Computer Graphics

Chapter Three

Faculty of Artificial Intelligence
Autonomous Systems Department

Content

1. Coordinate Reference Frames
2. Specifying a Two-Dimensional World-Coordinate Reference Frame in OpenGL
3. OpenGL Point Functions
4. OpenGL Line Functions
5. OpenGL Curve Functions
6. Fill-Area Primitives
7. Polygon Fill Areas
8. OpenGL Polygon Fill-Area Functions.

Definitions

- Describing various picture components of a virtual scene is one of the first things needed when generated a computer picture .
  - To provide the shape or structure of the individual objects
  - To provide their coordinate locations in the scene
- **We need:**
- **1) graphics output primitive** that are functions in CG API describe such picture components
- **2) Geometric primitives to d**efine the geometry of objects such as Lines, Triangles, Quadrics, Conic sections, Curved surfaces, ...

Coordinate Reference Frames

- The coordinate positions are stored in the scene description alongside the information about the objects, such as their color and their **coordinate extents.**
- **Coordinate extents** are the lowest and highest *x*, and *z* values for each object.
- A set of coordinate extents is also described as a **bounding box** for an object.
- For a two-dimensional figure, the coordinate extents are sometimes called an object's **bounding rectangle.**

Screen Coordinates

- Locations on a video monitor are referenced in integer **screen coordinates,** which correspond to the pixel positions in the frame buffer.

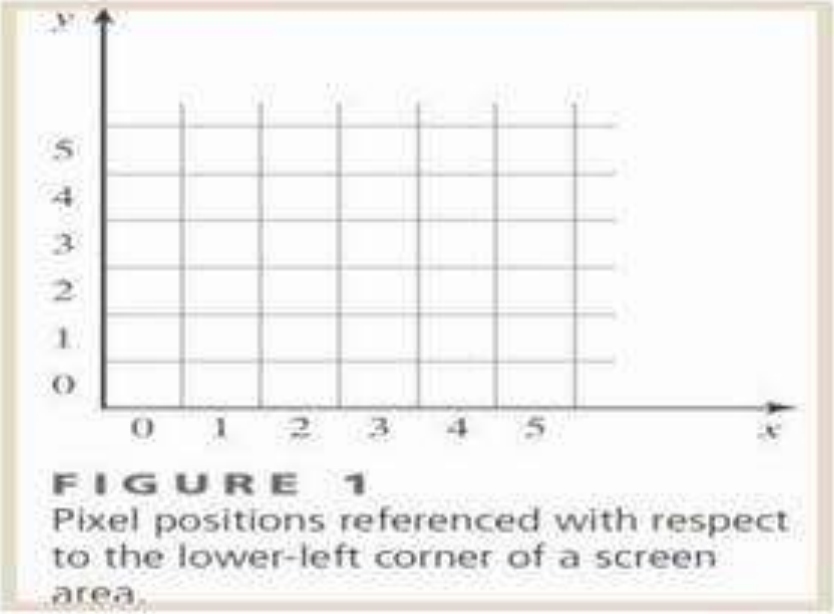

FIGURE 1
Pixel positions referenced with respect to the lower-left corner of a screen area.

Screen Coordinates

- Given the low-level procedure of the form **`setPixel (x, y);`**
- This procedure stores the current color setting into the frame buffer at integer position (x, y), relative to the selected position of the screen-coordinate origin.

Screen Coordinates

- To retrieve the current frame-buffer setting for a pixel location, use the following low-level function for obtaining a frame-buffer color value:

```
getPixel (x, y, color);
```

- In this function, parameter **color** receives an integer value corresponding to the combined red, green, and blue (RGB) bit codes stored for the specified pixel at position (*x*, *y*).

Specifying A Two-Dimensional World-Coordinate Reference Frame in OpenGL

- To set up any two-dimensional Cartesian reference frame, use the function **gluOrtho2D**.
- This function specifies an orthogonal projection, we need to ensure that the coordinate values are placed in the OpenGL projection matrix.
- We could also assign the identity matrix as the projection matrix before defining the world-coordinate range.

Specifying A Two-Dimensional World-Coordinate Reference Frame in OpenGL

- In the two-dimensional examples, define the coordinate frame for the screen display window with the following:

```
glMatrixMode (GL_PROJECTION);
glLoadIdentity ( );
gluOrtho2D (xmin, xmax, ymin, ymax);
```

- The display window is referenced by (**xmin, ymin**) coordinates at the lower-left corner, and by (**xmax, ymax**) coordinates at the upper-right corner, as shown in Figure 2.

A Two-Dimensional World-Coordinate Frame in OpenGL

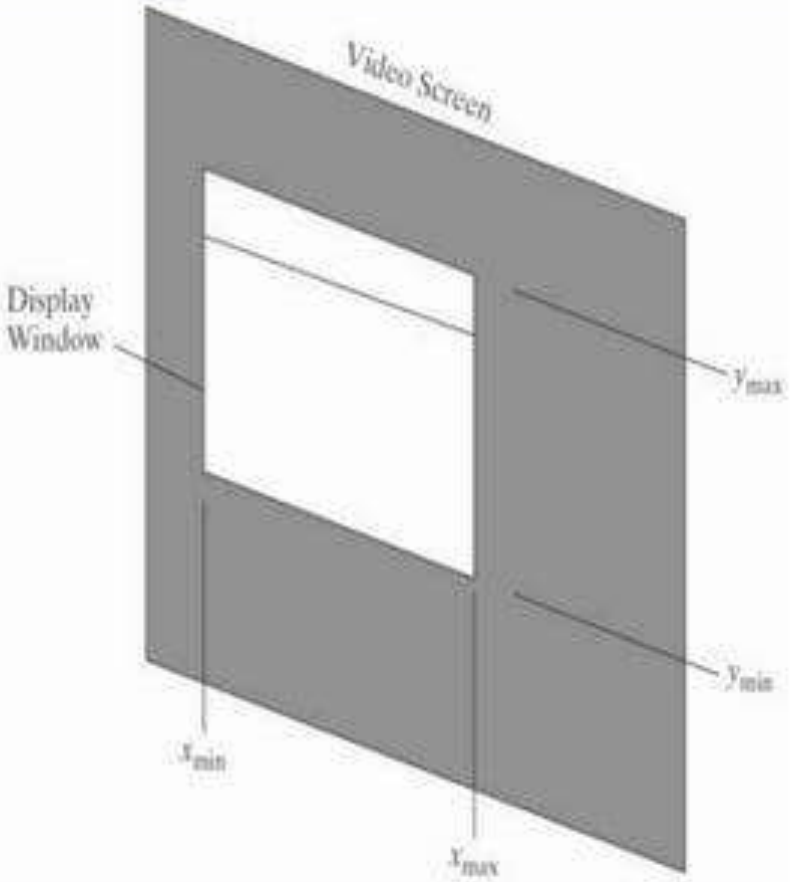

FIGURE 2
World-coordinate limits for a display window, as specified in the `glOrtho2D` function.

OpenGl Point Functions

- To state the coordinate values for a single position, use the OpenGL function :
  glVertex* ( );

- The asterisk (*) suffix codes are used in this function to identify the spatial dimension, the numerical data type to be used for the coordinate values, and a possible vector form for the coordinate specification.

OpenGl Point Functions

- Calls to **glVertex** functions must be placed between a **glBegin** function and a **glEnd** function.
- The argument of the **glBegin** function is used to identify the kind of geometric output primitive that is to be displayed, and **glEnd** takes no arguments.
- For point plotting, the argument of the **glBegin** function is the symbolic constant **GL_POINTS**.
- Thus, the form for an OpenGL specification of a point position is:

```
glBegin (GL_POINTS);
   glVertex* ( );
     glEnd ( );
```

OpenGl Point Functions

- Coordinate positions in OpenGL can be given in two, three, or four dimensions.
- We use a suffix value of 2, 3, or 4 on the **glVertex** function to indicate the dimensionality of a coordinate position.
- A four-dimensional specification indicates a ***homogeneous-coordinate*** representation, where the fourth coordinate *parameter h* is a scaling factor for the Cartesian-coordinate values.
- Homogeneous-coordinate representations are useful for expressing transformation operations in Matrix form.

OpenGl Point Functions

- Because OpenGL treats two-dimensions as a special case of three dimensions, any (*x*, *y*) coordinate specification is equivalent to a three-dimensional specification of (*x*, *y*, 0).
- We need to specify which data type to be used for the numerical value specifications of the coordinates.
- This is accomplished with a second suffix code on the **glVertex** function.
- Suffix codes for specifying a numerical data type are **i** (integer), **s** (short), **f** (float), and **d** (double).

OpenGl Point Functions

- Finally, the coordinate values can be listed explicitly in the **glVertex** function, or a single argument can be used that references a coordinate position as an array.

- If we use an array specification for a coordinate position, we need to append **v** (for "vector") as a third suffix code.

OpenGl Point Functions

- In this example, three equally spaced points are plotted along a two dimensional, straight-line path, where coordinates are given as integer pairs:

```
glBegin (GL_POINTS);
glVertex2i (50, 100);
glVertex2i (75, 150);
glVertex2i (100, 200);
glEnd ( );
```

OpenGl Point Functions

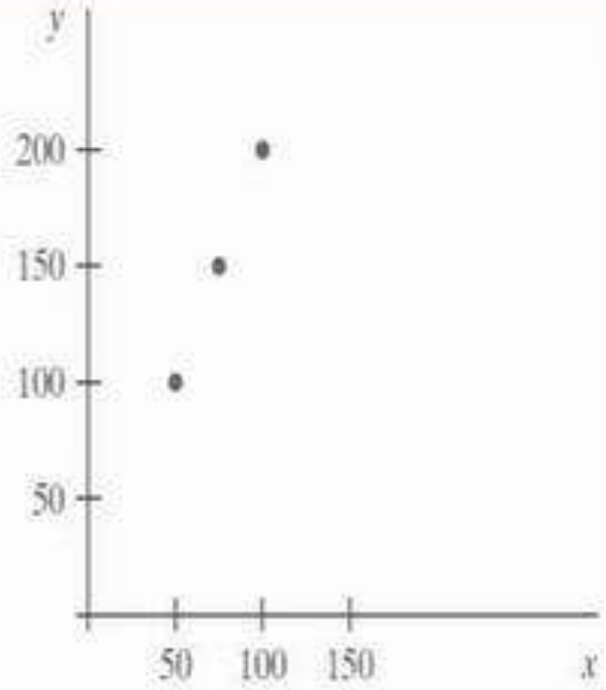

FIGURE 3
Display of three point positions generated with `glBegin (GL_POINTS)`.

OpenGl Point Functions

- Alternatively, we could use the coordinate values for the preceding points in arrays as:

```
int point1 [ ] = {50, 100};
int point2 [ ] = {75, 150};
int point3  [ ] = {100, 200};
```

- and call the OpenGL functions for plotting the three points as

```
glBegin (GL_POINTS);
glVertex2iv (point1);
glVertex2iv (point2);
glVertex2iv (point3);
glEnd ( );
```

OpenGl Line Functions

- Graphic packages normally provide a function for specifying one or more straight-line segments, where each line segment is defined by two endpoint coordinate positions.

- For example, if we have five coordinate positions, labelled **p1** through **p5**, and each position is represented by a two-dimensional array, then the following code could create the display shown in Figure 4(a):

```
glBegin (GL_LINES);
 glVertex2iv (p1);
 glVertex2iv (p2);
 glVertex2iv (p3);
 glVertex2iv (p4);
 glVertex2iv (p5);
    glEnd ( );
```

OpenGl Line Functions

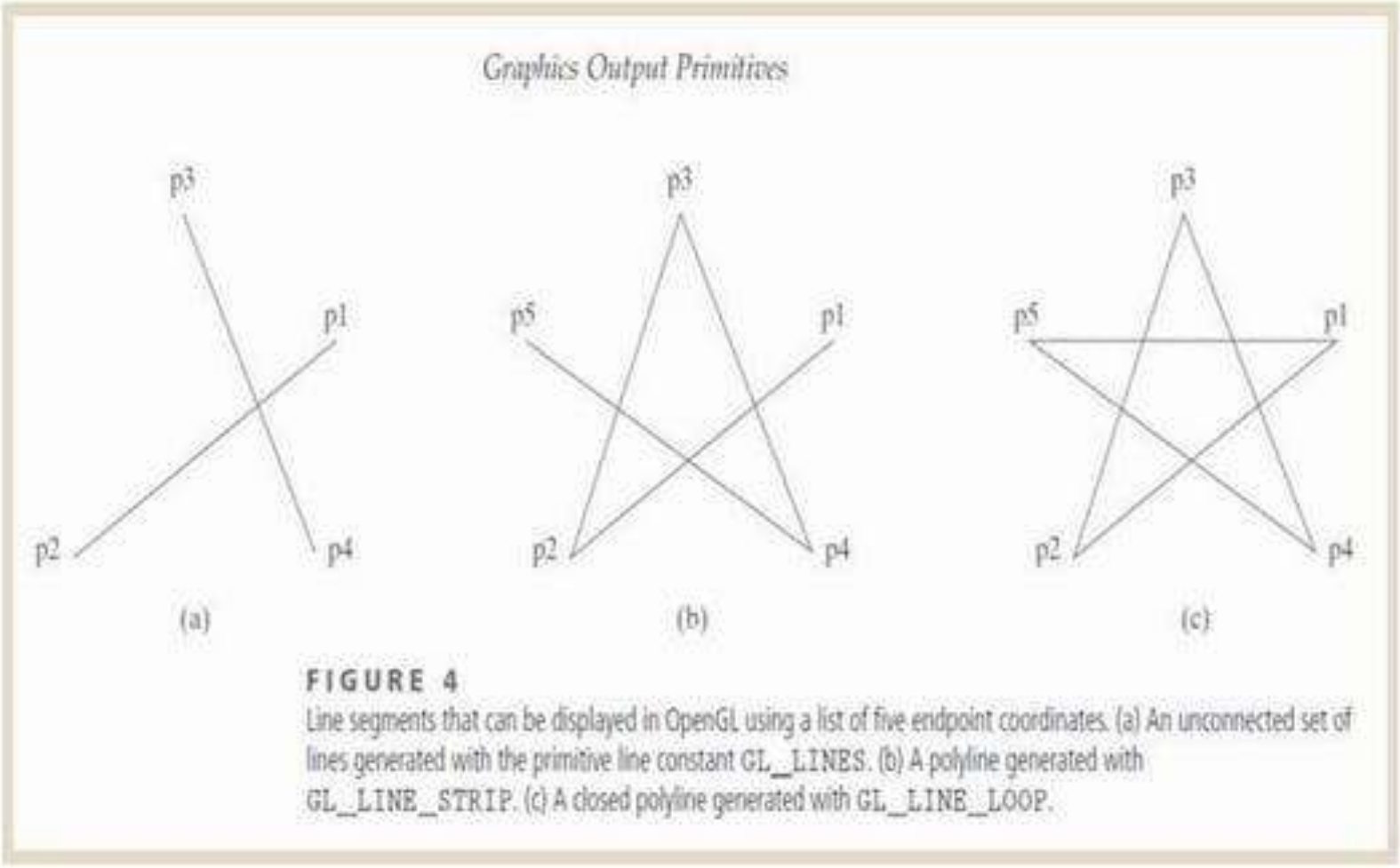

FIGURE 4
Line segments that can be displayed in OpenGL using a list of five endpoint coordinates. (a) An unconnected set of lines generated with the primitive line constant `GL_LINES`. (b) A polyline generated with `GL_LINE_STRIP`. (c) A closed polyline generated with `GL_LINE_LOOP`.

OpenGl Line Functions

- Using the OpenGL primitive constant **GL_LINE_STRIP**, we obtain a **polyline.**
- Using the same five coordinate positions as in the previous example, we obtain the display in Figure 4(b) with the code

```
glBegin (GL_LINE_STRIP);
    glVertex2iv (p1);
    glVertex2iv (p2);
    glVertex2iv (p3);
    glVertex2iv (p4);
    glVertex2iv (p5);
    glEnd ( );
```

OpenGl Line Functions

- The third OpenGL line primitive is **GL_LINE_LOOP**, which produces a **closed polyline.**

- Figure 4(c) shows the display of our endpoint list when we select this line option, using the code

```
glBegin (GL_LINE_LOOP);
   glVertex2iv (p1);
   glVertex2iv (p2);
   glVertex2iv (p3);
   glVertex2iv (p4);
   glVertex2iv (p5);
      glEnd ( );
```

OpenGl Curve Functions

- Routines for generating basic curves, such as circles and ellipses, are not included as primitive functions in the OpenGL core library.
- But this library does contain functions for displaying B´ezier splines, which are polynomials that are defined with a discrete point set.
- And the OpenGL Utility (GLU) library has routines for three-dimensional quadrics, such as spheres and cylinders, as well as routines for producing rational B-splines, which are a general class of splines that include the simpler B´ezier curves.

OpenGl Curve Functions

- Using rational B-splines, we can display circles, ellipses, and other two-dimensional quadrics.
- In addition, there are routines in the OpenGL Utility Toolkit (GLUT) that we can use to display some three-dimensional quadrics, such as spheres and cones, and some other shapes.
- However, all these routines are more involved than the basic primitives we introduce in this chapter.

OpenGl Curve Functions

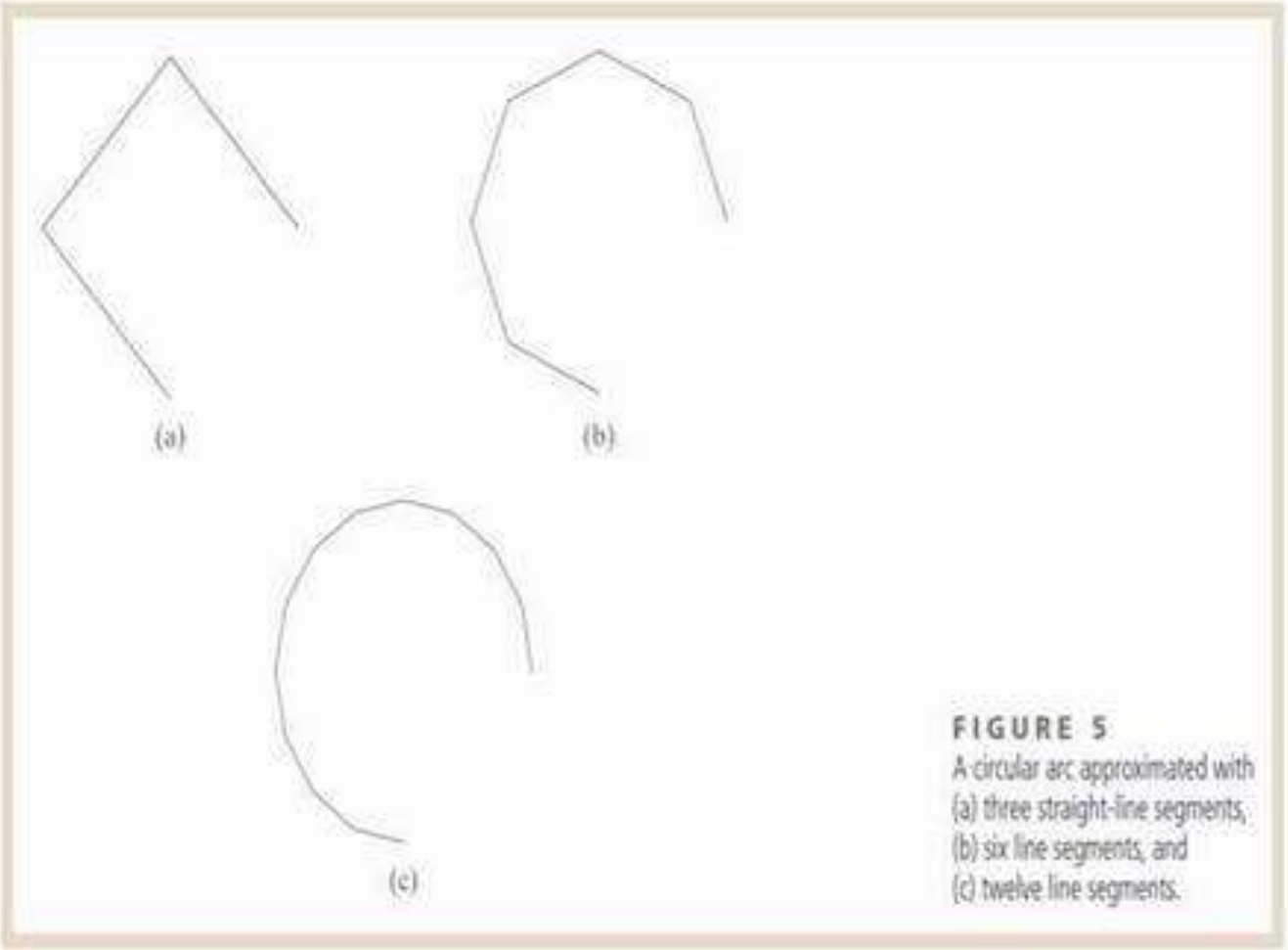

FIGURE 5
A circular arc approximated with (a) three straight-line segments, (b) six line segments, and (c) twelve line segments.

Fill-Area Primitives

- Another useful construct, besides points, straight-line segments, and curves, for describing components of a picture is an area that is filled with some solid color or pattern.

- A picture component of this type is typically referred to as a **fill area** or a **filled area.**

Fill-Area Primitives

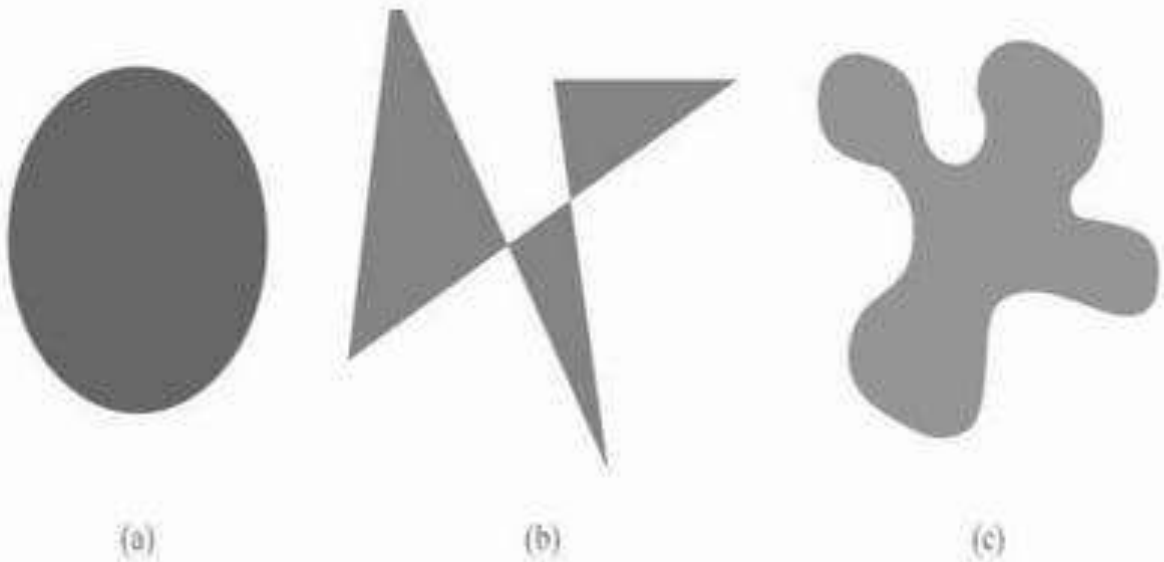

FIGURE 6
Solid-color fill areas specified with various boundaries. (a) A circular fill region. (b) A fill area bounded by a closed polyline. (c) A filled area specified with an irregular curved boundary.

Fill-Area Primitives

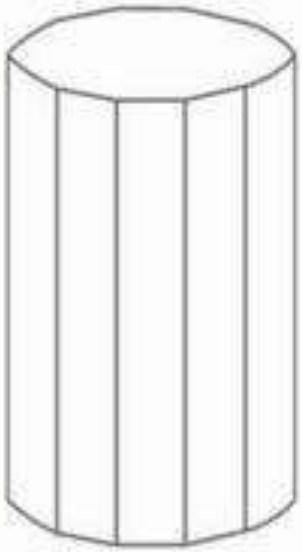

FIGURE 7
Wire-frame representation for a cylinder, showing only the front (visible) faces of the polygon mesh used to approximate the surfaces.

Fill-Area Primitives

- Objects described with a set of polygon surface patches are usually referred to as **standard graphics objects,** or just **graphics objects.**

Polygon Fill Areas

- Mathematically defined, a **polygon** is a plane figure specified by a set of three or more coordinate positions, called *vertices,* that are connected in sequence by straight-line segments, called the *edges* or *sides* of the polygon.
- Examples of polygons include triangles, rectangles, octagons, and decagons.

Polygon Fill Areas

- Sometimes, any plane figure with a closed-polyline boundary is alluded to as a polygon, and one with no crossing edges is referred to as a *standard polygon* or a *simple polygon.*

- In an effort to avoid ambiguous object references, we will use the term *polygon* to refer only to those planar shapes that have a closed-polyline boundary and no edge crossings.

OpenGL Polygon Fill-Area Functions

- With one exception, the OpenGL procedures for specifying fill polygons are similar to those for describing a point or a polyline.
- A **glVertex** function is used to input the coordinates for a single polygon vertex, and a complete polygon is described with a list of vertices placed between a **glBegin/glEnd** pair.
- However, there is one additional function that we can use for displaying a rectangle that has an entirely different format.

OpenGL Polygon Fill-Area Functions

- By default, a polygon interior is displayed in a solid color, determined by the current color settings.
- As options, we can fill a polygon with a pattern and we can display polygon edges as line borders around the interior fill.
- There are six different symbolic constants that we can use as the argument in the **glBegin** function to describe polygon fill areas.
- These six primitive constants allow us to display a single fill polygon, a set of unconnected fill polygons, or a set of connected fill polygons.

OpenGL Polygon Fill-Area Functions

- In OpenGL, a fill area must be specified as a convex polygon.
- Thus, a vertex list for a fill polygon must contain at least three vertices, there can be no crossing edges, and all interior angles for the polygon must be less than 180◦.

OpenGL Polygon Fill-Area Functions

- Because graphics displays often include rectangular fill areas, OpenGL provides a special rectangle function that directly accepts vertex specifications in the *xy* plane.
- In some implementations of OpenGL, the following routine can be more efficient than generating a fill rectangle using **glVertex** specifications:

```
glRect* (x1, y1, x2, y2);
```

- One corner of this rectangle is at coordinate position (*x*1, *y*1), and the opposite corner of the rectangle is at position (*x*2, *y*2).

- Suffix codes for **glRect** specify the coordinate data type and whether coordinates are to be expressed as array elements.
- These codes are **i** (for integer), **s** (for short), **f** (for float), **d** (for double), and **v** (for vector). The rectangle is displayed with edges parallel to the *xy* coordinate axes.

OpenGL Polygon Fill-Area Functions

- As an example, the following statement defines the square shown in Figure 21:

```
glRecti (200, 100, 50, 250);
```

- If we put the coordinate values for this rectangle into arrays, we can generate the same square with the following code:

```
int vertex1 [ ] = {200, 100};
int vertex2 [ ] = {50, 250};
glRectiv (vertex1, vertex2);
```

OpenGL Polygon Fill-Area Functions

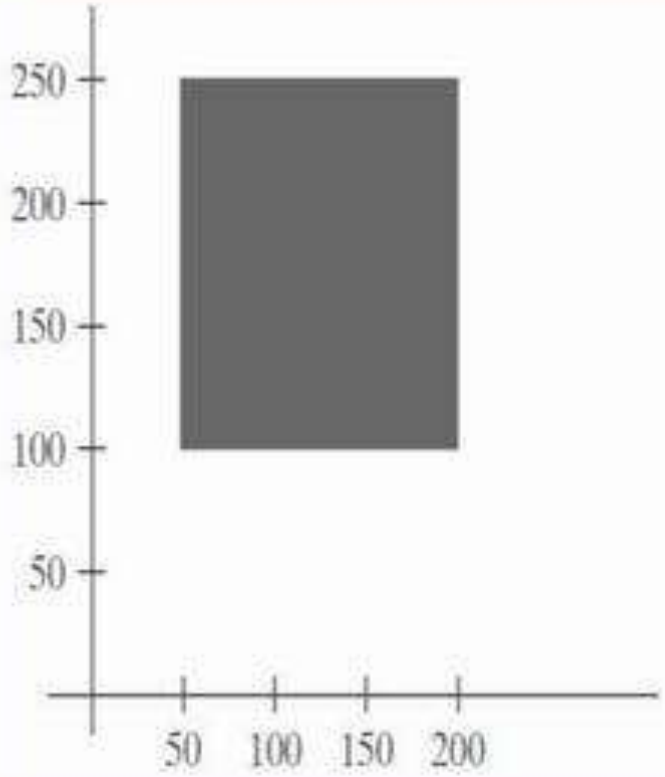

FIGURE 21
The display of a square fill area using the `glRect` function.

OpenGL Polygon Fill-Area Functions

- Each of the other six OpenGL polygon fill primitives is specified with a symbolic constant in the **glBegin** function, along with a a list of **glVertex** commands.

- With the OpenGL primitive constant **GL_POLYGON**, we can display a single polygon fill area such as that shown in Figure 22(a).

OpenGL Polygon Fill-Area Functions

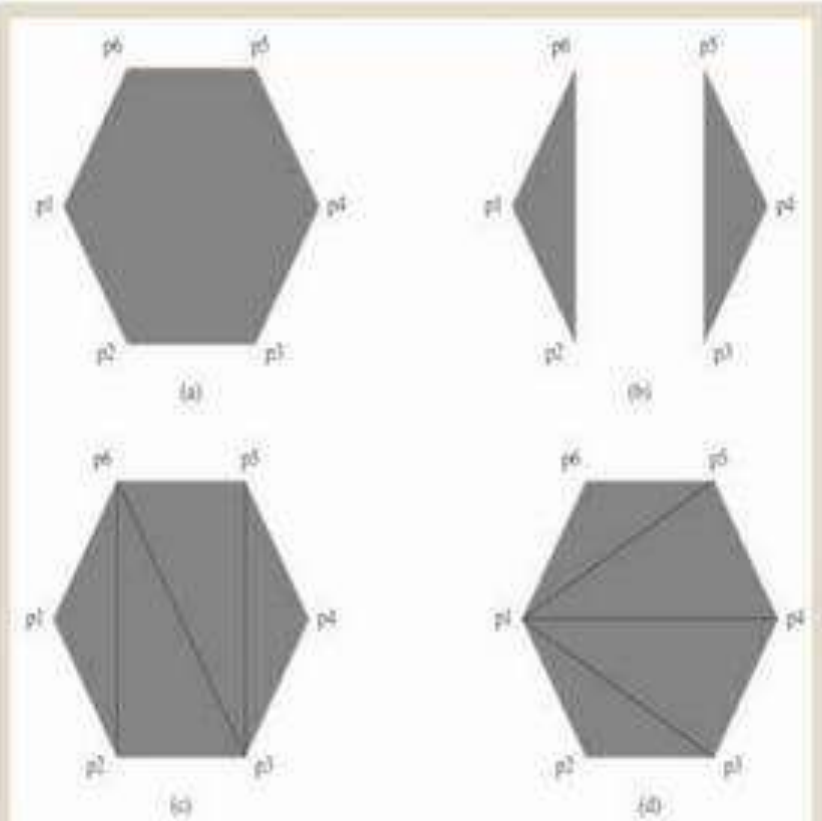

FIGURE 22
Displaying polygon fill areas using a list of six vertex positions. (a) A single convex polygon fill area generated with the primitive constant GL_POLYGON. (b) Two unconnected triangles generated with GL_TRIANGLES. (c) Four connected triangles generated with GL_TRIANGLE_STRIP. (d) Four connected triangles generated with GL_TRIANGLE_FAN.

OpenGL Polygon Fill-Area Functions

- For this example, we assume that we have a list of six points, labeled p1 through p6, specifying two-dimensional polygon vertex positions in a counterclockwise ordering.
- Each of the points is represented as an array of (x, y) coordinate values:

```
glBegin (GL_POLYGON);
   glVertex2iv (p1);
   glVertex2iv (p2);
   glVertex2iv (p3);
   glVertex2iv (p4);
   glVertex2iv (p5);
   glVertex2iv (p6);
      glEnd ( );
```

OpenGL Polygon Fill-Area Functions

- If we reorder the vertex list and change the primitive constant in the previous code example to **GL_TRIANGLES**, we obtain the two separated triangle fill areas in Figure 22(b):

```
glBegin (GL_TRIANGLES);
   glVertex2iv (p1);
   glVertex2iv (p2);
   glVertex2iv (p6);
   glVertex2iv (p3);
   glVertex2iv (p4);
   glVertex2iv (p5);
      glEnd ( );
```

OpenGL Polygon Fill-Area Functions

- By reordering the vertex list once more and changing the primitive constant to **GL_TRIANGLE_STRIP**, we can display the set of connected triangles shown in Figure 22(c):

```
glBegin (GL_TRIANGLE_STRIP);
        glVertex2iv (p1);
        glVertex2iv (p2);
        glVertex2iv (p6);
        glVertex2iv (p3);
        glVertex2iv (p5);
        glVertex2iv (p4);
           glEnd ( );
```

OpenGL Polygon Fill-Area Functions

- Another way to generate a set of connected triangles is to use the "fan" approach illustrated in Figure 22(d), where all triangles share a common vertex.
- We obtain this arrangement of triangles using the primitive constant **GL_TRIANGLE_FAN** and the original ordering of our six vertices:

```
glBegin (GL_TRIANGLE_FAN);
    glVertex2iv (p1);
    glVertex2iv (p2);
    glVertex2iv (p3);
    glVertex2iv (p4);
    glVertex2iv (p5);
    glVertex2iv (p6);
    glEnd ( );
```

OpenGL Polygon Fill-Area Functions

- Besides the primitive functions for triangles and a general polygon, OpenGL provides for the specifications of two types of quadrilaterals (four-sided polygons).

OpenGL Polygon Fill-Area Functions

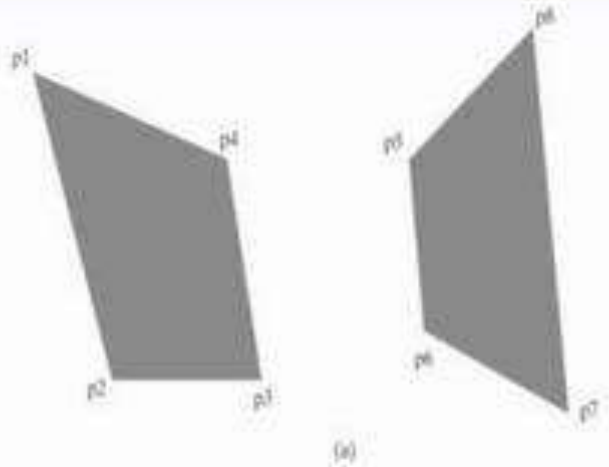

FIGURE 2.3
Displaying quadrilateral fill areas using a list of eight vertex positions. (a) Two unconnected quadrilaterals generated with GL_QUADS. (b) Three connected quadrilaterals generated with GL_QUAD_STRIP.

OpenGL Polygon Fill-Area Functions

- With the **GL_QUADS** primitive constant and the following list of eight vertices, specified as two-dimensional coordinate arrays, we can generate the display shown in Figure 23(a):

```
glBegin (GL_QUADS);
  glVertex2iv (p1);
  glVertex2iv (p2);
  glVertex2iv (p3);
  glVertex2iv (p4);
  glVertex2iv (p5);
  glVertex2iv (p6);
  glVertex2iv (p7);
  glVertex2iv (p8);
     glEnd ( );
```

OpenGL Polygon Fill-Area Functions

- Rearranging the vertex list in the previous quadrilateral code example and changing the primitive constant to **GL_QUAD_STRIP**, we can obtain the set of connected quadrilaterals shown in Figure 23(b):

```
glBegin (GL_QUAD_STRIP);
    glVertex2iv (p1);
    glVertex2iv (p2);
    glVertex2iv (p4);
    glVertex2iv (p3);
    glVertex2iv (p5);
    glVertex2iv (p6);
    glVertex2iv (p8);
    glVertex2iv (p7);
        glEnd ( );
```