Computer Graphics

Chapter 5

Attributes of Graphics Primitives

Chapter 5.
Attributes of Graphics Primitives

Part I.

Color, point and line attributes

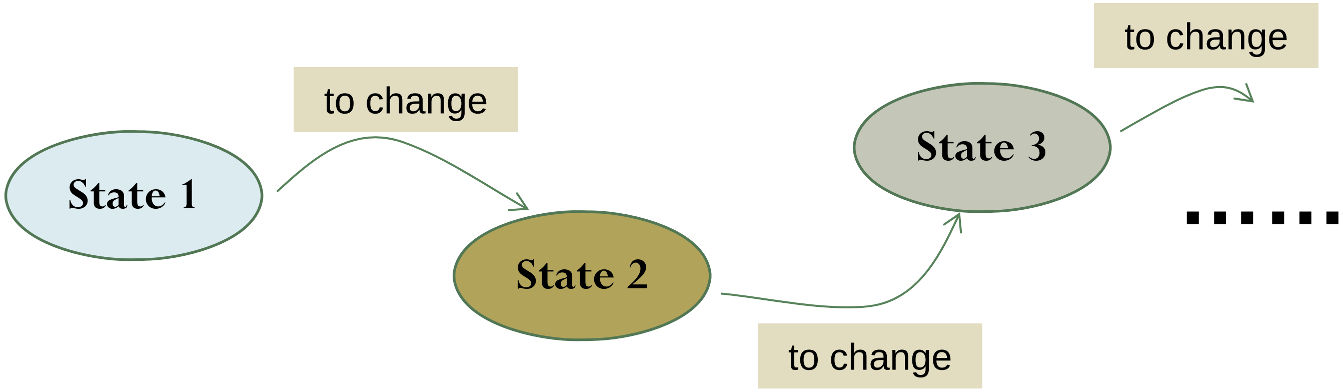
OpenGL functions

Outline

- OpenGL State Machine and Variables
- Color and Gray Scale
- OpenGL Color Functions
- OpenGL Point-Attribute Functions
- OpenGL Line-Attribute Functions

OpenGL State Machine

- State system (or state machine)
  - A graphics system that maintains a list for the current values of attributes and other parameters (state variables or state parameters).
  - We assign a value to one or more state parameters, that is, we put the system into a certain state.
  - The state will be persistent until we change the value of a state parameter.



OpenGL State Machine

- OpenGL is a finite state machine
  - A predetermined and countable number of different states
  - The graphics system behaviors are determined by these **system state**, which can be modified by calling OpenGL **functions**.

- The OpenGL state includes:
  - The current color or other attributes
  - The current model & viewing transformations
  - The current camera model & clipping
  - The current lighting & reflectance model
  - The current viewport

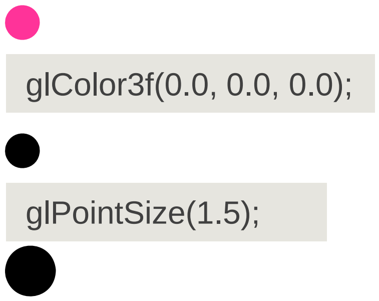
```
…
glMatrixMode (GL_PROJECTION);
glLoadIdentity( );
gluOrtho2D( … );
…
```

- All have default values, remaining until a new setting on it.

OpenGL State Machine

A common misconception

- The primitive drawing functions are state changes ✗
- They are the output functions telling the system to draw something to the screen with the certain specified current state. ✓
  - The options of the current state
    - the current color
    - the current point size
    - the depth function - enabled or not
    - …



```
glColor3f(0.0, 0.0, 0.0);
```

```
glPointSize(1.5);
```

OpenGL State Machine

- What do you do to maintain states and state variables by OpenGL?
  - To set states for drawing geometric primitives

    E.g: glPointSize (size);
    glLineStipple (repeat, pattern);
    glShadeModel (GL_ SMOOTH);

  - or change (enable) states of how OpenGL draws them
    - By default, most of these states are initially **inactive**.
    - Such states include: objects rendered with lighting, texturing, or undergo different processes, such as hidden surface removal, fog, and other states, which affect the appearance.
    - To turn on/off states

      glEnable (GL_ LIGHTING);
      glDisable (GL_BLEND);

Basic State Management

void **glEnable (GLenum** *capability*);
void **glDisable (GLenum** *capability*);

- More than 60 enumerated values can be passed as parameters to them.
- E.g: **GL_BLEND**
  (controls blending of RGBA values)
  GL_DEPTH_TEST
  (controls depth comparisons and updates to the depth buffer)
  GL_FOG
  (controls fog)
  GL_LINE_STIPPLE
  (patterned lines)
  GL_LIGHTING
  (light effect)

Basic State Management

- To check whether a state is currently enabled or disabled by

  GLboolean **glIsEnabled (GLenum** *capability*)

  - Returns GL_TRUE or GL_FALSE, depending on whether or not it is currently activated.

- For more complicated state variables, such as

  glColor3f ( ) set three values, which are part of the GL_CURRENT_COLOR state.

  - Query routines: **glGet* ( )**;

Basic State Management

- Five querying routines

```
void glGetBooleanv (GLenum pname, GLboolean *params);
void glGetIntegerv (GLenum pname, GLint *params);
void glGetFloatv (GLenum pname, GLfloat *params);
void glGetDoublev (GLenum pname, GLdouble *params);
void glGetPointerv (GLenum pname, GLvoid **params);
```

  - *pname*: a symbolic constant indicating the state variable to return;
    E.g.: GL_CURRENT_COLOR, GL_CURRENT_NORMAL
  - *params*: a pointer to an array of the returned value.

Basic State Management

Example

- To get the current RGBA color:

  glGetIntegerv (GL_CURRENT_COLOR, *params*) or

  glGetFloatv (GL_CURRENT_COLOR, *params*)

- To get how many bits per pixel are available for each individual color component:

  glGetIntegerv (GL_RED_BITS, redBitSize)

- The possible values for *pname* can be referred to the tables in "OpenGL State Variables“ in “The Red Book”.

  http://fly.srk.fer.hr/~unreal/theredbook/appendixb.html (Release one)

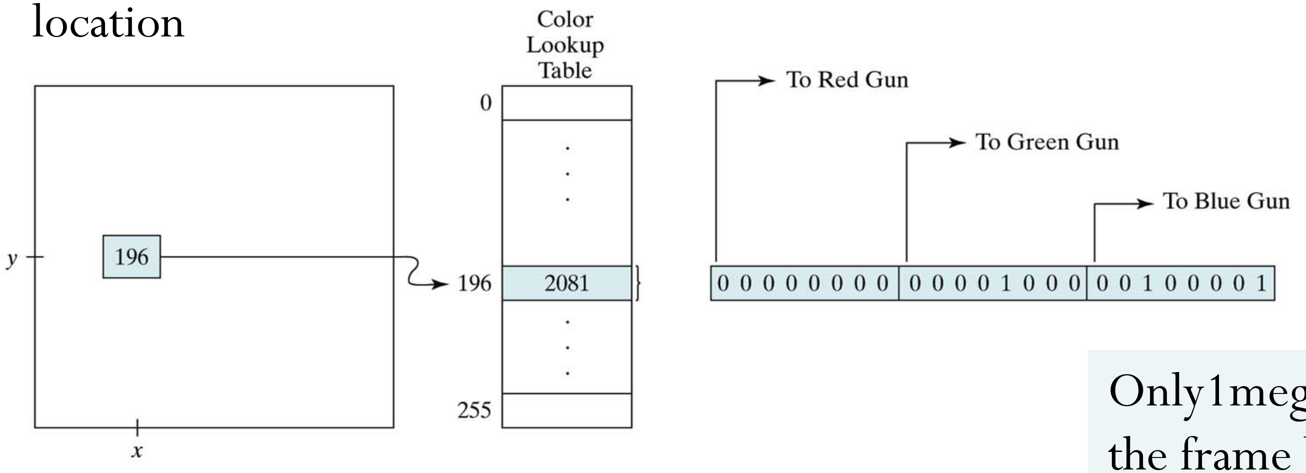
Color and Gray Scale

- The basic attribute for all primitives is **color**.
- In a color raster system, the number of colors available depends on the **amount of storage per pixel** in the frame buffer.
- Two ways to store the color information
  - RGB color code
    - directly stored in the frame buffer
    - E.g.: resolution - 1024X1024, a full-color (24-bit per pixel) color system

      3 megabytes of storage for the frame buffer
  - Color index value (next slide)
    - it references a color lookup table

Color and Gray Scale

- Color index value (cont.)
  - Color code into a table (color lookup table or color map)
  - To keep the index value referencing the color-table entries into each pixel location



Only1megabyte for the frame buffer (size: 1024X1024)

Figure 5-1

A color lookup table with 24 bits per entry that is accessed from a frame buffer with 8 bits per pixel. A value of 196 stored at pixel position (x, y) references the location in this table containing the hexadecimal value 0x0821 (a decimal value of 2081). Each 8-bit segment of this entry controls the intensity level of one of the three electron guns in an RGB monitor.

Computer Graphics with Open GL, Third Edition, by Donald Hearn and M.Pauline Baker.
ISBN 0-13-0-15390-7 © 2004 Pearson Education, Inc., Upper Saddle River, NJ. All rights reserved.

OpenGL&GLUT Color Functions

- Color display mode
  - RGB (RGBA) Mode: GLUT_RGB, GLUT_RGBA; GL_RGB, …
    - "A" is the alpha value for color blending.
  - Color-Index Mode: GLUT_INDEX; GL_COLOR_INDEX
- Set up for using the function

```
glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
```

RGB (RGBA) mode

```
glColor* (colorComponents);

E.g.: glColor3f (0.0, 1.0, 1.0);
      glColor3fv (colorArray);
      glColor3i (0, 255, 255);
```

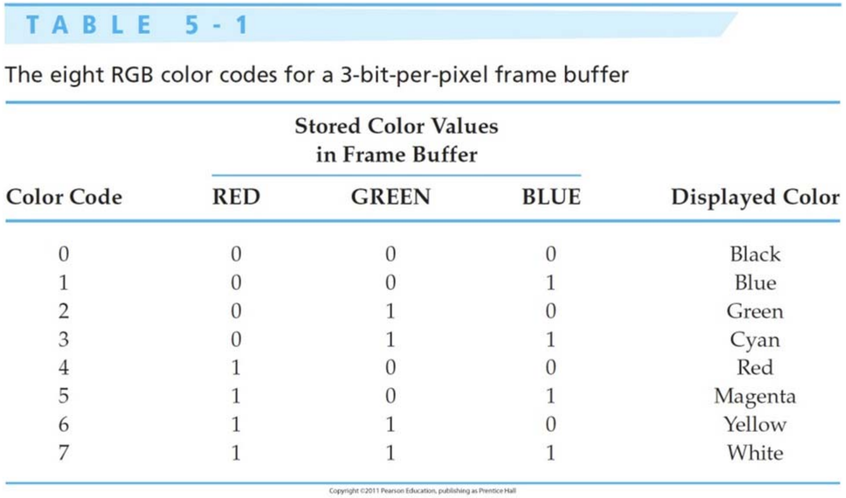


TABLE 5-1

The eight RGB color codes for a 3-bit-per-pixel frame buffer

	Stored Color Values in Frame Buffer			
Color Code	RED	GREEN	BLUE	Displayed Color
0	0	0	0	Black
1	0	0	1	Blue
2	0	1	0	Green
3	0	1	1	Cyan
4	1	0	0	Red
5	1	0	1	Magenta
6	1	1	0	Yellow
7	1	1	1	White

OpenGL Color Functions

- Color display mode (cont.)

Color-index mode

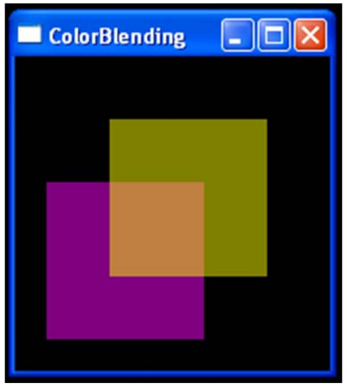
glIndex* (colorIndex);
*: ub, s, i, d, or f
colorIndex is a non-negative integer value.

the number of index in a color table is always a power of 2, such as 256 or 1024.

E.g.:

```
//set the current color by the index in a color table
glIndexi (196);
//to establish the color table
glutSetColor (index, red, green, blue);
```

OpenGL Color Blending



- Methods for producing color-mixing effects

  (only performed in **RGB** or **RGBA** mode)

- Blending effects are generated with the blending factors for destination object (the current object in the frame buffer) and source object (the incoming object).

- The new blended color is calculated as

  $(\text{factor})_{\text{Source}} * \mathbf{(R, G, B, A)_{Source}} + (\text{factor})_{\text{Des.}} * \mathbf{(R, G, B, A)_{Des.}}$

$$\underline{(Sr*Rs + Dr*Rd, Sg*Gs + Dg*Gd, Sb*Bs + Db*Bd, Sa*As + Da*Ad)} \quad (5\text{-}1)$$

(Rs, Gs, Bs, As) – Source Color

(Rd, Gd, Bd, Ad) – Destination Color

(Sr, Sg, Sb, Sa) – Source blending factors

(Dr, Dg, Db, Da) – Destination blending factors

OpenGL Color Blending

- How to set up color blending in OpenGL
  - Firstly, to activate this feature

```
glEnable (GL_BLEND);
glDisable (GL_BLEND);
```

  - Then, use the function

```
glBlendFunc (sFactor, dFactor);
```

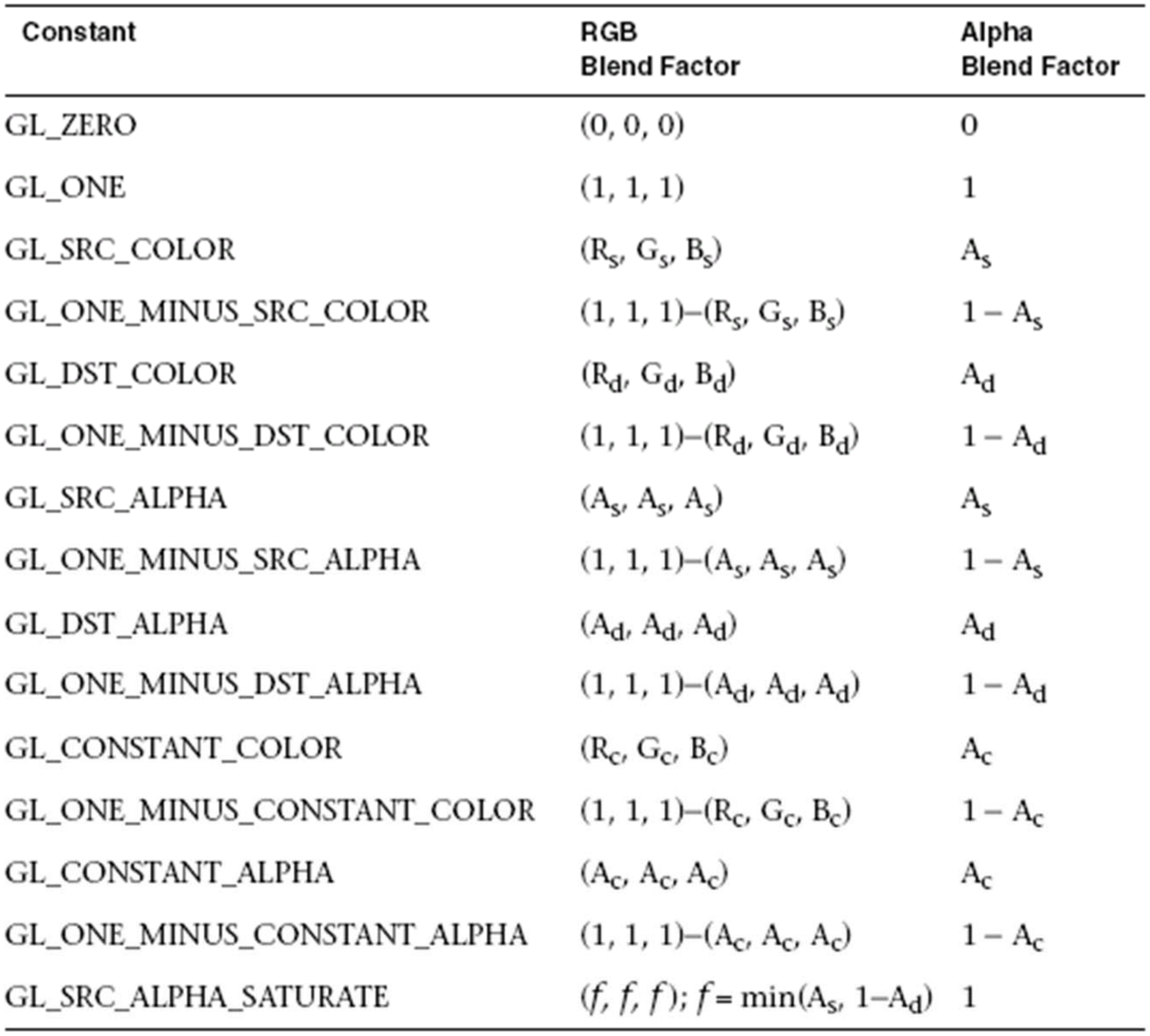
sFactor (default: GL_ONE) [default: "replacing"]

dFactor (default: GL_ZERO) :

GL_ZERO -- (0.0, 0.0, 0.0, 0.0),

GL_ONE -- (1.0, 1.0, 1.0, 1.0)

OpenGL Color Blending



Constant	RGB Blend Factor	Alpha Blend Factor
GL_ZERO	(0, 0, 0)	0
GL_ONE	(1, 1, 1)	1
GL_SRC_COLOR	(R_s, G_s, B_s)	A_s
GL_ONE_MINUS_SRC_COLOR	$(1, 1, 1)-(R_s, G_s, B_s)$	$1 - A_s$
GL_DST_COLOR	(R_d, G_d, B_d)	A_d
GL_ONE_MINUS_DST_COLOR	$(1, 1, 1)-(R_d, G_d, B_d)$	$1 - A_d$
GL_SRC_ALPHA	(A_s, A_s, A_s)	A_s
GL_ONE_MINUS_SRC_ALPHA	$(1, 1, 1)-(A_s, A_s, A_s)$	$1 - A_s$
GL_DST_ALPHA	(A_d, A_d, A_d)	A_d
GL_ONE_MINUS_DST_ALPHA	$(1, 1, 1)-(A_d, A_d, A_d)$	$1 - A_d$
GL_CONSTANT_COLOR	(R_c, G_c, B_c)	A_c
GL_ONE_MINUS_CONSTANT_COLOR	$(1, 1, 1)-(R_c, G_c, B_c)$	$1 - A_c$
GL_CONSTANT_ALPHA	(A_c, A_c, A_c)	A_c
GL_ONE_MINUS_CONSTANT_ALPHA	$(1, 1, 1)-(A_c, A_c, A_c)$	$1 - A_c$
GL_SRC_ALPHA_SATURATE	$(f, f, f); f = \min(A_s, 1-A_d)$	1

Table 6-1 Source and Destination Blending Factors

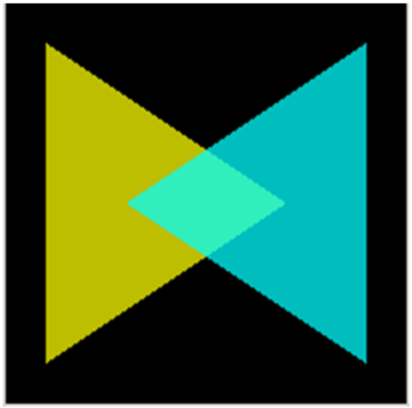
(From: OpenGL programming guide, 7th Ed.)

OpenGL Color Blending

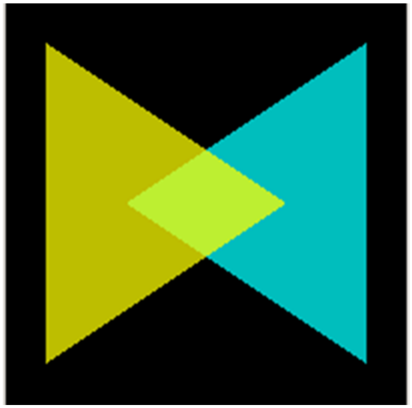
- Example: the drawn order effects the blending result
  - Alpha: 0.75;
  - Source and destination blending factors:

    GL_SRC_ ALPHA and GL_ONE_MINUS_SRC_ALPHA

    0.75*(source color) + (1.0-0.75)*(des. color)



yellow one;
cyan one.



cyan one;
yellow one.

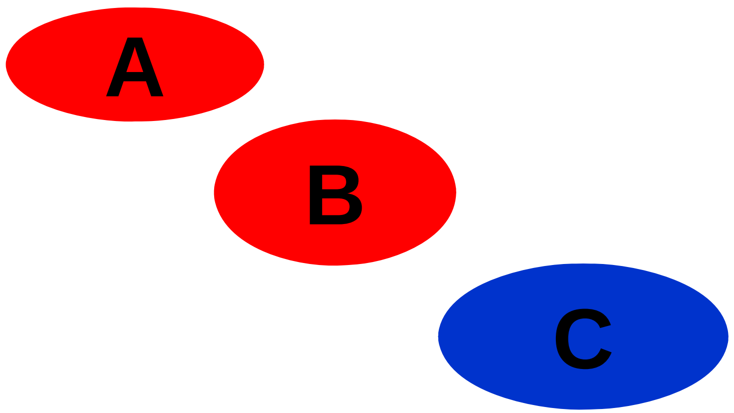
(From: OpenGL programming guide, 7th Ed.)

Specifying Color in OpenGL

- Don’t forget the OpenGL is a state machine!
- To define the shape of an object is independent of specifying its color.
    - Color is a state.

For example, the pseudocode

```
set_current_color(red);
        draw_object(A);
        draw_object(B);
set_current_color(green);
set_current_color(blue);
        draw_object(C);
```



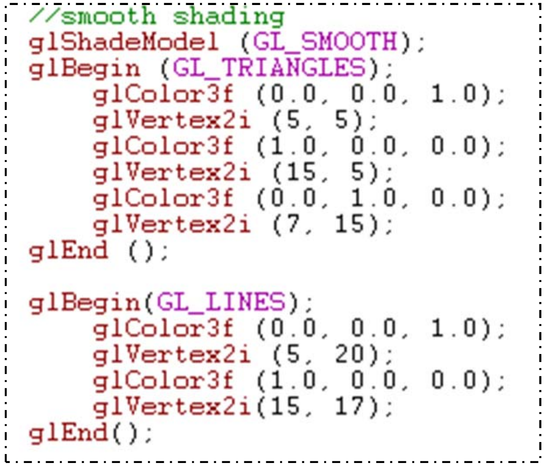
Specifying Shading Model

- A line or a filled polygon primitive can be drawn with a single color (**flat shading**) or with many different colors (**smooth shading**, also called Gouraud shading).

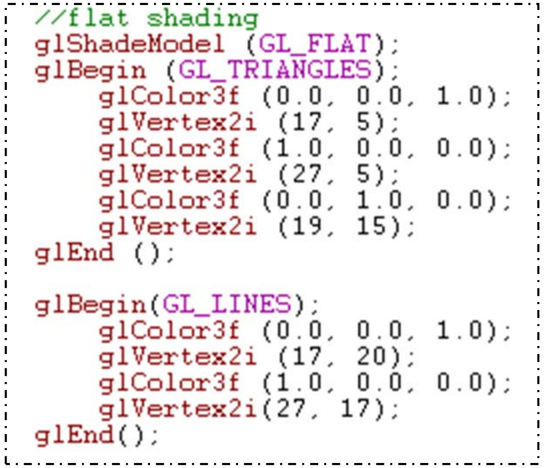
void glShadeModel (GLenum *mode*);

mode: GL_SMOOTH (default)

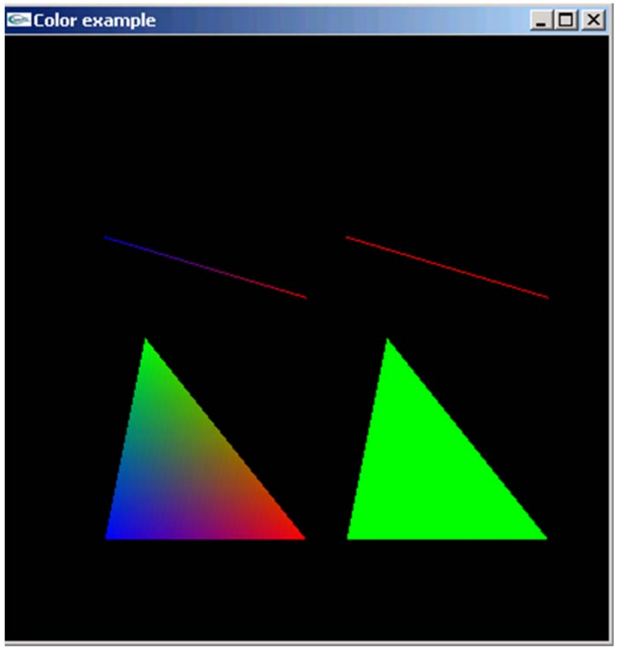
GL_FLAT



```
//smooth shading
glShadeModel (GL_SMOOTH);
glBegin (GL_TRIANGLES);
     glColor3f (0.0, 0.0, 1.0);
     glVertex2i (5, 5);
     glColor3f (1.0, 0.0, 0.0);
     glVertex2i (15, 5);
     glColor3f (0.0, 1.0, 0.0);
     glVertex2i (7, 15);
glEnd ();

glBegin(GL_LINES);
     glColor3f (0.0, 0.0, 1.0);
     glVertex2i (5, 20);
     glColor3f (1.0, 0.0, 0.0);
     glVertex2i(15, 17);
glEnd();
```



```
//flat shading
glShadeModel (GL_FLAT);
glBegin (GL_TRIANGLES);
     glColor3f (0.0, 0.0, 1.0);
     glVertex2i (17, 5);
     glColor3f (1.0, 0.0, 0.0);
     glVertex2i (27, 5);
     glColor3f (0.0, 1.0, 0.0);
     glVertex2i (19, 15);
glEnd ();

glBegin(GL_LINES);
     glColor3f (0.0, 0.0, 1.0);
     glVertex2i (17, 20);
     glColor3f (1.0, 0.0, 0.0);
     glVertex2i(27, 17);
glEnd();
```



Specifying Color in OpenGL Summary

- RGBA color components
  - glClearColor (r, g, b, a); glClear (GL_COLOR_BUFFER_BIT);
  - glColor* (r, g, b);
  - glutInitDisplayMode(GLUT_RGBA | …); //GLUT_RGB
- Color index mode
  - glClearIndex (index);); glClear (GL_COLOR_BUFFER_BIT);
  - glIndex* (colorIndex);
  - glutInitDisplayMode(GLUT_INDEX | …);
- Color blending – only for RGB/RGBA mode
  - glEnable (GL_BLEND);
  - glDisable (GL_BLEND);
  - glBlendFunc (sFactor, dFactor);

OpenGL Point-Attribute Functions

- Two basic attributes for points: Color and Size

```
void glPointSize (GLfloat size);
```

size must be greater than 0.0 and by default is 1.0.

- Example:

```
glColor3f (1.0, 0.0, 0.0);
    glBegin (GL_POINTS);
            glVertex2i (50, 100);
            glPointSize (2.0);
            glColor3f (0.0, 1.0, 0.0);
            glVertex2i (75, 150);
            glPointSize (3.0);
            glColor3f (0.0, 0.0, 1.0);
            glVertex2i (100, 200);
    glEnd ();
```

Query the point size by **glGetFloatv(GL_POINT_SIZE)**.

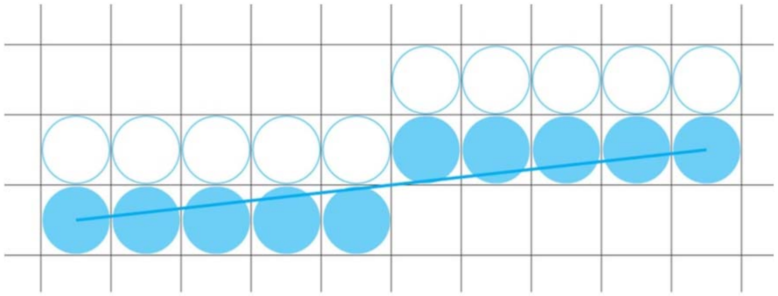
The color and size are determined by current values.

OpenGL Line-Attribute Functions

- Three basic attributes for lines: Color, Width, and Style
- OpenGL line-width function

  void glLineWidth (GLfloat width);

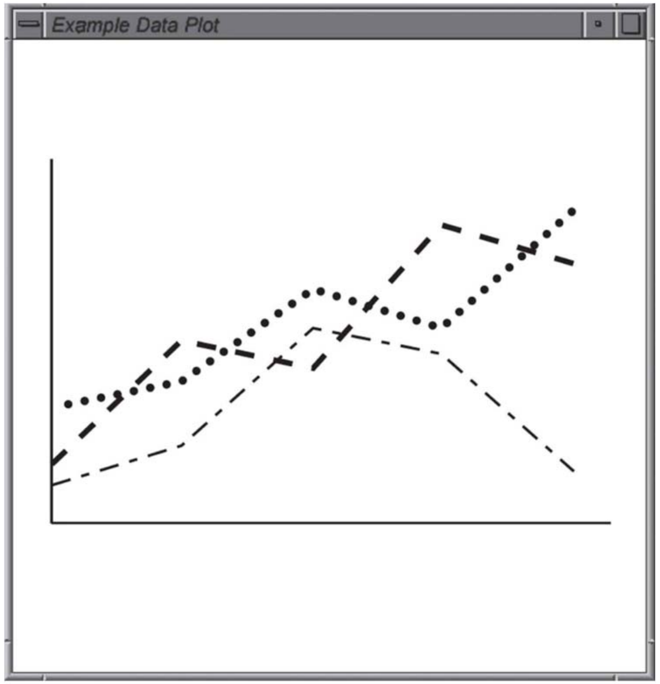
  width must be greater than 0.0 and by default is 1.0.



A double-wide raster line with slope |m| < 1.0 generated with vertical pixel spans.

OpenGL Line-Attribute Functions

- OpenGL Line-Style Function (Stippled lines)



Plotting three data sets with three different OpenGL line styles and line widths: single-width dash-dot pattern, double-width dash pattern, and triple-width dot pattern

OpenGL Line-Attribute Functions

- How to set up OpenGL Line-Style Function
  - Activate the line-style feature by glEnable()

    glEnable (GL_LINE_STIPPLE);

    glDisable (GL_LINE_STIPPLE);
  - Define the current stipple pattern (a pattern of binary digits)

    glLineStipple (repeatFactor, pattern);

    pattern: GLushort

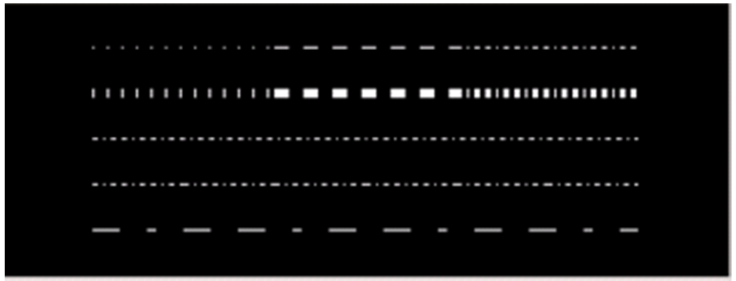
    - The16-bit integer describing how the line should be displayed.
    - A bit "1: an "on" pixel position, and a bit "0":an "off" pixel position.
    - The default pattern is 0xFFFF (each bit has a value of 1) producing a solid line.

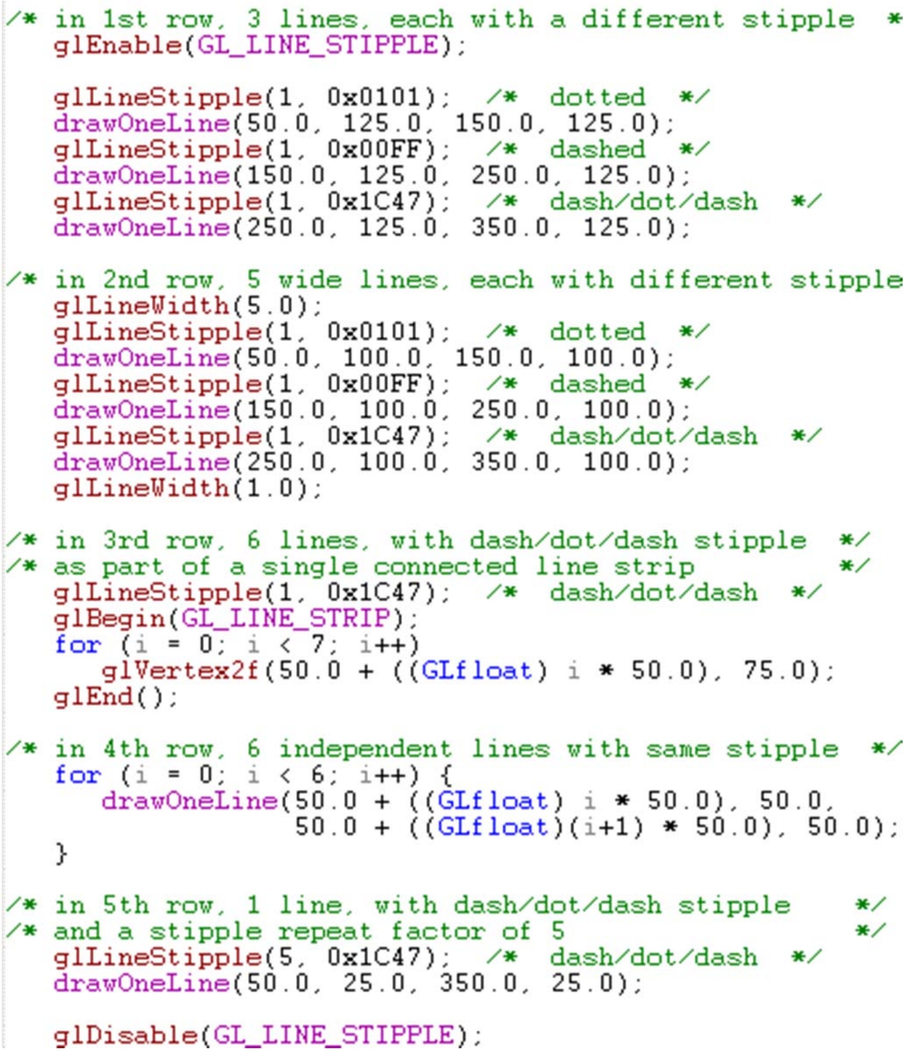
    repeatFactor: GLint

    - Specifies how many times each bit in the pattern is to be repeated.
    - Value is clamped to be between 1 and 256. (default: 1)

OpenGL Line-Attribute Functions

- **Example**





```
/* in 1st row, 3 lines, each with a different stipple  *
   glEnable(GL_LINE_STIPPLE);

   glLineStipple(1, 0x0101);  /*  dotted  */
   drawOneLine(50.0, 125.0, 150.0, 125.0);
   glLineStipple(1, 0x00FF);  /*  dashed  */
   drawOneLine(150.0, 125.0, 250.0, 125.0);
   glLineStipple(1, 0x1C47);  /*  dash/dot/dash  */
   drawOneLine(250.0, 125.0, 350.0, 125.0);

/* in 2nd row, 5 wide lines, each with different stipple
   glLineWidth(5.0);
   glLineStipple(1, 0x0101);  /*  dotted  */
   drawOneLine(50.0, 100.0, 150.0, 100.0);
   glLineStipple(1, 0x00FF);  /*  dashed  */
   drawOneLine(150.0, 100.0, 250.0, 100.0);
   glLineStipple(1, 0x1C47);  /*  dash/dot/dash  */
   drawOneLine(250.0, 100.0, 350.0, 100.0);
   glLineWidth(1.0);

/* in 3rd row, 6 lines, with dash/dot/dash stipple  */
/* as part of a single connected line strip         */
   glLineStipple(1, 0x1C47);  /*  dash/dot/dash  */
   glBegin(GL_LINE_STRIP);
   for (i = 0; i < 7; i++)
      glVertex2f(50.0 + ((GLfloat) i * 50.0), 75.0);
   glEnd();

/* in 4th row, 6 independent lines with same stipple  */
   for (i = 0; i < 6; i++) {
      drawOneLine(50.0 + ((GLfloat) i * 50.0), 50.0,
                  50.0 + ((GLfloat)(i+1) * 50.0), 50.0);
   }

/* in 5th row, 1 line, with dash/dot/dash stipple    */
/* and a stipple repeat factor of 5                  */
   glLineStipple(5, 0x1C47);  /*  dash/dot/dash  */
   drawOneLine(50.0, 25.0, 350.0, 25.0);

   glDisable(GL_LINE_STIPPLE);
```

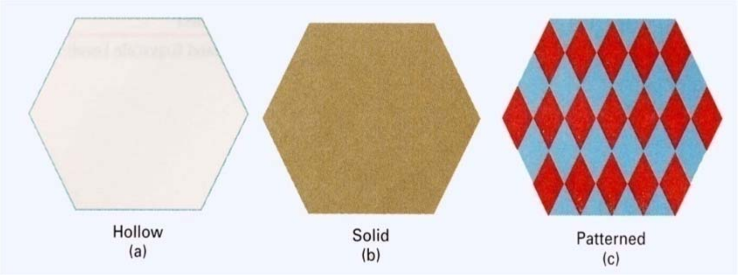
Chapter 5.
Attributes of Graphics Primitives

Part II.

Fill-area methods and attributes

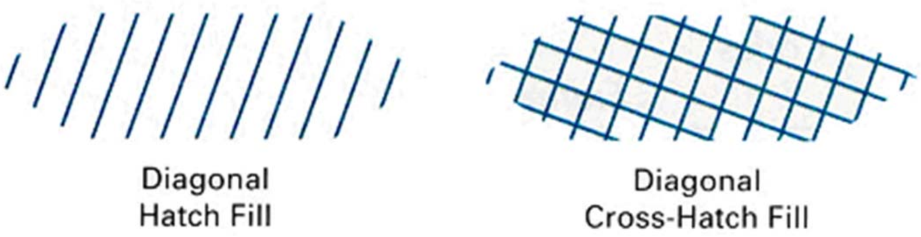
OpenGL functions

Fill-Area Attributes

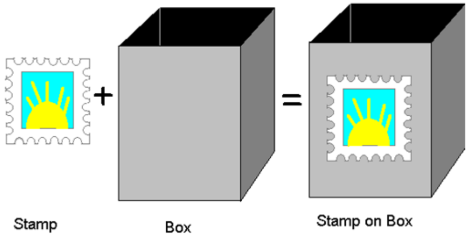


Basic polygon fill styles

- **Fill Styles** (Polygon - displayMode):
  - **Hollow** style
    - Only the boundary in Current Color (GL_LINE)
  - **Solid** fill
    - Filled in Current Color including boundary (GL_FILL) (*default*)
  - **Pattern** fill
    - Fill-Pattern (define a mask filled in Current Color)
    - Hatch fill (e.g., parallel lines)
    - Texture (in Chapter 10)



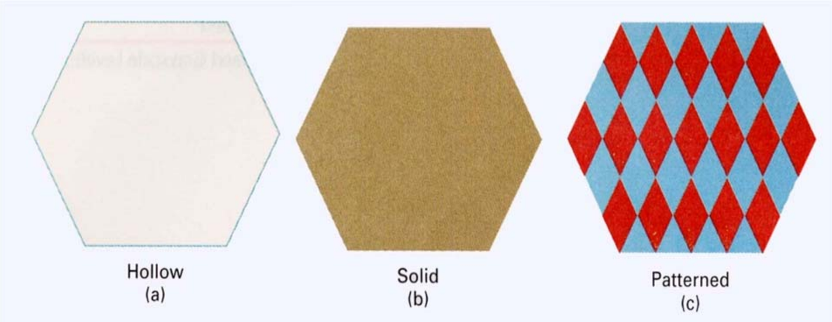
Areas filled with hatch patterns.



(From: MSDN, Microsoft)

OpenGL Fill-Area Attribute Functions

- In OpenGL, fill-area routines are **ONLY** for **convex** polygons
- Polygons
  - Filling within the boundary
    - solidly filled
    - stippled with a certain pattern
  - Outlined polygons
  - Points at the vertices



- Four steps to generate pattern-filled convex polygons:
  - Define a fill pattern;
  - Invoke the polygon-fill routine; (glPolygonStipple())
  - Activate the polygon-fill feature of OpenGL; (glEnable())
  - Describe the polygon to be filled.

OpenGL: Polygon Front and Back

- Two sides of polygon: front and back
  - They can be rendered differently
  - By default, both front and back faces are drawn in the same way

- Control the drawing mode front and back faces

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

  face: GL_FRONT_AND_BACK, GL_FRONT, or GL_BACK;

  mode: GL_POINT (drawn as points), GL_LINE (outlined), GL_FILL (filled, default)

Reversing and Culling Polygon Faces

- Explicitly define the orientation of the front-facing polygon

  void glFrontFace (GLenum *mode*);

  mode: GL_CCW (in counterclockwise order)
  GL_CW (in clockwise order)

- To show the visible surfaces and discard the invisible ones

  void glCullFace (GLenum *mode*);

  mode: GL_FRONT, GL_BACK, or GL_FRONT_AND_BACK

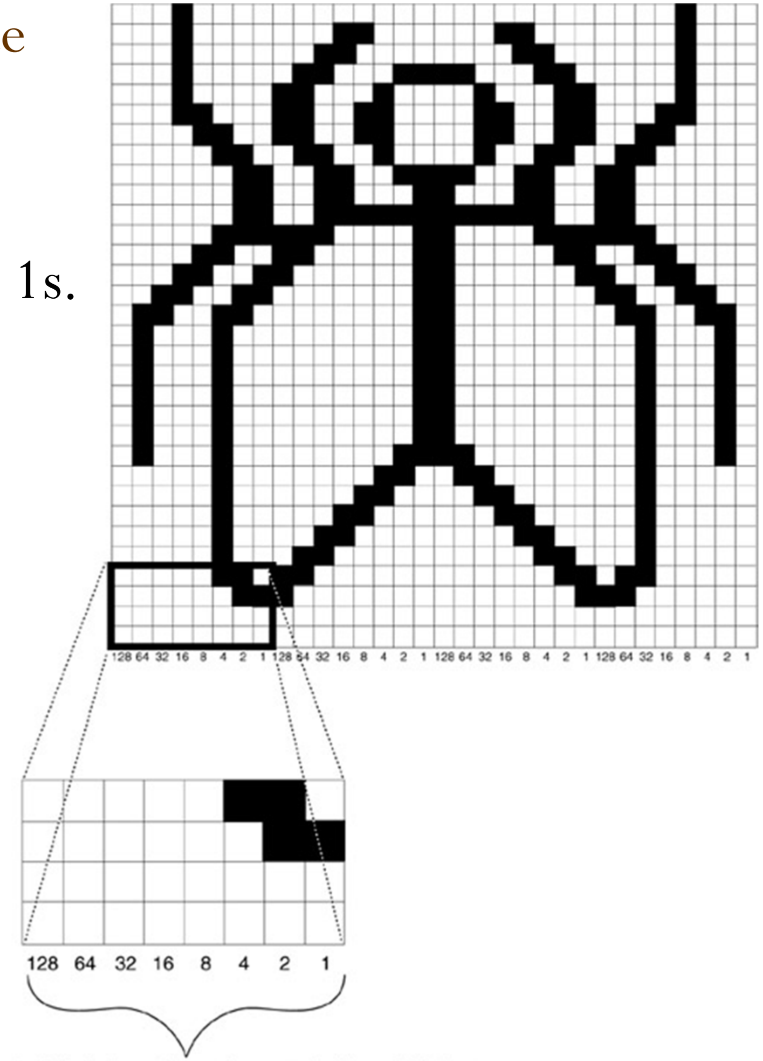
- To activate this feature

```
glEnable (GL_CULL_FACE);
glDisable (GL_CULL_FACE);
```

OpenGL: Stipple-Pattern Function

void glPolygonStipple (const GLubyte *mask);

mask: a pointer to a 32 × 32 bitmap that's interpreted as a mask of 0s and 1s.

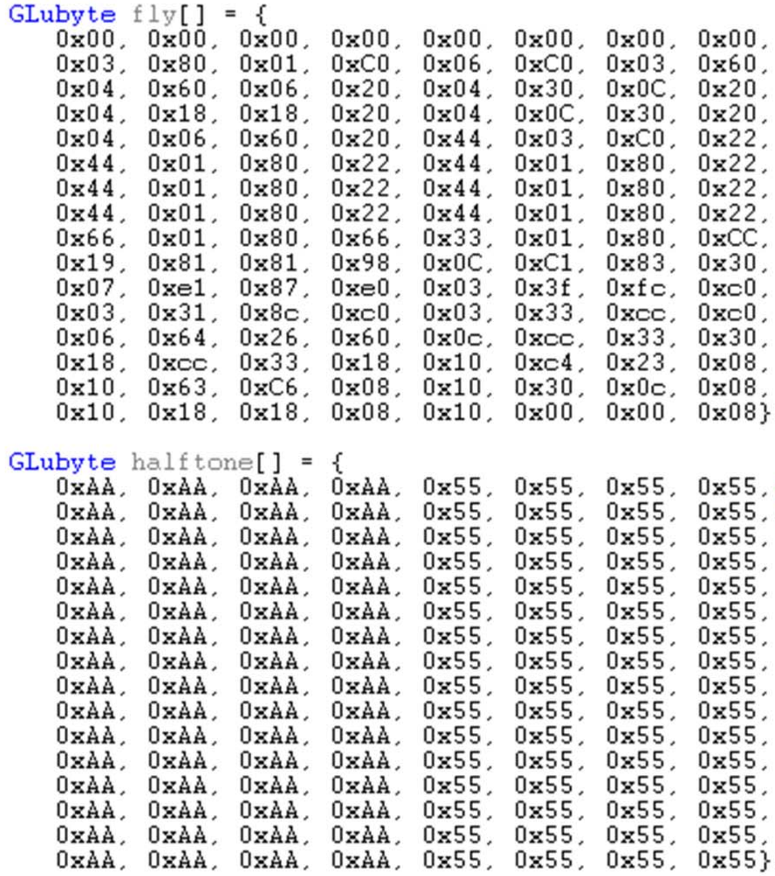


By default, for each byte the most significant bit is first.
Bit ordering can be changed by calling **glPixelStore*()**.

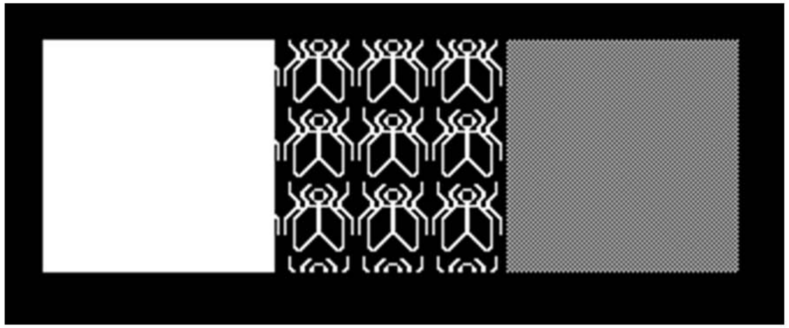
```
glEnable
(GL_POLYGON_STIPPLE);
glDisable
(GL_POLYGON_STIPPLE);
```

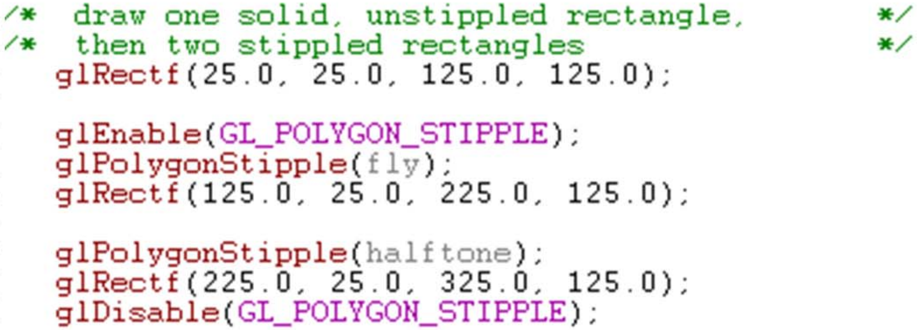
OpenGL: Stipple-Pattern Function

Example



```
GLubyte fly[] = {
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x03, 0x80, 0x01, 0xC0, 0x06, 0xC0, 0x03, 0x60,
    0x04, 0x60, 0x06, 0x20, 0x04, 0x30, 0x0C, 0x20,
    0x04, 0x18, 0x18, 0x20, 0x04, 0x0C, 0x30, 0x20,
    0x04, 0x06, 0x60, 0x20, 0x44, 0x03, 0xC0, 0x22,
    0x44, 0x01, 0x80, 0x22, 0x44, 0x01, 0x80, 0x22,
    0x44, 0x01, 0x80, 0x22, 0x44, 0x01, 0x80, 0x22,
    0x44, 0x01, 0x80, 0x22, 0x44, 0x01, 0x80, 0x22,
    0x66, 0x01, 0x80, 0x66, 0x33, 0x01, 0x80, 0xCC,
    0x19, 0x81, 0x81, 0x98, 0x0C, 0xC1, 0x83, 0x30,
    0x07, 0xe1, 0x87, 0xe0, 0x03, 0x3f, 0xfc, 0xc0,
    0x03, 0x31, 0x8c, 0xc0, 0x03, 0x33, 0xcc, 0xc0,
    0x06, 0x64, 0x26, 0x60, 0x0c, 0xcc, 0x33, 0x30,
    0x18, 0xcc, 0x33, 0x18, 0x10, 0xc4, 0x23, 0x08,
    0x10, 0x63, 0xC6, 0x08, 0x10, 0x30, 0x0c, 0x08,
    0x10, 0x18, 0x18, 0x08, 0x10, 0x00, 0x00, 0x08};

GLubyte halftone[] = {
    0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
    0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
    0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
    0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
    0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
    0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
    0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
    0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
    0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
    0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
    0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
    0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
    0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
    0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
    0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
    0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55};
```





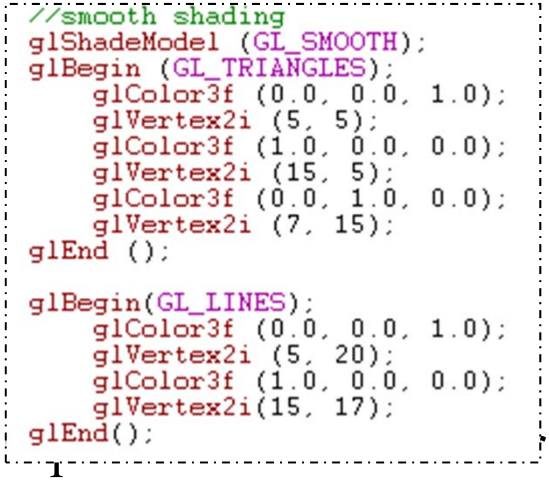
```
/*  draw one solid, unstippled rectangle,          */
/*  then two stippled rectangles                   */
glRectf (25.0, 25.0, 125.0, 125.0);

glEnable(GL_POLYGON_STIPPLE);
glPolygonStipple(fly);
glRectf (125.0, 25.0, 225.0, 125.0);

glPolygonStipple(halftone);
glRectf (225.0, 25.0, 325.0, 125.0);
glDisable(GL_POLYGON_STIPPLE);
```

OpenGL: Polygon Solid Color

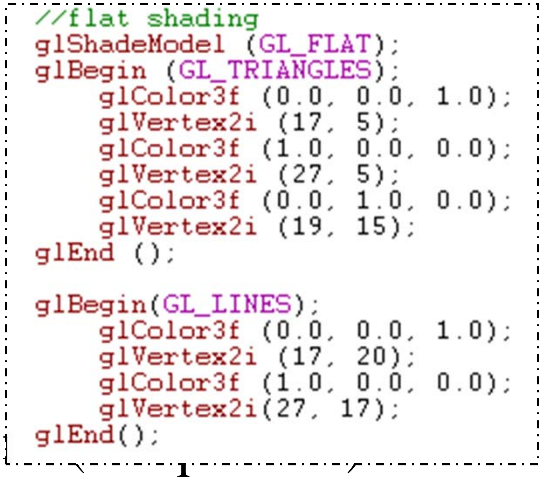
- Polygon filled with solid colors: GL_SMOOTH (default)

  an interpolation of the vertex colors

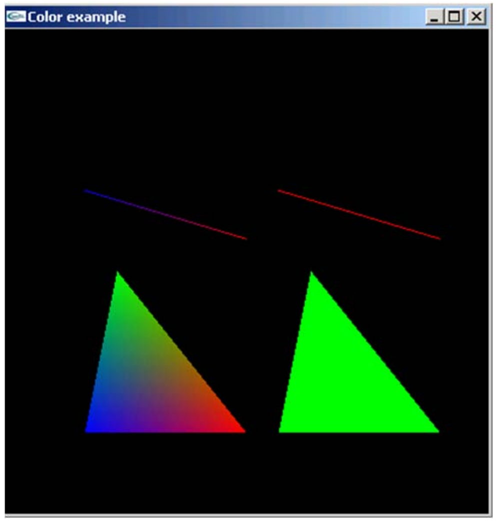
  E.g., Each of the three vertices is assigned different color. The polygon is to be filled as a linear interpolation of the vertices colors.



```
//smooth shading
glShadeModel (GL_SMOOTH);
glBegin (GL_TRIANGLES);
    glColor3f (0.0, 0.0, 1.0);
    glVertex2i (5, 5);
    glColor3f (1.0, 0.0, 0.0);
    glVertex2i (15, 5);
    glColor3f (0.0, 1.0, 0.0);
    glVertex2i (7, 15);
glEnd ();

glBegin(GL_LINES);
    glColor3f (0.0, 0.0, 1.0);
    glVertex2i (5, 20);
    glColor3f (1.0, 0.0, 0.0);
    glVertex2i(15, 17);
glEnd();
```



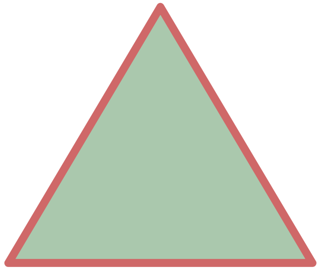
```
//flat shading
glShadeModel (GL_FLAT);
glBegin (GL_TRIANGLES);
    glColor3f (0.0, 0.0, 1.0);
    glVertex2i (17, 5);
    glColor3f (1.0, 0.0, 0.0);
    glVertex2i (27, 5);
    glColor3f (0.0, 1.0, 0.0);
    glVertex2i (19, 15);
glEnd ();

glBegin(GL_LINES);
    glColor3f (0.0, 0.0, 1.0);
    glVertex2i (17, 20);
    glColor3f (1.0, 0.0, 0.0);
    glVertex2i(27, 17);
glEnd();
```



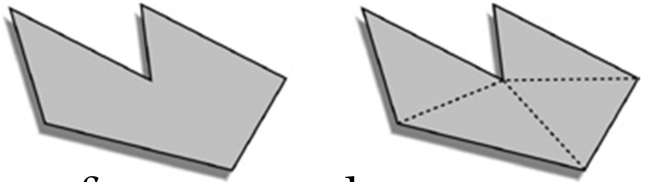
OpenGL: Polygon Wireframe Methods

- Example: Display a polygon with both an interior fill and a different color or pattern for its edges (or vertices).

```
glColor3f (0.0, 1.0, 0.0);
/* Invoke polygon-generating routine. */
// by default, GL_FILL

glColor3f (1.0, 0.0, 0.0);
glPolygonMode (GL_FRONT, GL_LINE);
/* Invoke polygon-generating routine again. */
```



OpenGL: Polygon Wireframe Methods



- Show concave polygons
  - A concave polygon is separated into a set of convex polygons to be rendered by OpenGL.
  - In a wire-frame form, the interior edges are shown.
  - To eliminate some edges, set edges bit flags to "off".

void glEdgeFlag (GLboolean *flag*);
void glEdgeFlagv (const GLboolean **flag*);

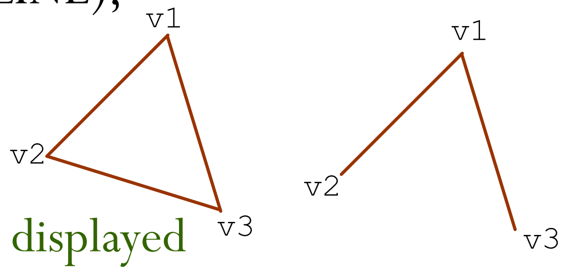
flag: GL_TRUE (default); GL_FALSE

- It is used between glBegin() and glEnd() pairs.
- It applies to all subsequently specified vertices until the next glEdgeFlag() call is made.

OpenGL: Polygon Wireframe Methods

- Example: Display only two edges of the define triangle

```
glPolygonMode (GL_FRONT_AND_BACK, GL_LINE);
glBegin (GL_POLYGON);
   glVertex3fv (v1);
   glEdgeFlag (GL_FALSE);
   glVertex3fv (v2); //the edge from v2 will not be displayed
   glEdgeFlag (GL_TRUE);
   glVertex3fv (v3);
glEnd ();
```

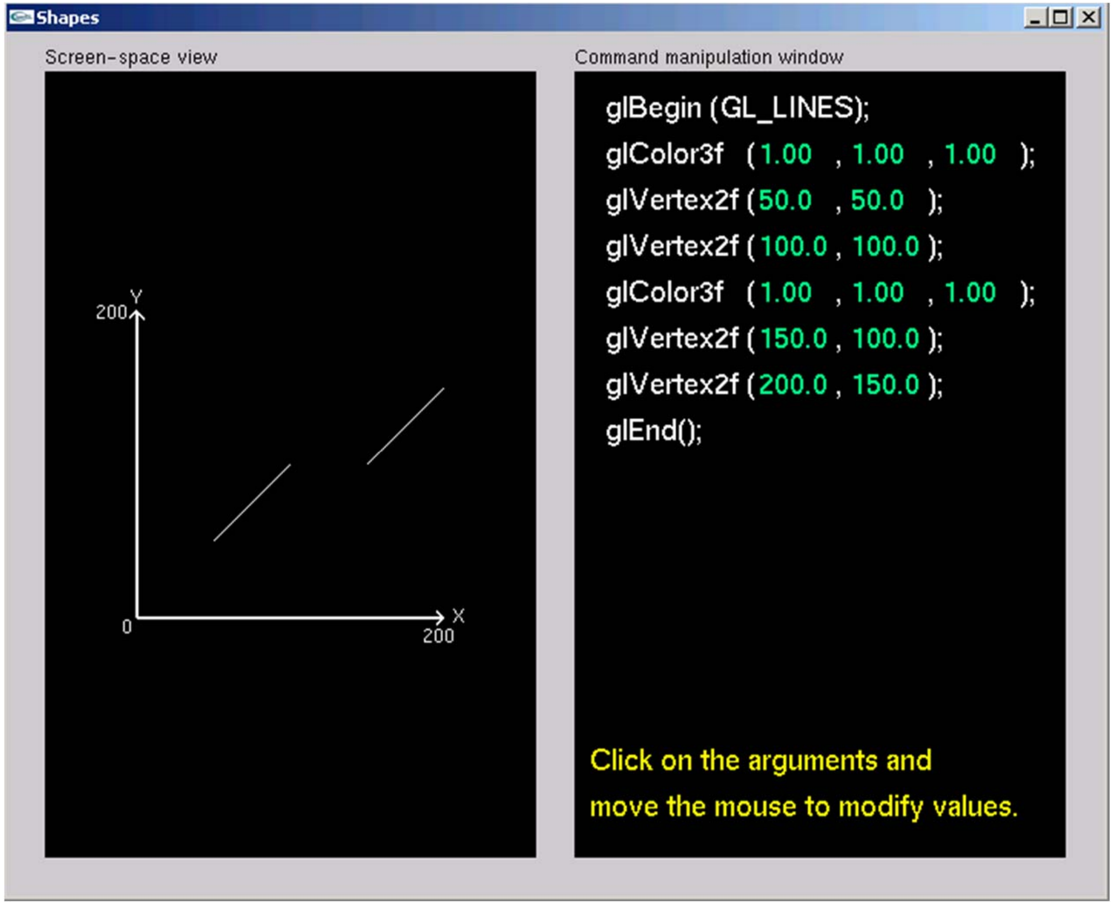


- To specify the polygon edge flags in an array:

```
glEnableClientState (GL_EDGE_FLAG_ARRAY);
glEdgeFlagPointer (offset, edgeFlagArray);
```

offset: the number of bytes between the values for the edge flags in edgeFlagArray; default: 0.

OpenGL Primitives and Attributes



```
glBegin (GL_LINES);
glColor3f  (1.00  , 1.00  , 1.00  );
glVertex2f (50.0  , 50.0  );
glVertex2f (100.0 , 100.0 );
glColor3f  (1.00  , 1.00  , 1.00  );
glVertex2f (150.0 , 100.0 );
glVertex2f (200.0 , 150.0 );
glEnd();
```

Click on the arguments and
move the mouse to modify values.

Nate Robins Tutorial: http://www.xmission.com/~nate/tutors.html

Fill-Area Methods

- Two basic methods for filling an area on raster systems
  - **Scan-Conversion**

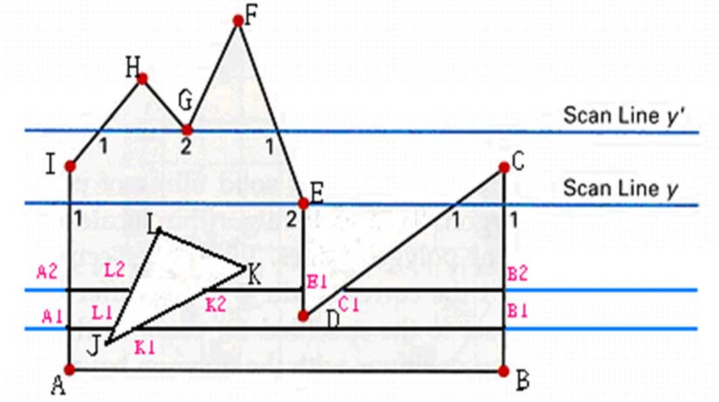
    Determine the overlap intervals for scan lines crossing the area, and set the pixel positions along these overlap intervals to the fill color.
  - **Boundary-Fill/ Flood-Fill**

    Start from a given interior position and "paint" outward, pixel by-pixel, until it gets to the boundary.

    Good for arbitrary shape of boundary.

Fill-Area Method on Raster Systems

- **Scan-Conversion**
  - Determining the intersection positions of the boundaries of the fill region with the screen scan lines;
  - To apply the fill color to each section of a scan line that lies in the interior of the fill region.



Scan-conversion to a fill-area (ABCDEFGHI-JKL) with the line segments produced at the scan lines:

A-B: [A,B]

A1-B1: [A1, L1], [K1, B1]

A2-B2: [A2, L2], [K2, E1], [C1, B2]

Odd-even rule:

odd – interior; even – exterior.

Chapter 5.
Attributes of Graphics Primitives

Part III.

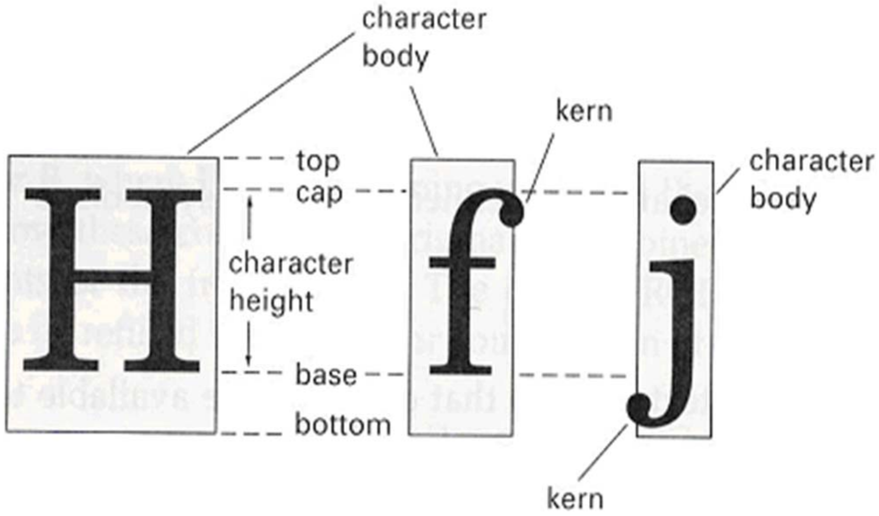
Character attributes and OpenGL functions

Outline

- OpenGL Character-Attributes Functions
- OpenGL Attribute Groups

Character Attributes

The appearance of displayed characters is controlled by attributes such as font, size, color, and orientation.



Examples of character bodies.

Character size (height) is specified by printers in *points*, where

1 *point* = 0.035146 centimeters.

Examples of Strings Sizes

- The **size** of text strings

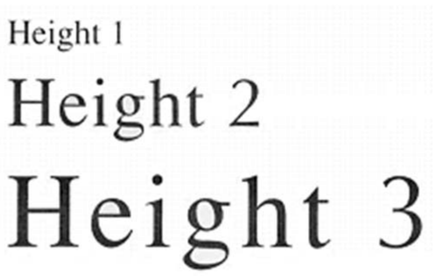


FIGURE 5-12

Different character-height settings with a constant width-to-height ratio.

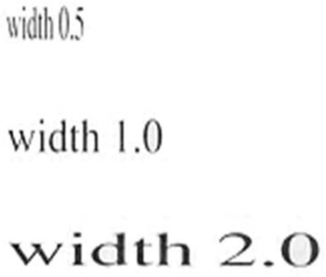


FIGURE 5-13

Varying sizes of the character

widths and a fixed height.

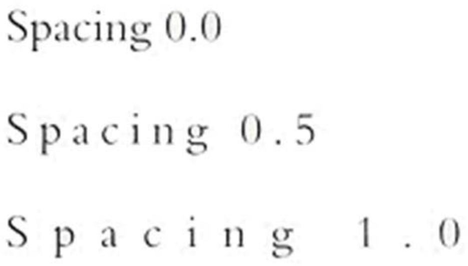


FIGURE 5-14

Text strings displayed with different character-spacing values.

Character Attributes

- Set the **orientation** according to the direction of a character up vector

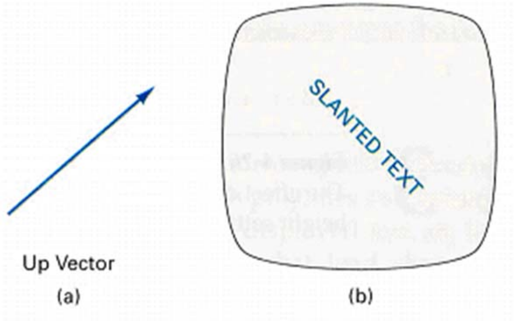


FIGURE 5-18 Direction of the up vector at 45° (a); controls the orientation of the displayed text (b).

- Set **text-path** up (vertically) and down (horizontally); left (forward) and right (backward).

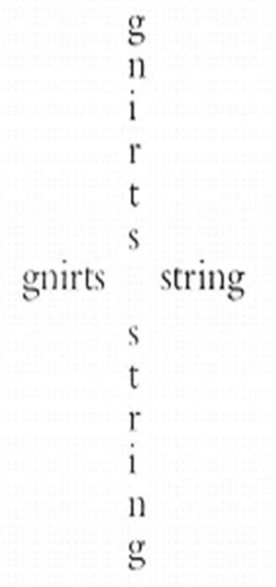


FIGURE 5-19 A text string displayed with the four text-path options: left, right, up, and down.

Character Attributes

- The **alignments** of a character string
  - How text is to be displayed with respect to a reference position.

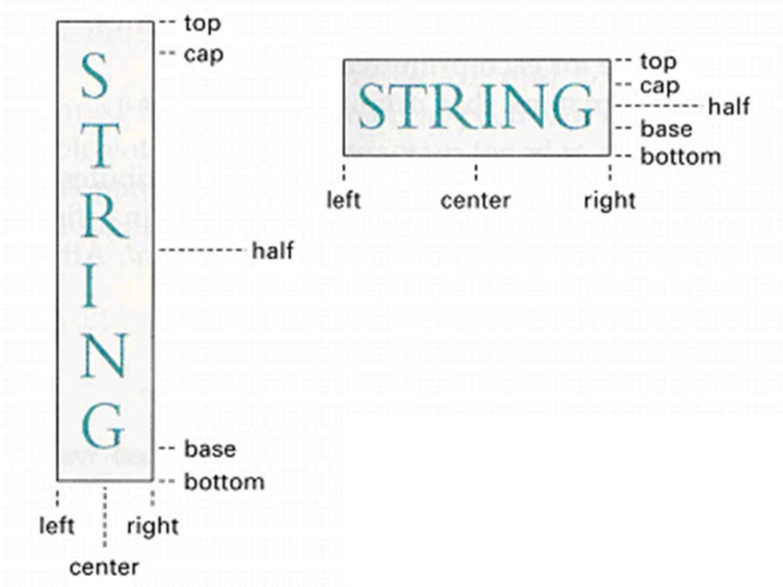


FIGURE 5-20 Typical character alignments for horizontal and vertical strings.

OpenGL Character-Attributes Functions

- Two methods for displaying characters in OpenGL
  - Design a font set using the bitmap functions in the OpenGL core library.

    glBitmap (w,h,x0,y0,xShift,yShift,pattern); [in 4-11]

  - Invoke the GLUT character-generation routines.

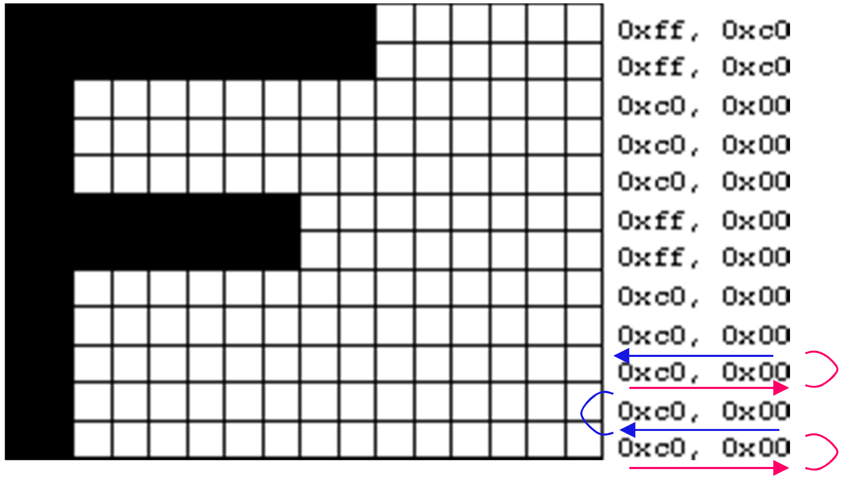
    GLUT library contains functions for displaying predefined bitmap and stroke character sets.

    glutBitmapCharacter(font,char); [in 4-13]

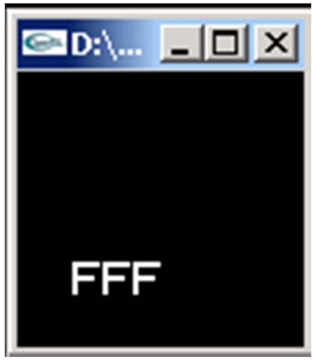
    glutStrokeCharacter(font,char);

Bitmap Function in OpenGL

- Example: draws the character F three times on the screen



```
0xff, 0xc0
0xff, 0xc0
0xc0, 0x00
0xc0, 0x00
0xc0, 0x00
0xff, 0x00
0xff, 0x00
0xc0, 0x00
0xc0, 0x00
0xc0, 0x00
0xc0, 0x00
0xc0, 0x00
```



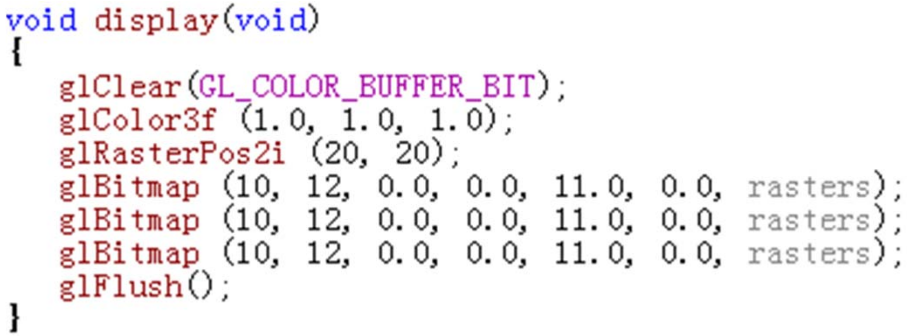
FFF

Shows the F as a bitmap and its corresponding bitmap data.

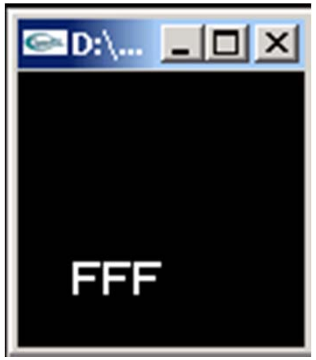
```
GLubyte rasters[24] = {
    0xc0, 0x00, 0xc0, 0x00, 0xc0, 0x00, 0xc0, 0x00, 0xc0, 0x00,
    0xff, 0x00, 0xff, 0x00, 0xc0, 0x00, 0xc0, 0x00, 0xc0, 0x00,
    0xff, 0xc0, 0xff, 0xc0};
```

Note: the visible part of the F character is at most 10 bits wide. Bitmap data is always stored in chunks that are multiples of 8 bits, but the width of the actual bitmap doesn't have to be a multiple of 8.

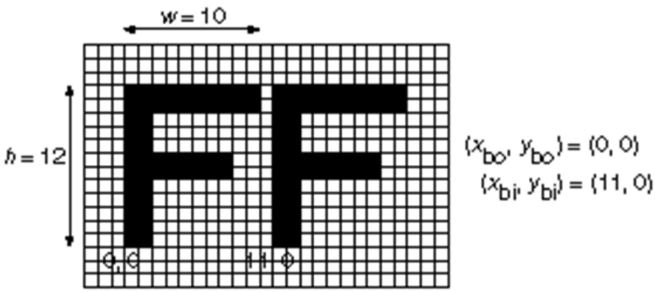
Bitmap Function in OpenGL



```
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f (1.0, 1.0, 1.0);
    glRasterPos2i (20, 20);
    glBitmap (10, 12, 0.0, 0.0, 11.0, 0.0, rasters);
    glBitmap (10, 12, 0.0, 0.0, 11.0, 0.0, rasters);
    glBitmap (10, 12, 0.0, 0.0, 11.0, 0.0, rasters);
    glFlush();
}
```



- Sets the current raster position: the current raster position is the origin where the next bitmap (or image) is to be drawn.
- Use the **glBitmap()** command to draw the data.



A Bitmap and Its Associated Parameters

OpenGL Character-Attributes Functions

- Color of Characters is defined by the Current Color.

- Size & Spacing is determined by Font designation in OpenGL/GLUT, such as **GLUT_BITMAP_9_BY_15**.

- Line Width & Line Type for the outline fonts may be defined with the **glLineWidth** and **glLineStipple** functions.

OpenGL Attribute Groups

- Attribute group: keep attributes and OpenGL state parameters.
  - Each group contains a set of related state parameters.
  - **20 different attribute groups** in OpenGL.

  Example

  - **Point** attribute group

    -- the size and point-smooth (anti-aliasing) parameters;
  - **Line** attribute group: 5 state variables

    -- the width, stipple status, stipple pattern, stipple repeat counter, and line smooth status.
  - **Polygon** attribute group

    -- eleven polygon parameters, such as fill pattern, front-face flag, and polygon-smooth status.
  - ……
- Some state variables are in **more than one** group
  - GL_CULL_FACE: both the polygon and the enable attribute groups.

OpenGL Attribute Groups

- Related OpenGL commands
  - The attributes are represented by bits in **mask**
  - Function to push the attributes onto the attribute stack

    void glPushAttrib (GLbitfield mask);

    mask: GL_POINT_BIT, GL_LINE_BIT, GL_POLYGON_BIT, GL_CURRENT_BIT (color parameter), GL_ALL_ATTRIB_BITS,…

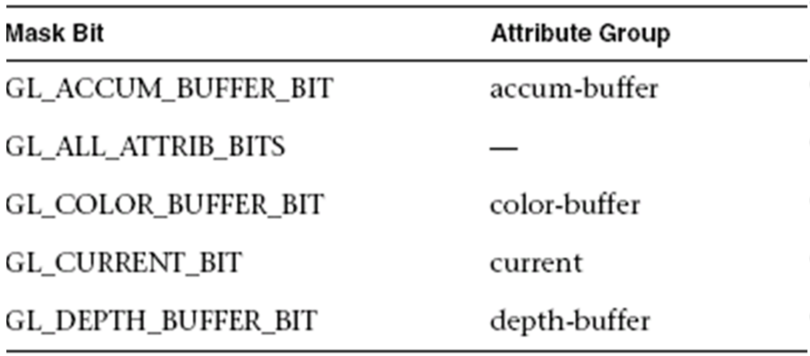
    Example: save attributes within two or more attribute groups onto an attribute stack:

    glPushAttrib (GL_POINT_BIT | GL_LINE_BIT | GL_POLYGON_BIT);

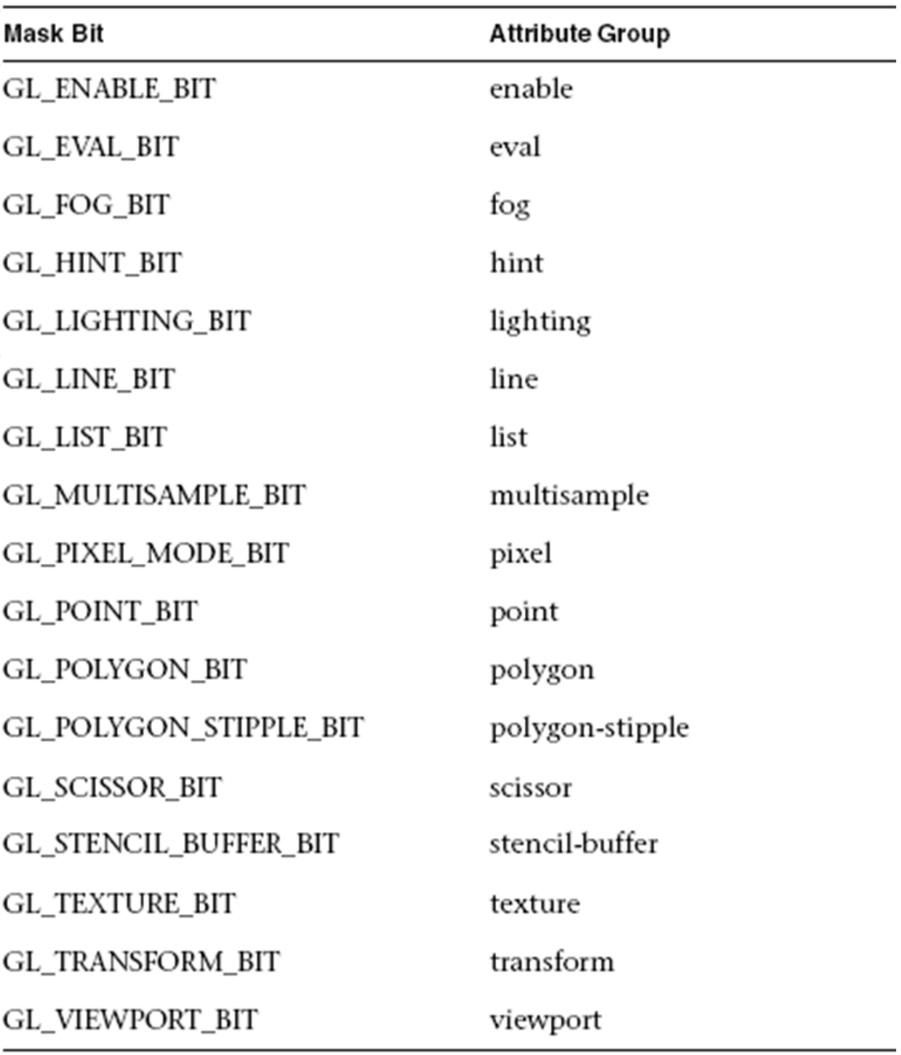
  - Function to restore the state variables which are saved with the last **glPushAttrib()**

    void glPopAttrib(void);

Attribute Groups



Mask Bit	Attribute Group
GL_ACCUM_BUFFER_BIT	accum-buffer
GL_ALL_ATTRIB_BITS	—
GL_COLOR_BUFFER_BIT	color-buffer
GL_CURRENT_BIT	current
GL_DEPTH_BUFFER_BIT	depth-buffer



Mask Bit	Attribute Group
GL_ENABLE_BIT	enable
GL_EVAL_BIT	eval
GL_FOG_BIT	fog
GL_HINT_BIT	hint
GL_LIGHTING_BIT	lighting
GL_LINE_BIT	line
GL_LIST_BIT	list
GL_MULTISAMPLE_BIT	multisample
GL_PIXEL_MODE_BIT	pixel
GL_POINT_BIT	point
GL_POLYGON_BIT	polygon
GL_POLYGON_STIPPLE_BIT	polygon-stipple
GL_SCISSOR_BIT	scissor
GL_STENCIL_BUFFER_BIT	stencil-buffer
GL_TEXTURE_BIT	texture
GL_TRANSFORM_BIT	transform
GL_VIEWPORT_BIT	viewport

(From: OpenGL Programming Guide, 7th)

Summary

- OpenGL is a state machine
  - State variables
- Attributes
  - Color
    - RGB/RGBA
    - Index: color lookup table
  - Blending
  - Line: width, style
  - Fill-area: style
  - Characters: bitmap, outline
- OpenGL functions and attribute groups