparently convex polygons.

Other Primitives

- Triangles are the preferred primitive for object composition because most OpenGL hardware specifically accelerates triangles, but they are not the only primitives available.
- Some hardware provides for acceleration of other shapes as well, and programmatically, using a general-purpose graphics primitive might be simpler.
- These OpenGL primitives provide for rapid specification of:
    - quadrilaterals
    - quadrilateral strips
    - general-purpose polygons

Quads

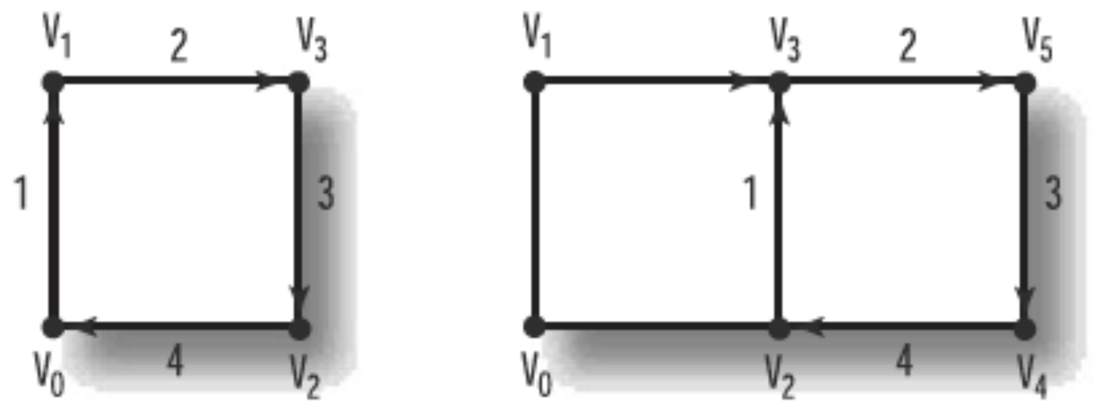


- If you add one more side to a triangle, you get a quadrilateral, or a four-sided figure.
- OpenGL's GL_QUADS primitive draws a four-sided polygon.
- This quad has a clockwise winding.
- All four corners of the quadrilateral must lie in a plane (no bent quads).

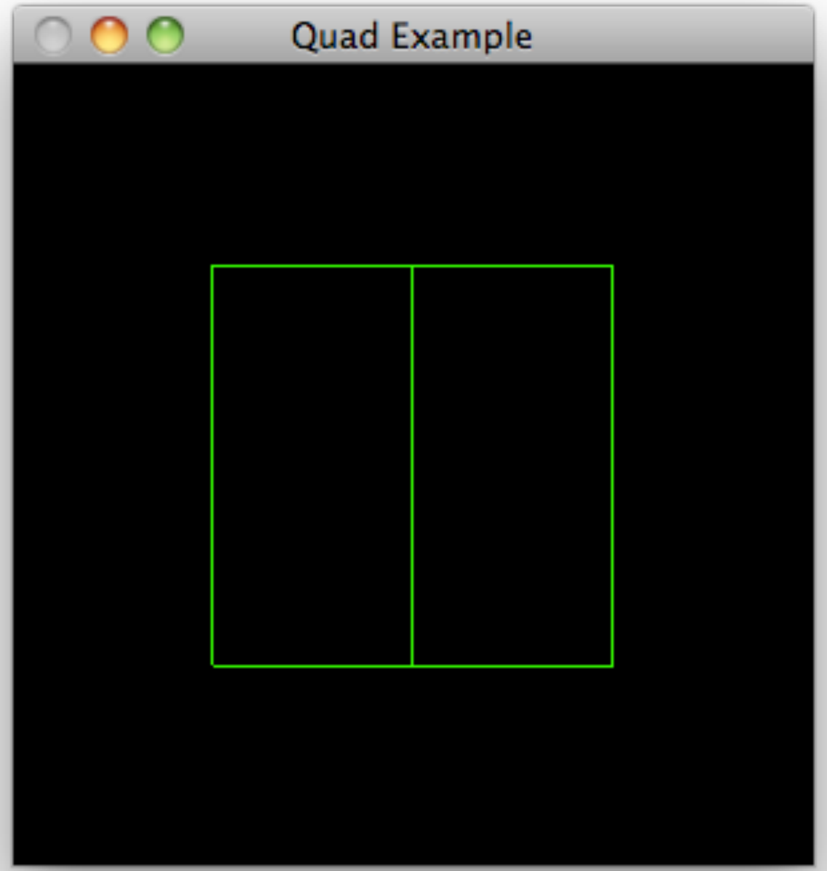
```
void renderScene(void)
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  glBegin(GL_QUADS);
    glVertex3f(-50.0f,  0.0f, 0.0f);
    glVertex3f( 0.0f,  50.0f, 0.0f);
    glVertex3f( 50.0f,  0.0f, 0.0f);
    glVertex3f( 0.0f, -50.0f, 0.0f);
 glEnd();

 glutSwapBuffers();
}
```

Quad Strip

- As you can for triangle strips, you can specify a strip of connected quadrilaterals with the GL_QUAD_STRIP primitive.
- Eg. a quad strip specified by six vertices.
- Note that these quad strips maintain a clockwise winding.

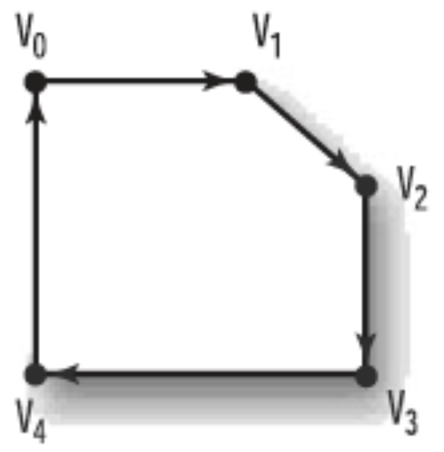


```
void renderScene(void)
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  glBegin(GL_QUAD_STRIP);
    glVertex3f(-50.0f,  -50.0f, 0.0f);
    glVertex3f( -50.0f,  50.0f, 0.0f);
    glVertex3f( 0.0f, -50.0f, 0.0f);
    glVertex3f( 0.0f,  50.0f, 0.0f);
    glVertex3f( 50.0f, -50.0f, 0.0f);
    glVertex3f( 50.0f, 50.0f, 0.0f);
 glEnd();

 glutSwapBuffers();
}
```

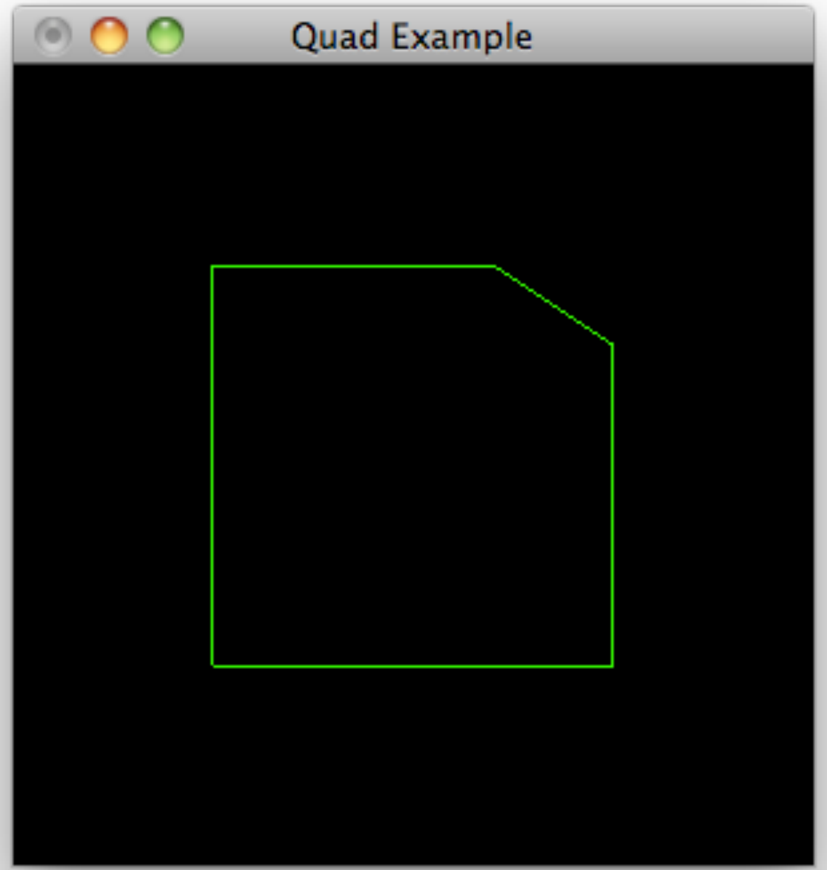


General Polygons

- The final OpenGL primitive is the GL_POLYGON, which you can use to draw a polygon having any number of sides.
- Eg a polygon consisting of five vertices.
- Polygons, like quads, must have all vertices on the same plane.



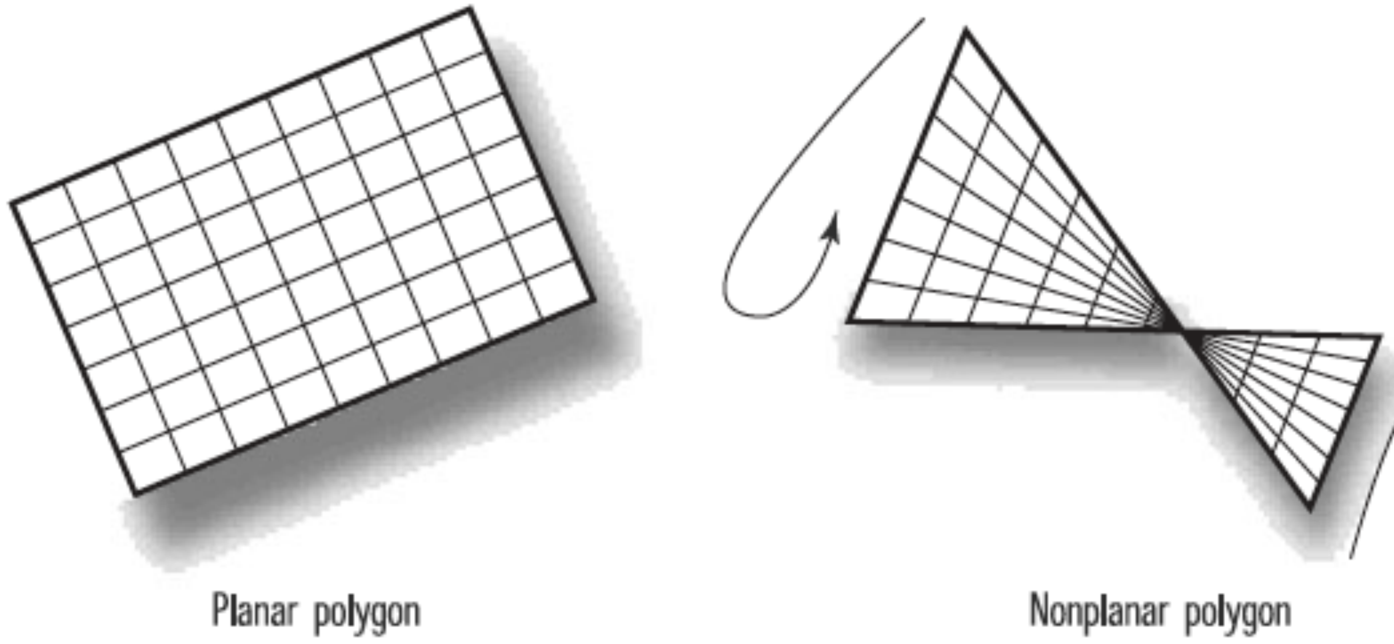
```
void renderScene(void)
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  glBegin(GL_POLYGON);
    glVertex3f(-50.0f,  50.0f, 0.0f);
    glVertex3f( 20.0f,  50.0f, 0.0f);
    glVertex3f( 50.0f,  30.0f, 0.0f);
    glVertex3f( 50.0f, -50.0f, 0.0f);
    glVertex3f(-50.0f, -50.0f, 0.0f);
 glEnd();

 glutSwapBuffers();
}
```



Polygon Construction Rules (1) Planar Rule

- Two important rules when using many polygons to construct a complex surface

  (1) All polygons must be planar. That is, all the vertices of the polygon must lie in a single plane. The polygon cannot twist or bend in space.



A good reason to use triangles. No triangle can ever be twisted so that all three points do not line up in a plane.

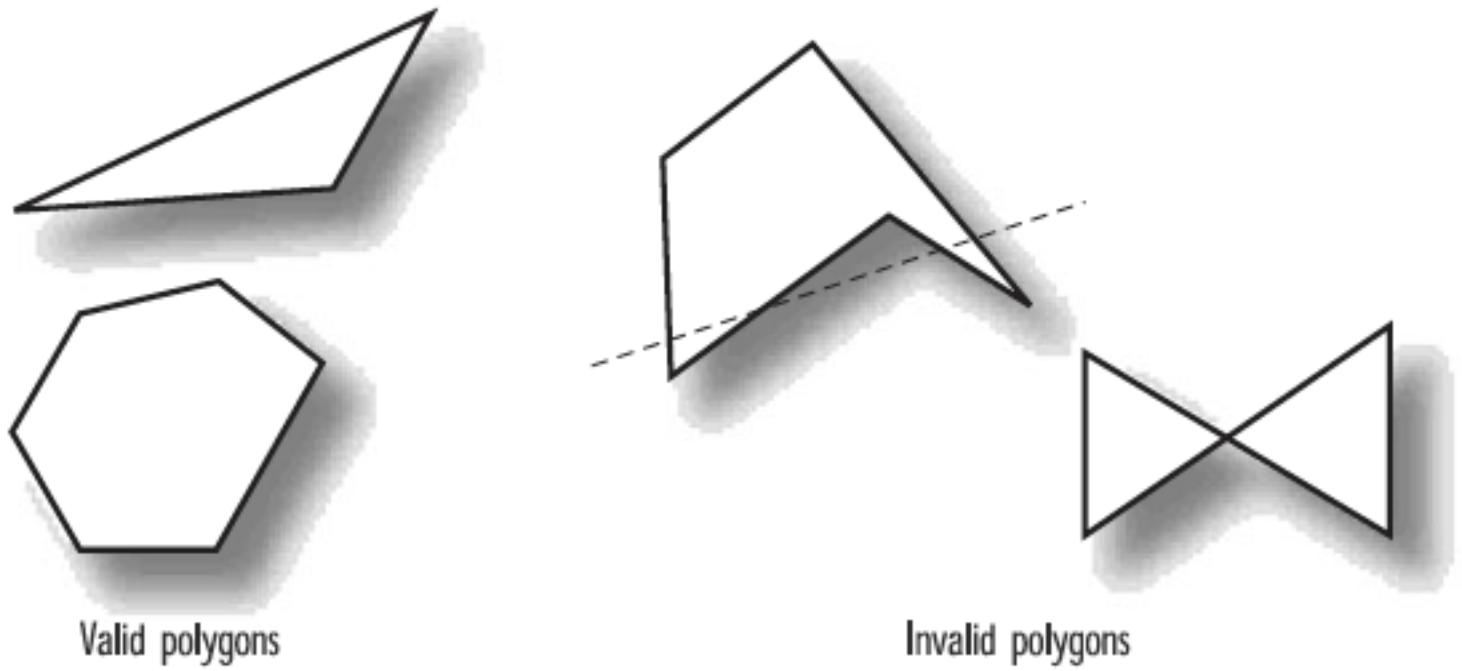
(2) Convex Rule

(2) A polygon's edges must not intersect, and the polygon must be convex.

A polygon intersects itself if any two of its lines cross.

Convex means that the polygon cannot have any indentions.

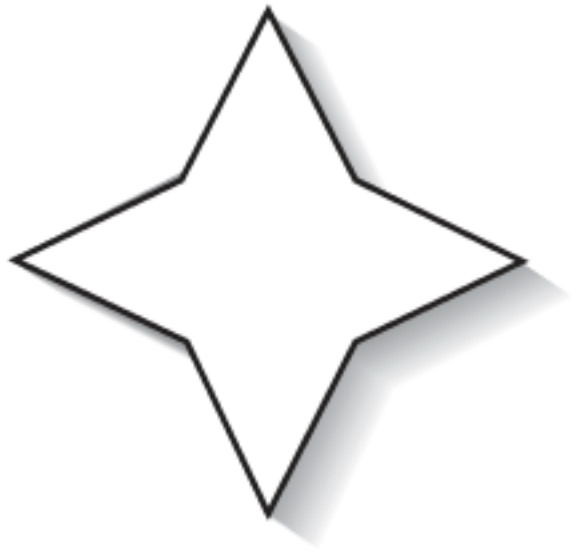
- These restrictions allow OpenGL to use very fast algorithms for rendering these polygons

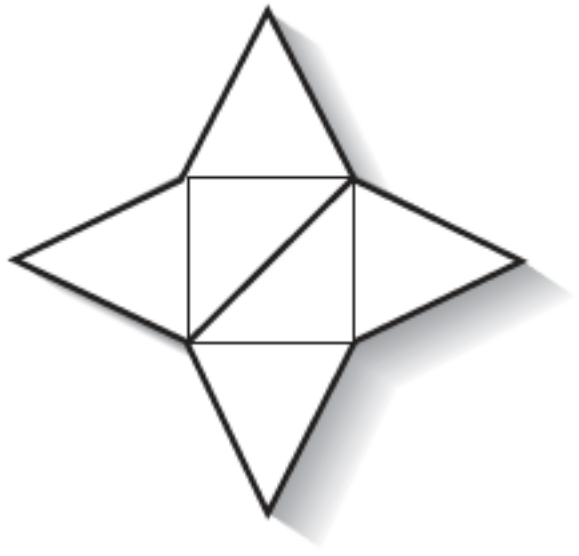


A useful test of a convex polygon is to draw some lines through it. If any given line enters and leaves the polygon more than once, the polygon is not convex.

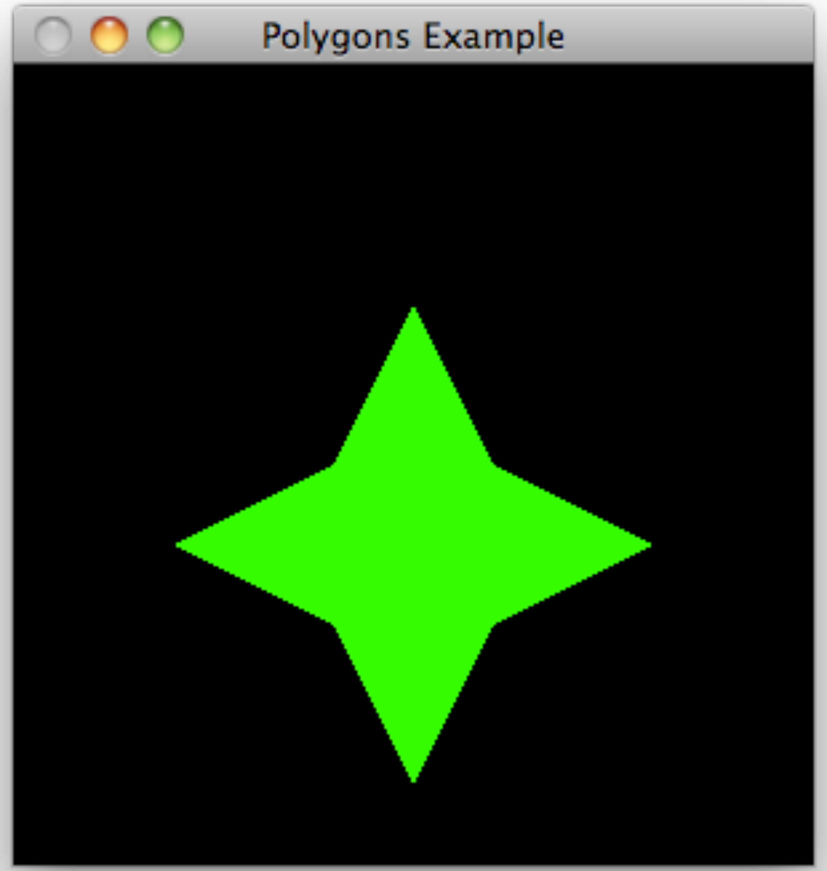
Subdivision and Edges

- Even though OpenGL can draw only convex polygons, there's still a way to create a non-convex polygon: by arranging two or more convex polygons together.
- E.G. a four-point star - obviously not convex and thus violates OpenGL's rules for simple polygon construction.
- However, one can be composed of six separate triangles, which are legal polygons.





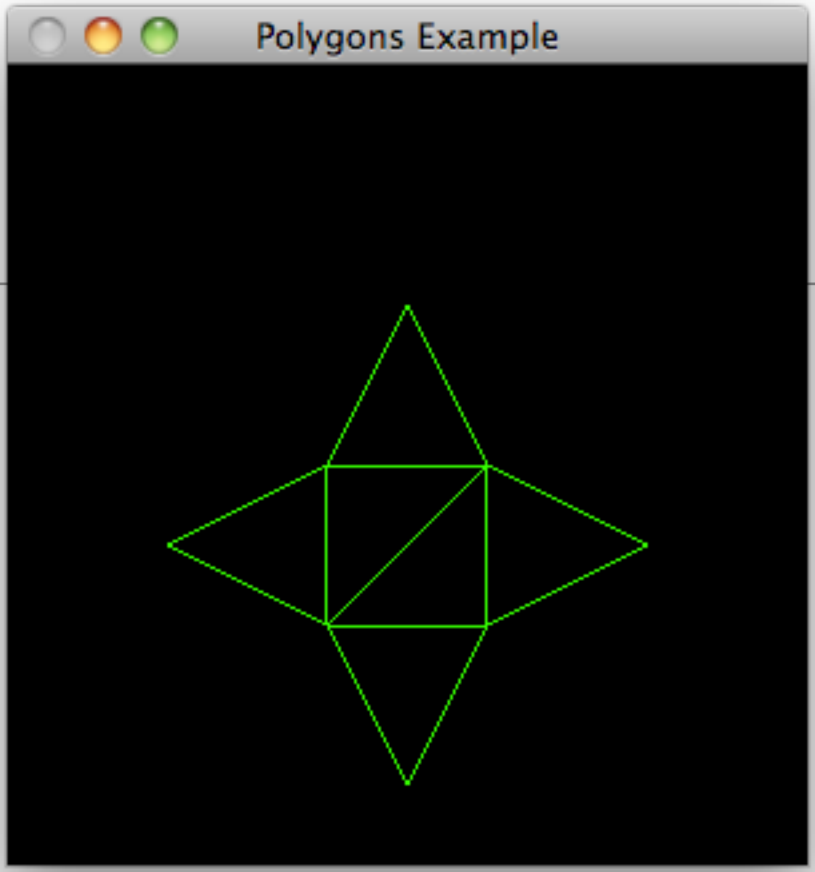
```
glBegin(GL_TRIANGLES);

   glVertex2f(-20.0f, 0.0f);
   glVertex2f(20.0f, 0.0f);
   glVertex2f(0.0f, 40.0f);

   glVertex2f(-20.0f,0.0f);
   glVertex2f(-60.0f,-20.0f);
   glVertex2f(-20.0f,-40.0f);

   glVertex2f(-20.0f,-40.0f);
   glVertex2f(0.0f, -80.0f);
   glVertex2f(20.0f, -40.0f);

   glVertex2f(20.0f, -40.0f);
   glVertex2f(60.0f, -20.0f);
   glVertex2f(20.0f, 0.0f);

   glVertex2f(-20.0f, 0.0f);
   glVertex2f(-20.0f,-40.0f);
   glVertex2f(20.0f, 0.0f);

   glVertex2f(-20.0f,-40.0f);
   glVertex2f(20.0f, -40.0f);
   glVertex2f(20.0f, 0.0f);

glEnd();
```

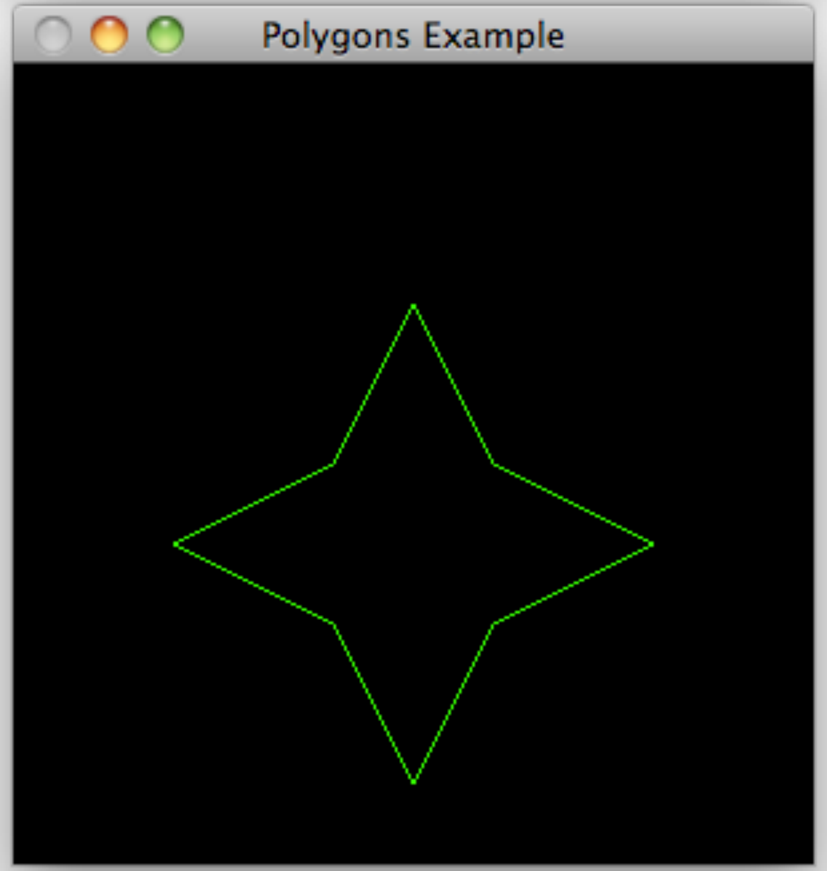


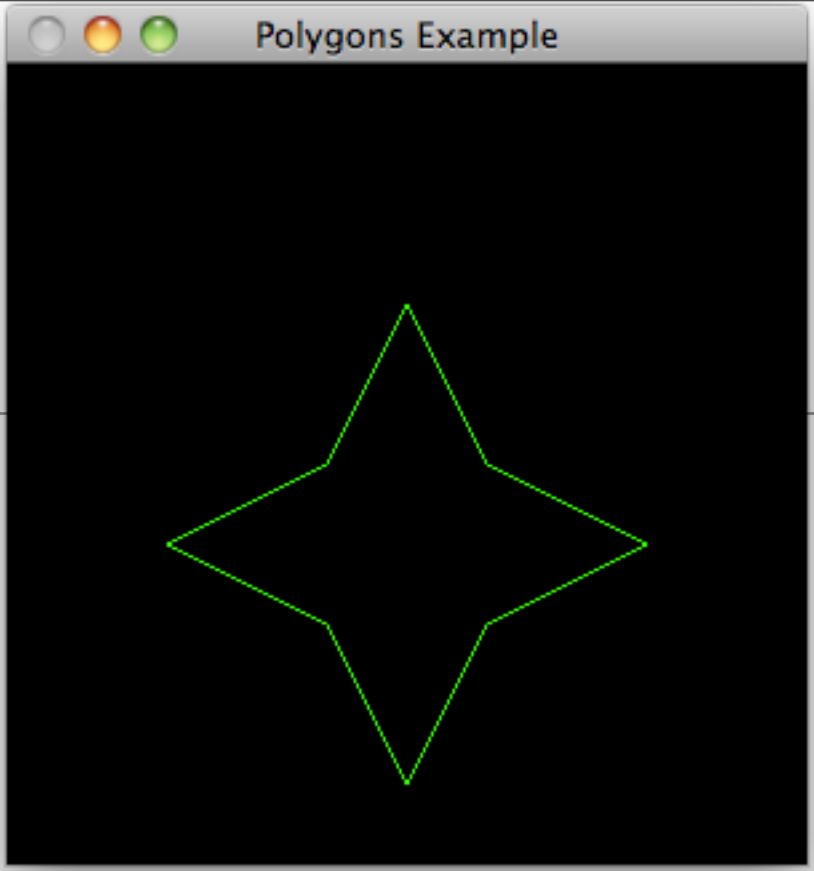
- When the polygons are filled, you won’t be able to see any edges and the figure will seem to be a single shape onscreen.

- However, if you use glPolygonMode to switch to an outline drawing, it is distracting to see all those little triangles making up some larger surface area.

- OpenGL provides a special flag called an edge flag to address those distracting edges.







- By setting and clearing the edge flag as you specify a list of vertices, you inform OpenGL which line segments are considered border lines (lines that go around the border of your shape) and which ones are not (internal lines that shouldn't be visible).
- The glEdgeFlag function takes a single parameter that sets the edge flag to True or False. When the function is set to True, any vertices that follow mark the beginning of a boundary line segment

```
 glBegin(GL_TRIANGLES);
   glEdgeFlag(GL_FALSE);
   glVertex2f(-20.0f, 0.0f);
   glEdgeFlag(GL_TRUE);
   glVertex2f(20.0f, 0.0f);
   glVertex2f(0.0f, 40.0f);

   glVertex2f(-20.0f,0.0f);
   glVertex2f(-60.0f,-20.0f);
   glEdgeFlag(GL_FALSE);
   glVertex2f(-20.0f,-40.0f);
   glEdgeFlag(GL_TRUE);

   glVertex2f(-20.0f,-40.0f);
   glVertex2f(0.0f, -80.0f);
   glEdgeFlag(GL_FALSE);
   glVertex2f(20.0f, -40.0f);
   glEdgeFlag(GL_TRUE);

   glVertex2f(20.0f, -40.0f);
   glVertex2f(60.0f, -20.0f);
   glEdgeFlag(GL_FALSE);
   glVertex2f(20.0f, 0.0f);
   glEdgeFlag(GL_TRUE);

   glEdgeFlag(GL_FALSE);
   glVertex2f(-20.0f, 0.0f);
   glVertex2f(-20.0f,-40.0f);
   glVertex2f(20.0f, 0.0f);

   glVertex2f(-20.0f,-40.0f);
   glVertex2f(20.0f, -40.0f);
   glVertex2f(20.0f, 0.0f);
   glEdgeFlag(GL_TRUE);
glEnd();
```