Introduction To OpenGL

V. Masilamani

OpenGL –What? and Why?

- An application programming interface (API)
- A (low-level) Graphics rendering API
- It considers primitive objects: points, line-segments, curves and polygons
- Cross-platform.
- Easier to learn compared to “Microsoft’s Direct3D (DirectX)”, Java3D
- Hardware-based device drivers widely supported.
- Captures the low-level pipeline

Primary Functionalities in OpenGL

- Geometric description of objects.
- Composition or lay-out of objects.
- Color specification and lighting calculations
- Rasterization or sampling – calculating the pixel color and depth values from the above mathematical descriptions
- User-interaction / user interfaces
- OpenGL can render(display) Geometric primitives, Bitmaps and Images

Naming Conventions

- OpenGL core functions are prefixed with gl
- OpenGL utility functions are prefixed with glu
- OpenGL typedef defined types are prefixed with GL
- OpenGL constants are all caps and prefixed with GL_

OpenGL Command Formats

- glVertex2f(x, y)

- number of
- Components/
- Dimensions

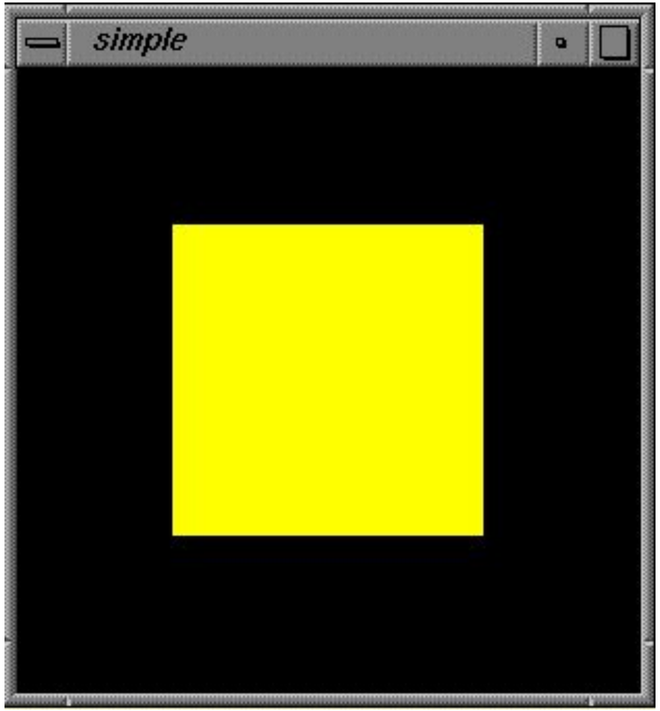
- b – byte
- ub – unsigned byte
- s – short
- us – unsigned short
- i – int

- Add 'v' for vector
- form

- glVertex2fv(v)

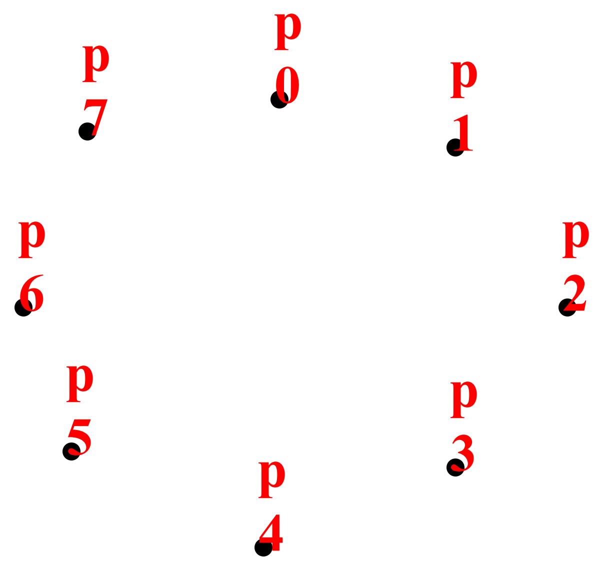
First Program using OpenGL –To display square

```
●   void Display()
    {
●     glColor3f(1.0f, 1.0f, 0.0f );
●     glBegin(GL_POLYGON);
         glVertex2f(-0.5f, -0.5f);
         glVertex2f(-0.5f,  0.5f);
         glVertex2f( 0.5f,  0.5f);
         glVertex2f( 0.5f, -0.5f);
      glEnd();
      glFlush();
●   }
```



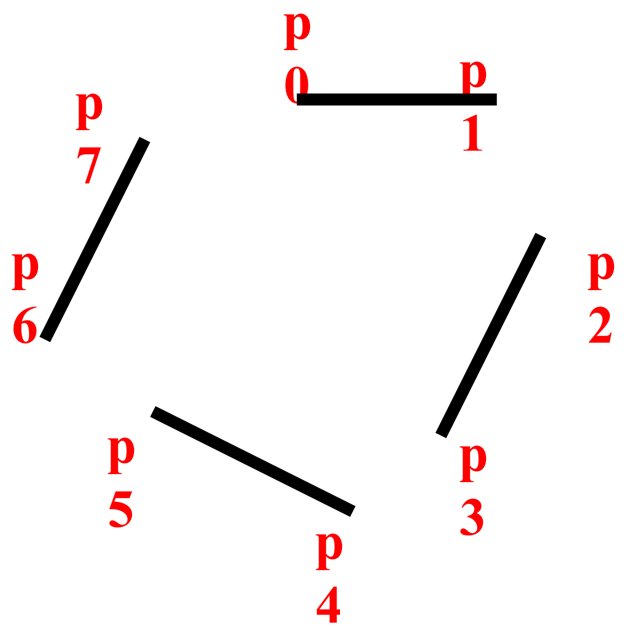
Plotting Points

```
glBegin(GL_POINTS);
   glVertex2fv(p0);
   glVertex2fv(p1);
   glVertex2fv(p2);
   glVertex2fv(p3);
   glVertex2fv(p4);
   glVertex2fv(p5);
   glVertex2fv(p6);
   glVertex2fv(p7);
glEnd();
```



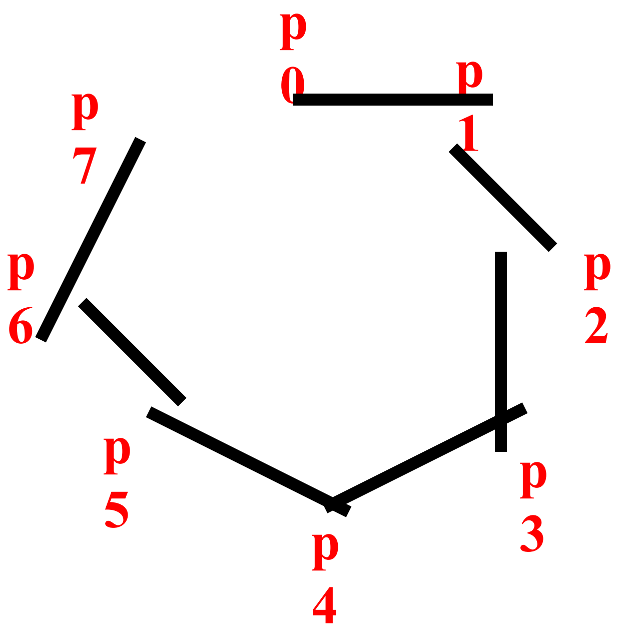
Drawing Line Segments

```
glBegin(GL_LINES);
   glVertex2fv(p0);
   glVertex2fv(p1);
   glVertex2fv(p2);
   glVertex2fv(p3);
   glVertex2fv(p4);
   glVertex2fv(p5);
   glVertex2fv(p6);
   glVertex2fv(p7);
glEnd();
```



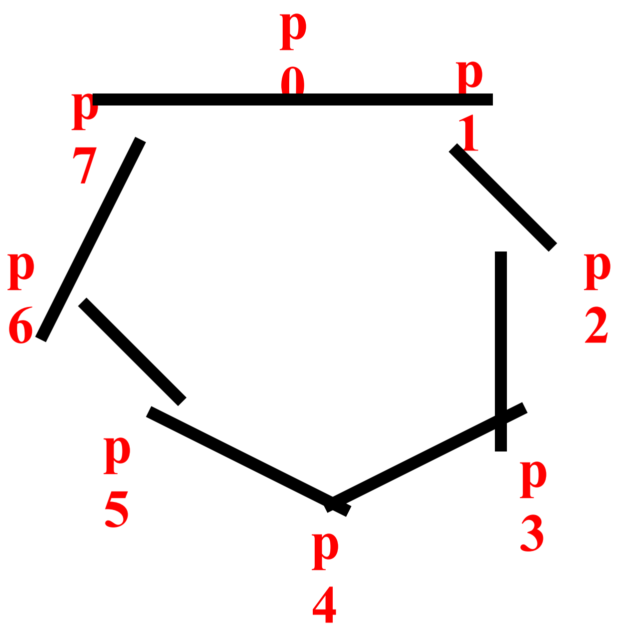
Drawing Polylines(line strip)

```
glBegin(GL_LINE_STRIP);
   glVertex2fv(p0);
   glVertex2fv(p1);
   glVertex2fv(p2);
   glVertex2fv(p3);
   glVertex2fv(p4);
   glVertex2fv(p5);
   glVertex2fv(p6);
   glVertex2fv(p7);
glEnd();
```



Drawing Line-Loop

```
glBegin(GL_LINE_LOOP);
   glVertex2fv(p0);
   glVertex2fv(p1);
   glVertex2fv(p2);
   glVertex2fv(p3);
   glVertex2fv(p4);
   glVertex2fv(p5);
   glVertex2fv(p6);
   glVertex2fv(p7);
glEnd();
```



Syntax to Specigy Geometric Primitives

- Primitives are specified using
  - glBegin(primType);
  - // define your vertices here
  - …
  - glEnd();

- primType: GL_POINTS, GL_LINES, GL_TRIANGLES, GL_QUADS, …

OpenGL: Front/Back Rendering

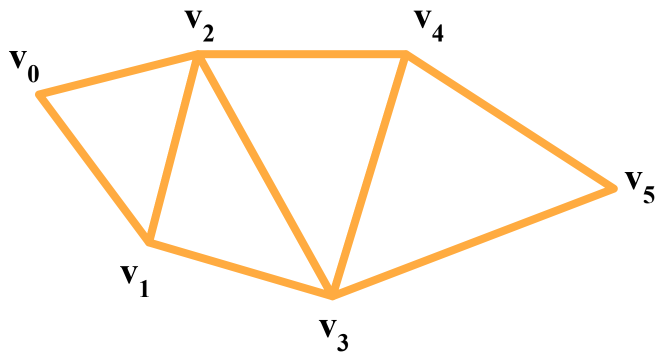
- Each polygon has two sides, front and back
- OpenGL can render the two differently
- The ordering of vertices in the list determines which is the front side
- When looking at the front side, the vertices go counter clock wise

Drawing Multiple Triangles

- You can draw multiple triangles between glBegin(GL_TRIANGLES) and glEnd():
  - float v1[3], v2[3], v3[3], v4[3];
  - glBegin(GL_TRIANGLES);
  - glVertex3fv(v1); glVertex3fv(v2); glVertex3fv(v3);
  - glVertex3fv(v1); glVertex3fv(v3); glVertex3fv(v4);
  - glEnd();
- The same vertex is used (sent, transformed, colored) many times (6 on average)

To Draw Triangle Strip

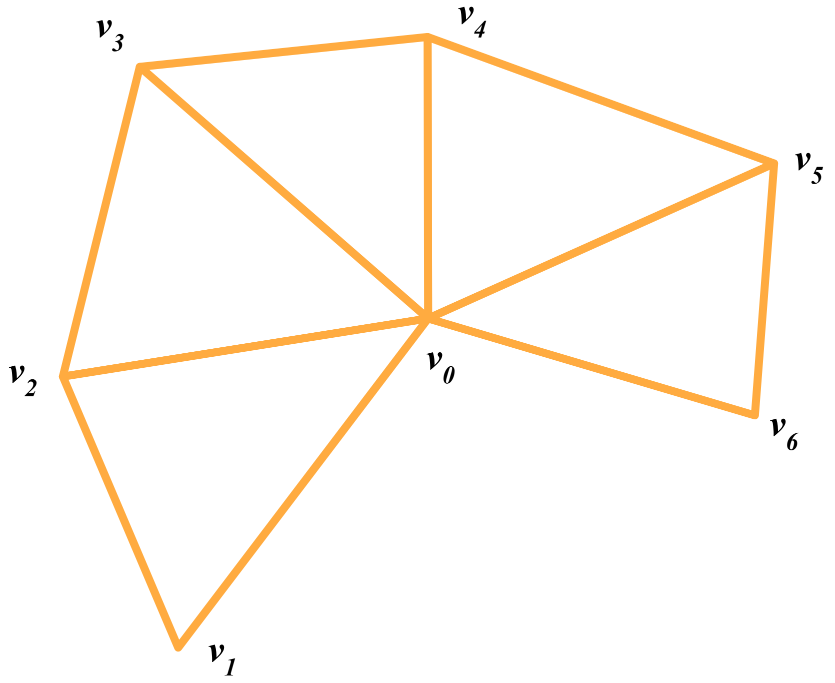
```
glBegin(GL_TRIANGLE_STRIP);

    glVertex3fv(v0);

    glVertex3fv(v1);

    glVertex3fv(v2);

    glVertex3fv(v3);

    glVertex3fv(v4);

    glVertex3fv(v5);
glEnd();
```



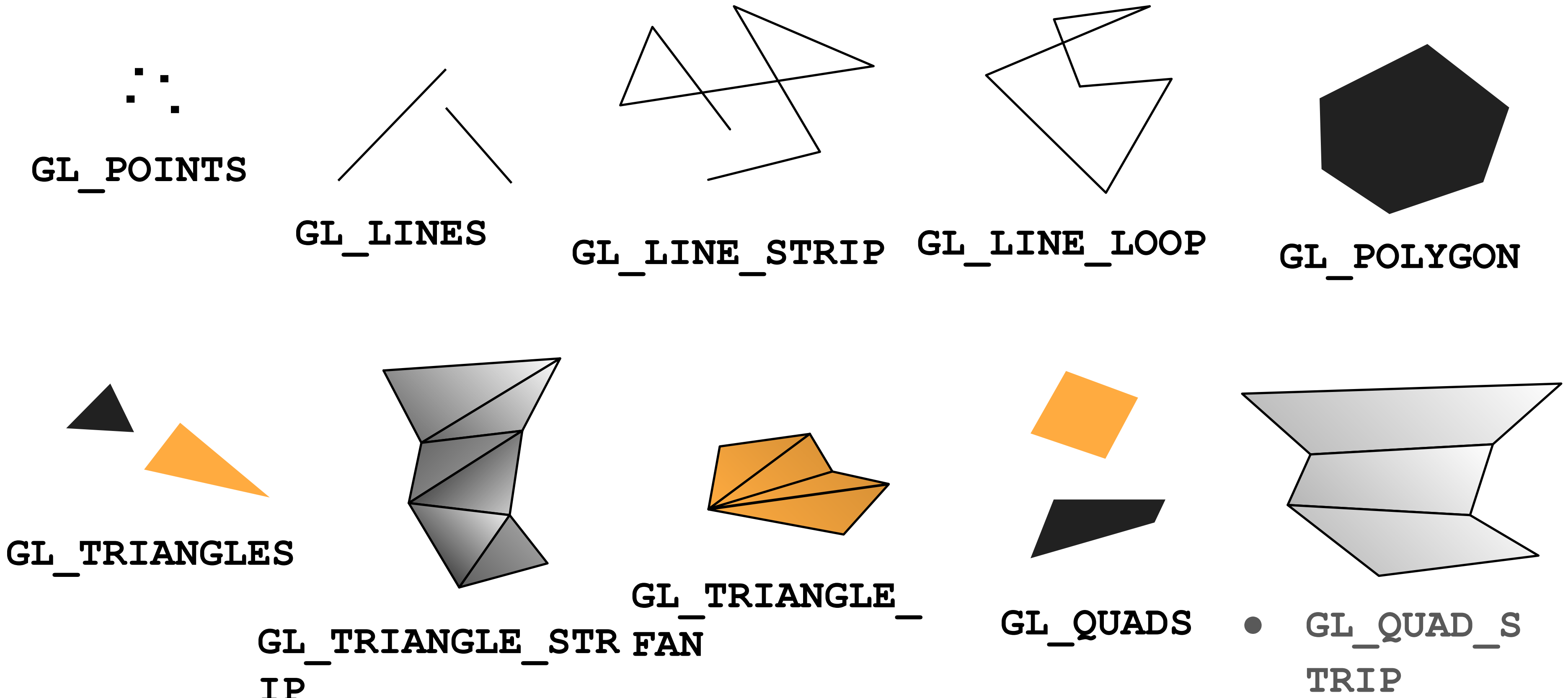
triangle 0 is v0, v1, v2
triangle 1 is v2, v1, v3 (*why not v1, v2, v3?*)
triangle 2 is v2, v3, v4
triangle 3 is v4, v3, v5 (again, **not** v3, v4, v5); Anti-clock wise; start from Top-Left

To Draw Triangle Fan

```
glBegin(GL_TRIANGLE_STRIP);

        glVertex3fv(v0);

        glVertex3fv(v1);

        glVertex3fv(v2);

        glVertex3fv(v3);

        glVertex3fv(v4);

        glVertex3fv(v5);

       glVertex3fv(v5);    glEnd();
```

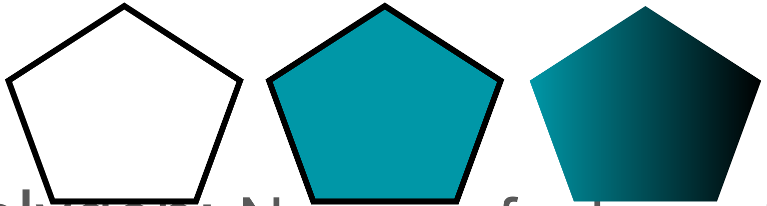


All primitives –Represented by vertices

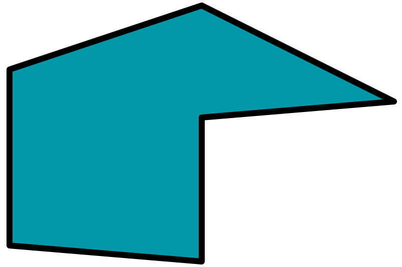


Polygons: Simple Vs Non Simple

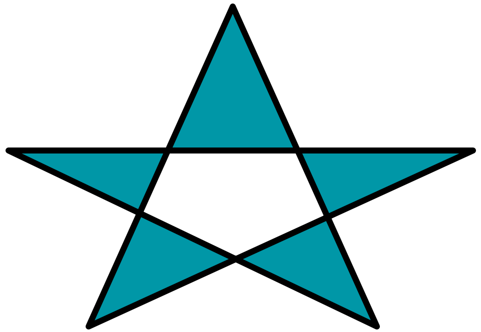
- Polygon: Object that is closed as in a line loop, but that has an interior



- Simple Polygon: No pair of edges of a polygon cross each other
- Simple:

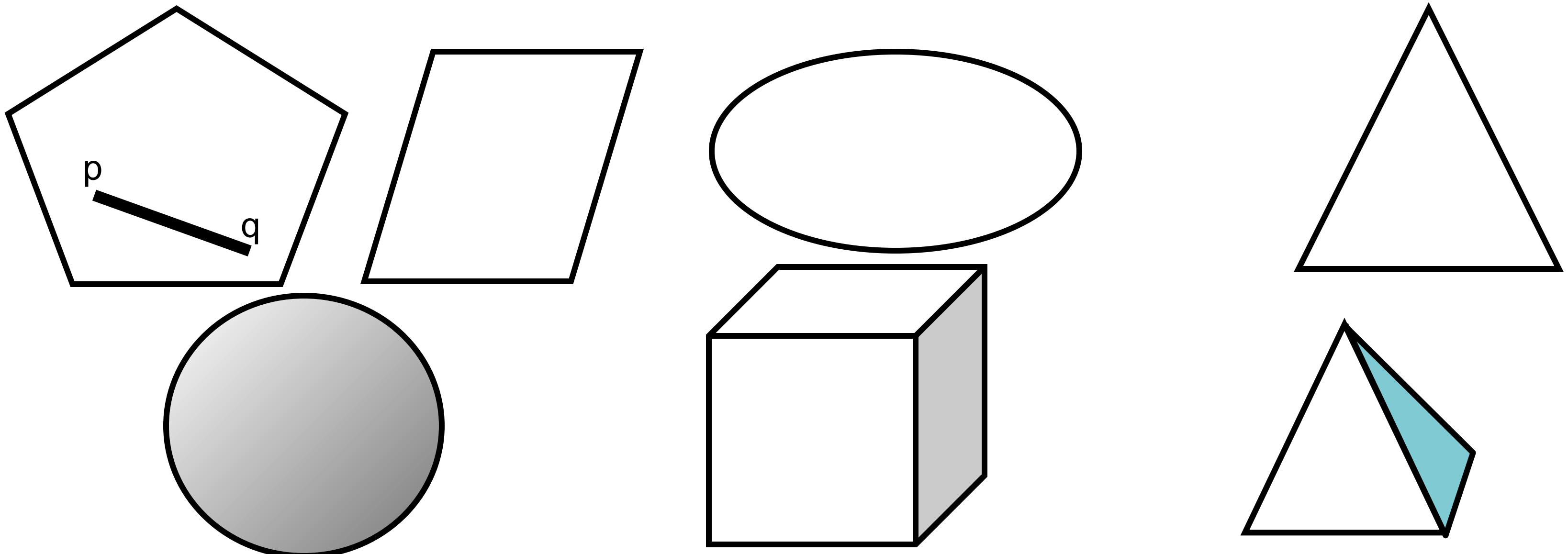


Non Simple:



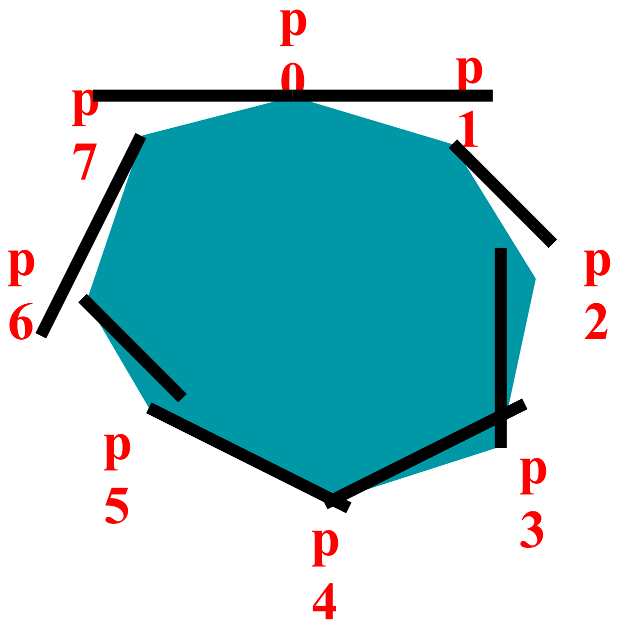
Convex Objects

- Defn: For every pair of points (p,q) in the object, If all points on the line segment joining p and q are inside the object, or on its boundary, then the object is convex



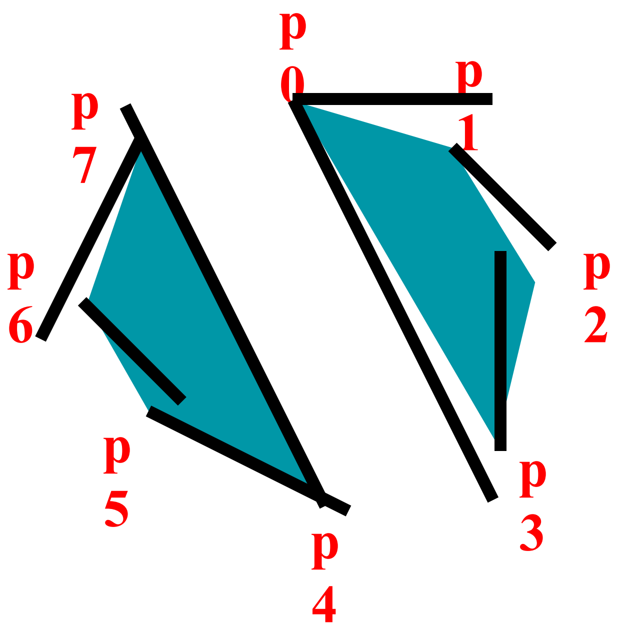
Drawing Polygon

```
glBegin(GL_POLYGON);
   glVertex2fv(p0);
   glVertex2fv(p1);
   glVertex2fv(p2);
   glVertex2fv(p3);
   glVertex2fv(p4);
   glVertex2fv(p5);
   glVertex2fv(p6);
   glVertex2fv(p7);
glEnd();
```



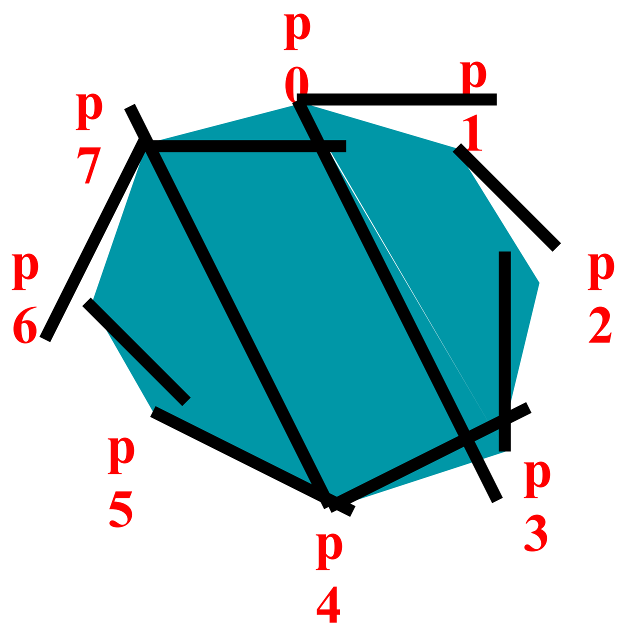
Drawing Quadrilaterals

```
glBegin(GL_QUADS);
   glVertex2fv(p0);
   glVertex2fv(p1);
   glVertex2fv(p2);
   glVertex2fv(p3);
   glVertex2fv(p4);
   glVertex2fv(p5);
   glVertex2fv(p6);
   glVertex2fv(p7);
glEnd();
```



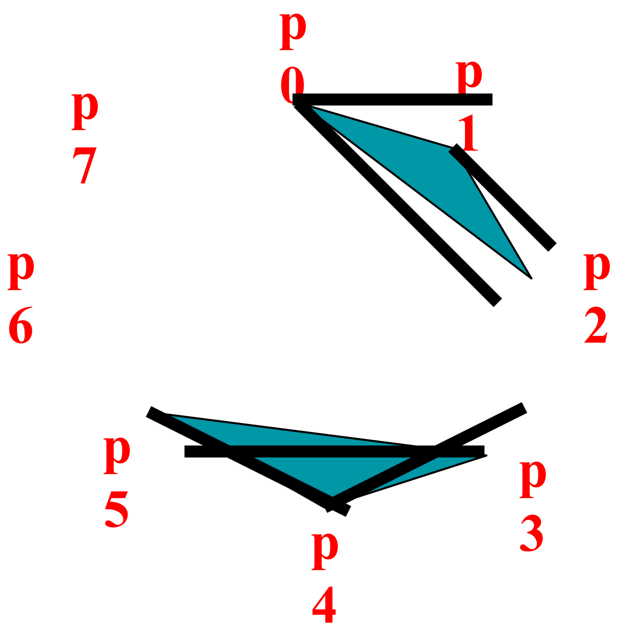
Drawing Quadrilateral strip

```
glBegin(GL_QUAD_STRIP);
   glVertex2fv(p1);
   glVertex2fv(p2);
   glVertex2fv(p3);
   glVertex2fv(p0);
   glVertex2fv(p4);
   glVertex2fv(p7);
   glVertex2fv(p5);
   glVertex2fv(p6);
glEnd();
```



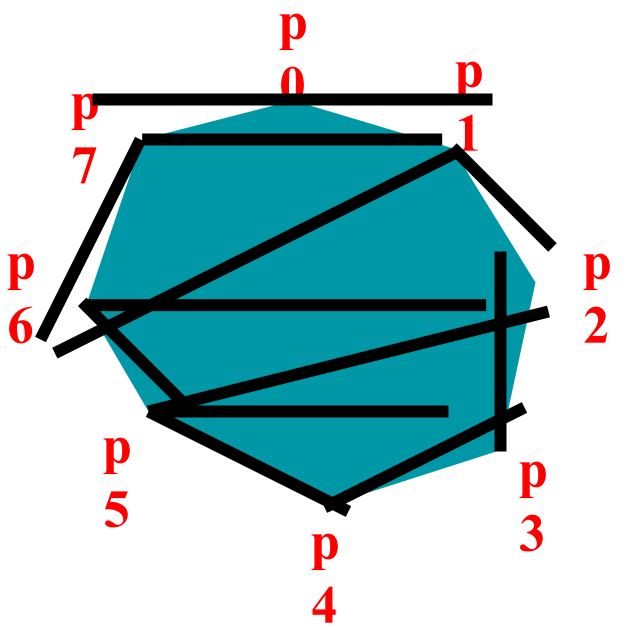
Drawing Triangle

```
glBegin(GL_TRIANGLES);
   glVertex2fv(p0);
   glVertex2fv(p1);
   glVertex2fv(p2);
   glVertex2fv(p3);
   glVertex2fv(p4);
   glVertex2fv(p5);
   glVertex2fv(p6);
   glVertex2fv(p7);
glEnd();
```



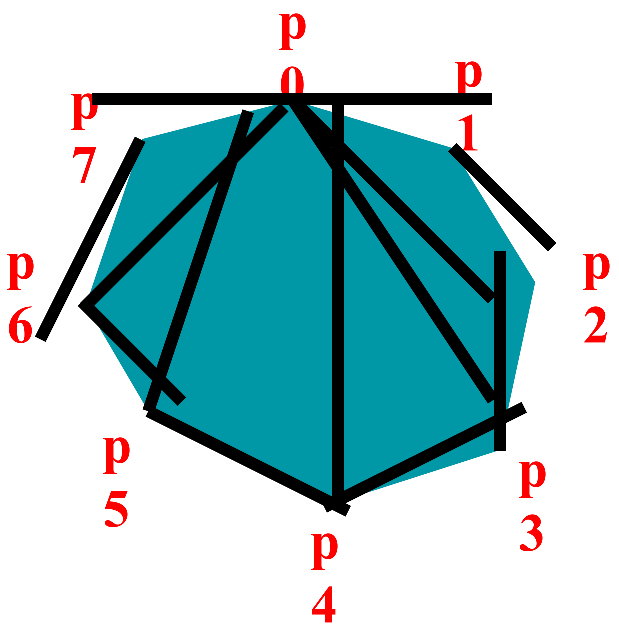
Drawing Triangle Strip

```
glBegin(GL_TRIANGLE_STRIP);
   glVertex2fv(p0);
   glVertex2fv(p7);
   glVertex2fv(p1);
   glVertex2fv(p6);
   glVertex2fv(p2);
   glVertex2fv(p5);
   glVertex2fv(p3);
   glVertex2fv(p4);
glEnd();
```



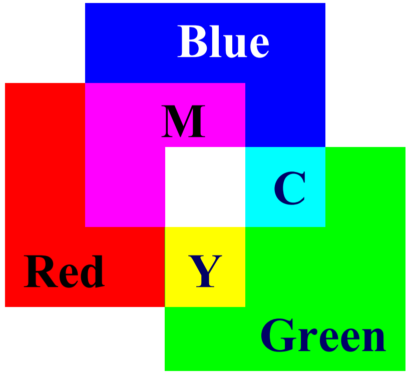
Drawing Triangle Fan

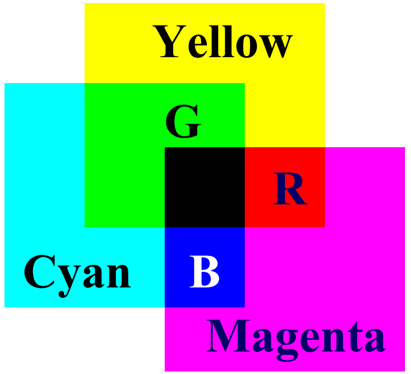
```
glBegin(GL_TRIANGLE_FAN);
   glVertex2fv(p0);
   glVertex2fv(p1);
   glVertex2fv(p2);
   glVertex2fv(p3);
   glVertex2fv(p4);
   glVertex2fv(p5);
   glVertex2fv(p6);
   glVertex2fv(p7);
glEnd();
```

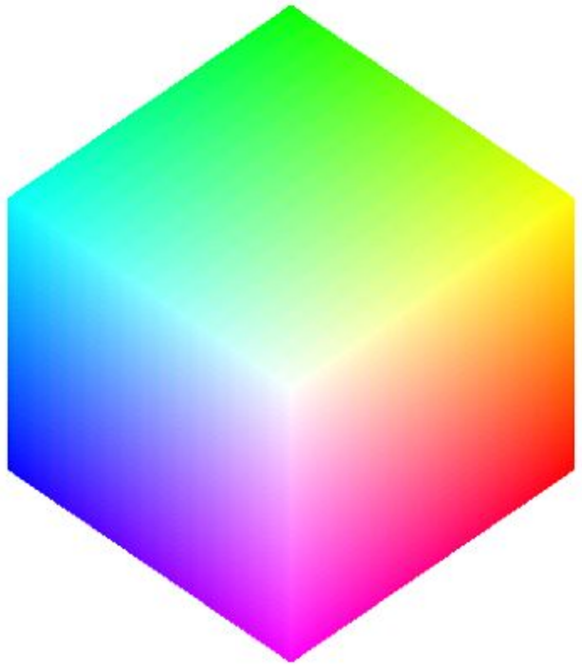


Attributes of Rendering

- Color, pattern of filling, etc.







OpenGL’s State Machine

- All rendering attributes are encapsulated in the OpenGL State
  - rendering styles
  - shading
  - lighting
  - texture mapping

Manipulating OpenGL State

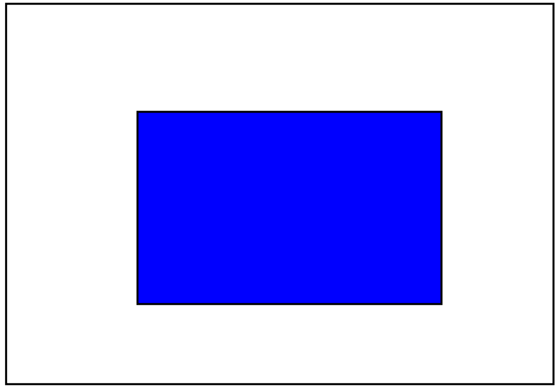
- Appearance is controlled by current state
    - for each (primitive to render) {
        - update OpenGL state
        - render primitive }
- Manipulating vertex attributes is the most common way to manipulate state
        - glColor*() / glIndex*()
        - glNormal*()
        - glTexCoord*()

Controlling current state

- Setting State
    - glPointSize(size);
    - glLineStipple(repeat, pattern);
    - glShadeModel(GL_SMOOTH);
- Enabling Features
    - glEnable(GL_LIGHTING);
    - glDisable(GL_TEXTURE_2D

Specifying Colour Attribute

```
Void DrawBlueQuad( )
{
  glColor3f(0.0f, 0.0f, 1.0f);
  glBegin(GL_QUADS);

        glVertex2f(0.0f, 0.0f);

        glVertex2f(1.0f, 0.0f);

        glVertex2f(1.0f, 1.0f);

        glVertex2f(0.0f, 1.0f);
  glEnd();
}
```



This type of operation is called *immediate-mode rendering*;

- Each command happens immediately
- Although you may not see the result if you use double buffering
  - Things get drawn into the back buffer

Specifying Colour attribute

```
glColor3f(0.1, 0.5, 1.0);

glVertex3fv(v0); glVertex3fv(v1); glVertex3fv(v2);
```

- To produce a smoothly shaded triangle:

```
glColor3f(1, 0, 0); glVertex3fv(v0);

glColor3f(0, 1, 0); glVertex3fv(v1);

glColor3f(0, 0, 1); glVertex3fv(v2);
```

- In OpenGL, colors can also have a fourth component α (opacity or 1-transparency); Generally want α = 1.0 (opaque);