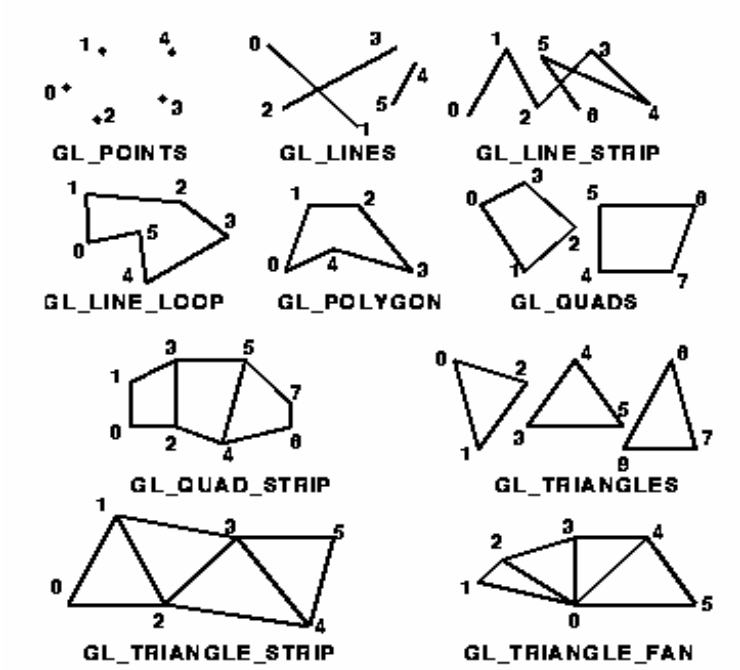
Lectures
OpenGL Primitives

By
Tom Duff
Pixar Animation Studios
Emeryville, California
and
George Ledin Jr
Sonoma State University
Rohnert Park, California

OpenGL Primitives

- In OpenGL, the programmer is provided the following primitives for use in constructing geometric objects.

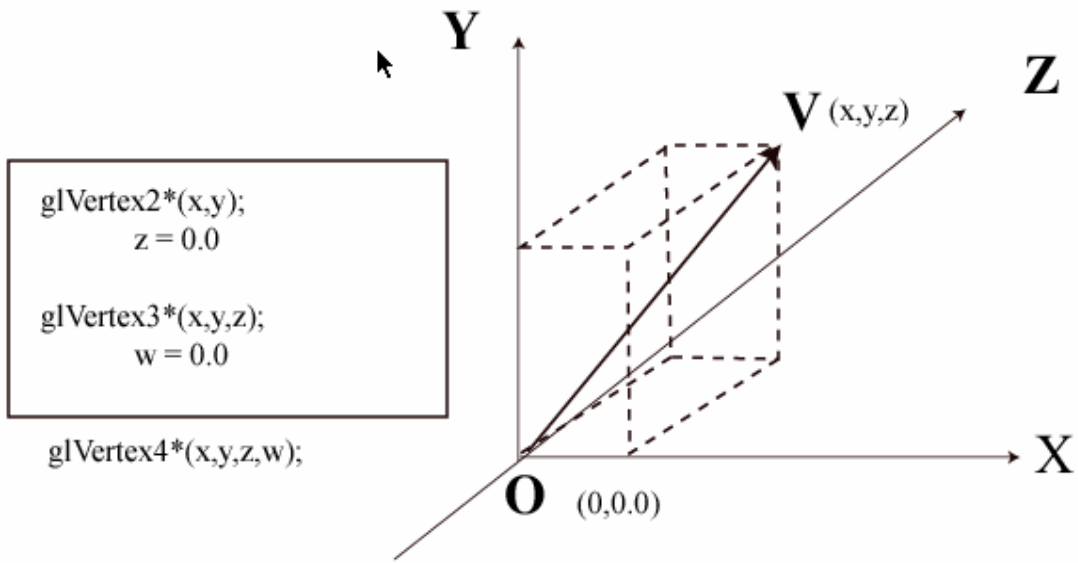


- Each geometric object is described by a set of vertices and the type of primitive to be drawn. Whether and how the vertices are connected is determined by the primitive type.

OpenGL Primitives, continued (Vertex Functions)

- The command glVertex*() is used to specify a vertex. Here are some sample uses of glVertex*():

- glVertex2s(1, 2);
- glVertex3d(0.0, 0.0, 3.1415926535898);
- glVertex4f(1.3, 2.0, -4.2, 1.0);
- GLdouble vector[3] = {3.0, 10.0 2033.0}; glVertex3dv(vector);

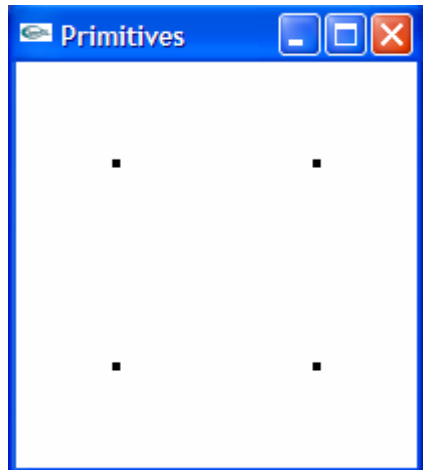


OpenGL Primitives, continued

- All calls to glVertex*() should occur between a glBegin() and glEnd() pair.
  - void glBegin (GLenum *mode*);
  - void glEnd();
    - delimit the vertices of a primitive or a group of like primitives
    - ***Mode***: Specifies the primitive or primitives that will be created from vertices presented between glBegin and the subsequent glEnd.
- The order in which the vertices are declared is very important.
- Some primitives, when given an incorrect number of vertices, will ignore any extra vertices.
  - For example, **GL_TRIANGLES** only draws the triangle corresponding to vertices 1, 2, and 3. Vertices 4 and 5 are ignored.

OpenGL Primitives – GL_POINTS

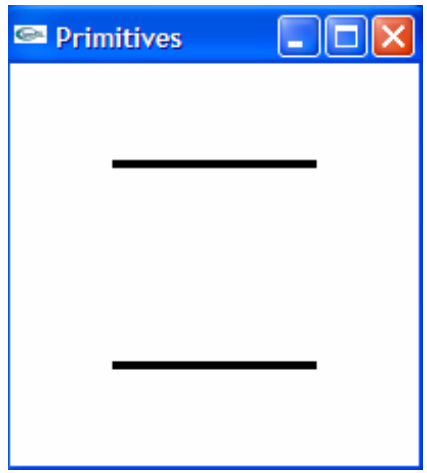
- GL_POINTS
  - Treats each vertex as a single point.
- void glPointSize(GLfloat size);
  - Specify the diameter of rasterized points
- void glVertex2f(TYPE xcoordinate, TYPE ycoordinate)
  - Specify the location of a vertex in 2D.




```
void display(void)
{
…
/* Note that in this program, world coordinate (0,0) is
     the center of the screen */
/* draw 4 white points, centered at (0,0) */
     glColor3f(0.0,0.0,0.0);

/* specify point to be 4 pixels thick */
     glPointSize(4);
     glBegin(GL_POINTS);
            glVertex2f (-0.5, -0.5);
            glVertex2f (0.5, -0.5);
            glVertex2f (0.5, 0.5);
            glVertex2f (-0.5, 0.5);
     glEnd();
…
}
```

OpenGL Primitives – GL_LINES

- GL_LINES
  - Treats each pair of vertices as an independent line segment.
- void glLineWidth(GLfloat *width*)
  - specify the width of rasterized lines

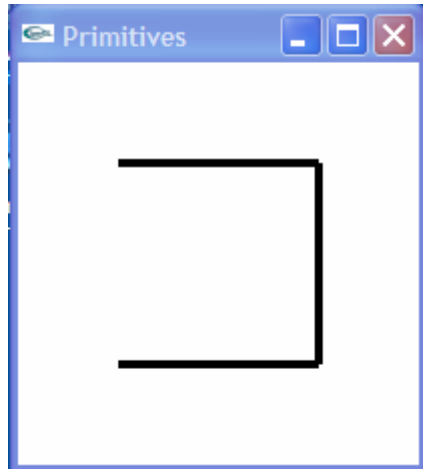



```
void display(void)
{
…
/* Note that in this program, world coordinate (0,0) is the
     center of the screen */

/* draw 2 white lines */
     glColor3f(0.0,0.0,0.0);

     glBegin(GL_LINES);

             glVertex2f (-0.5, -0.5); /* draw the bottom line */
             glVertex2f (0.5, -0.5);

             glVertex2f (0.5, 0.5);  /* draw the top line */
             glVertex2f (-0.5, 0.5);

     glEnd();…
}
```

OpenGL Primitives – GL_LINE_STRIP

- GL_LINE_STRIP
  - Draws a connected set of line segments from the first vertex to the last.

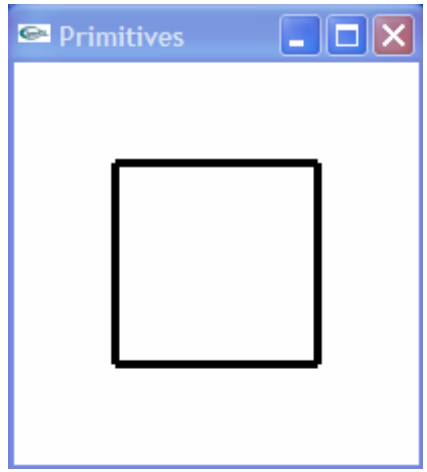
```
void display(void)
{
…
/* Note that in this program, world coordinate
     (0,0) is the center of the screen */

/* connecting 4 points from first vertex to the
     last */
     glColor3f(0.0,0.0,0.0);

     glLineWidth(4);
     glBegin(GL_LINE_STRIP);
            glVertex2f (-0.5, -0.5);
            glVertex2f (0.5, -0.5);
            glVertex2f (0.5, 0.5);
            glVertex2f (-0.5, 0.5);
     glEnd();
…
}
```



OpenGL Primitives – GL_LINE_LOOP

- GL_LINE_LOOP
  - Draws a connected set of line segments from the first vertex to the last, then back to the first.

```
void display(void)
{
…
/* Note that in this program, world
    coordinate (0,0) is the center of the
    screen */

/* connecting 4 points with white lines
    in a loop */
    glColor3f(0.0,0.0,0.0);

    glLineWidth(4);
    glBegin(GL_LINE_LOOP);
          glVertex2f (-0.5, -0.5);
          glVertex2f (0.5, -0.5);
          glVertex2f (0.5, 0.5);
          glVertex2f (-0.5, 0.5);
    glEnd();
…
}
```



OpenGL Primitives – GL_TRIANGES

- GL_TRIANGES
  - Treats each set of three vertices as an independent triangle.

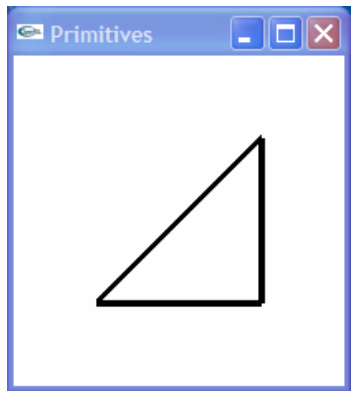
```
void display(void)
{
…
/* Note that in this program, world
     coordinate (0,0) is the center of the
     screen */

/* connecting 4 points with white lines in a
     loop */
     glColor3f(0.0,0.0,0.0);

/* draw only the outline of polygon */
     glPolygonMode(GL_FRONT, GL_LINE);

     glLineWidth(4);
     glBegin(GL_TRIANGES);
            glVertex2f (-0.5, -0.5);
            glVertex2f (0.5, -0.5);
            glVertex2f (0.5, 0.5);
            glVertex2f (-0.5, 0.5);
     glEnd();
…
}
```

- void glPolygonMode(GLenum *face*, GLenum *mode*)

  - *face*: Specifies the polygons that *mode* applies to. Must be **GL_FRONT** for front-facing polygons, **GL_BACK** for back-facing polygons, or **GL_FRONT_AND_BACK** for front- and back-facing polygons.

  - *mode:* Specifies the way polygons will be rasterized. Accepted values are **GL_POINT**, **GL_LINE**, and **GL_FILL**.

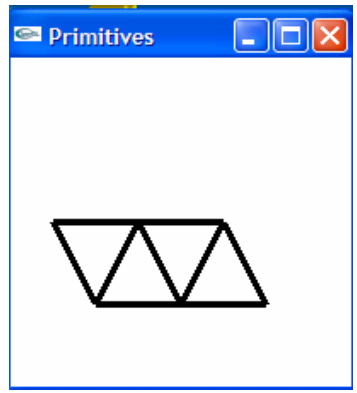
  -The default is **GL_FILL** for both front- and back-facing polygons.



OpenGL Primitives – GL_TRIANGE_STRIP

- GL_TRIANGE_STRIP
  - Draws a connected set of triangles. One triangle is defined for each vertex presented after the first two vertices. Note: Order of points does matter

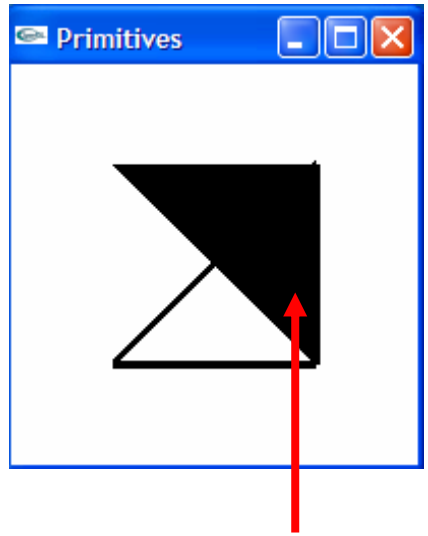
```
void display(void)
{
…
/* Note that in this program, world coordinate (0,0) is the center of the screen
     */

/* connecting 3 points to form two triangles  */
     glColor3f(0.0,0.0,0.0);

/* draw only the outline of polygon */
     glPolygonMode(GL_FRONT, GL_LINE);

     glLineWidth(4);
     glBegin(GL_TRIANGES_STRIP);
            glVertex2f (-0.75, 0.0);
            glVertex2f (-0.5, -0.5);
            glVertex2f (-0.25, 0.0);

            glVertex2f (0, -0.5);  //create 2nd triange
                        glVertex2f (0.25, 0); //create 3rd triange
                        glVertex2f (0.5, -0.5); //create 4th triange

     glEnd();
…
}
```



OpenGL Primitives – GL_TRIANGE_STRIP, continued

- GL_TRIANGE_STRIP
  - Draws a connected set of triangles. One triangle is defined for each vertex presented after the first two vertices.
  - Note:
    - Order of points does matter!
    - If the vertices are defined clockwise, the front of the polygon will be shown. Otherwise, the back of the polygon will be shown.

```
void display(void) {
…
/* Note that in this program, world coordinate (0,0) is the
     center of the screen */

/* connecting 4 points to form two triangles  */
     glColor3f(0.0,0.0,0.0);

/* draw only the outline of polygon */
     glPolygonMode(GL_FRONT, GL_LINE);

     glLineWidth(4);
     glBegin(GL_TRIANGES_STRIP);
             glVertex2f (-0.5, -0.5);
             glVertex2f (0.5, -0.5);
             glVertex2f (0.5, 0.5);
     glVertex2f (-0.5, 0.5);
     glEnd();
…
}
```



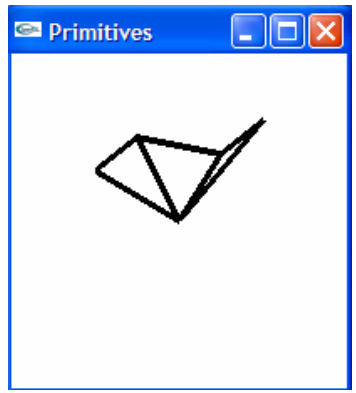
Because the vertex is defined counter clockwise, the back of the polygon is shown.

Therefore, this Part of polygon is black.

OpenGL Primitives – GL_TRIANGE_FAN

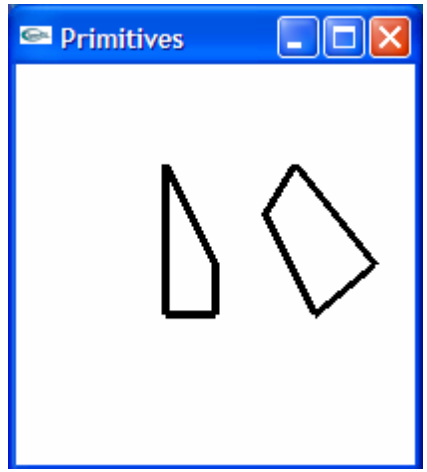
- GL_TRIANGE_FAN
  - Draws a connected set of triangles. One triangle is defined for each vertex presented after the first two vertices. Note: Order of points does matter!

```
void display(void)
{
…
/* Note that in this program, world coordinate (0,0) is the center of the
     screen */

/* connecting 4 points to form two triangles  */
     glColor3f(0.0,0.0,0.0);

/* draw only the outline of polygon */
     glPolygonMode(GL_FRONT, GL_LINE);

     glLineWidth(4);
     glBegin(GL_FAN);
            glVertex2f (0.0, 0.0);  // create 1st triangle
            glVertex2f (0.5, 0.6);
                        glVertex2f (0.25, 0.4);

                        glVertex2f (-0.25, 0.5); // create 2nd triangle
            glVertex2f(-0.5, 0.3); // create 3rd triangle
     glEnd();
…
}
```



OpenGL Primitives – GL_QUADS

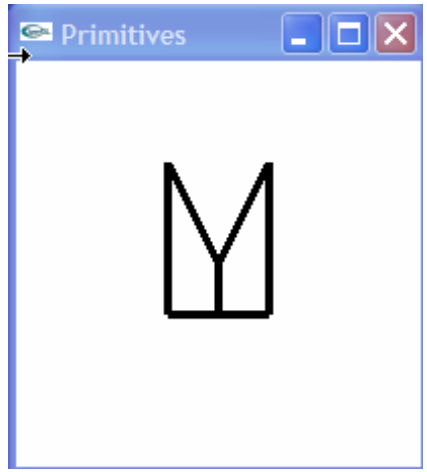
- GL_TRIANGE_QUADS
  - Treats each set of four vertices as an independent quadrilateral. Note: Order of points does matter!




```
void display(void)
{
…
/* Note that in this program, world coordinate
     (0,0) is the center of the screen */

/* creating 2 quadrilaterals */
     glColor3f(0.0,0.0,0.0);

     glPolygonMode(GL_FRONT, GL_LINE);
     glLineWidth(4);
     glBegin(GL_QUADS);
            glVertex2f (-0.25, -0.25);
            glVertex2f (0.0, -0.25);
                        glVertex2f (0.0, 0.0);
                        glVertex2f (-0.25, 0.5);

            glVertex2f (0.25, 0.25);
            glVertex2f (0.5, -0.25);
                        glVertex2f (0.8, 0.0);
                        glVertex2f (0.4, 0.5);
     glEnd();
…
}
```

OpenGL Primitives – GL_QUAD_STRIP

- GL_TRIANGE_QUAD_STRIP
  - Draws a connected set of quadrilaterals. One quadrilateral is defined for each pair of vertices presented after the first pair.




```
void display(void)
{
…
/* Note that in this program, world coordinate
     (0,0) is the center of the screen */

/* creating 2 quadrilaterals using GL_QUAD_STRIP */
     glColor3f(0.0,0.0,0.0);

     glPolygonMode(GL_FRONT, GL_LINE);
     glLineWidth(4);
     glBegin(GL_QUADS_STRIP);
            glVertex2f (-0.25, 0.5);
            glVertex2f (-0.25, -0.25);
                        glVertex2f (0.0, 0.0);
                        glVertex2f (0.0, -0.25);

            glVertex2f (0.25, 0.5);
            glVertex2f (0.25, -0.25);
     glEnd();
…
}
```

OpenGL Primitives – GL_POLYGON

- GL_TRIANGE_QUAD_STRIP
  - Draws a single, concave polygon. Vertices 1 through *N* define this polygon.

```
void display(void)
{
…
/* Note that in this program, world coordinate
     (0,0) is the center of the screen */

/* creating a six sided polygon */
     glColor3f(0.0,0.0,0.0);

     glPolygonMode(GL_FRONT, GL_LINE);
     glLineWidth(4);
     glBegin(GL_QUADS);
            glVertex2f (-0.25, 0.5);
            glVertex2f (-0.25, -0.25);
                        glVertex2f (0.0, 0.0);
                        glVertex2f (0.0, -0.25);

            glVertex2f (0.25, 0.5);
            glVertex2f (0.25, -0.25);
     glEnd();
…
}
```