Chapter 9- Introduction to OpenGL

History of OpenGL

- Silicon Graphics (SGI) revolutionized the graphics workstation by implementing the pipeline in hardware (1982)
- To access the system, application programmers used a library called GL
- With GL, it was relatively simple to program three dimensional interactive applications

What is OpenGL

➢OpenGL is a software API to graphics hardware

✓designed as a streamlined, hardware-independent interface to be implemented on many different hardware platforms

✓procedural interface with c binding

✓No windowing commands!

✓No high-level commands for describing models of three- dimensional objects

➢Graphics rendering API (Low Level)

✓High-quality color images composed of geometric and image primitives

✓Window system independent

✓Operating system independent

✓Display device independent

➢Generate high-quality color images composed of geometric and image primitives

OpenGL Libraries

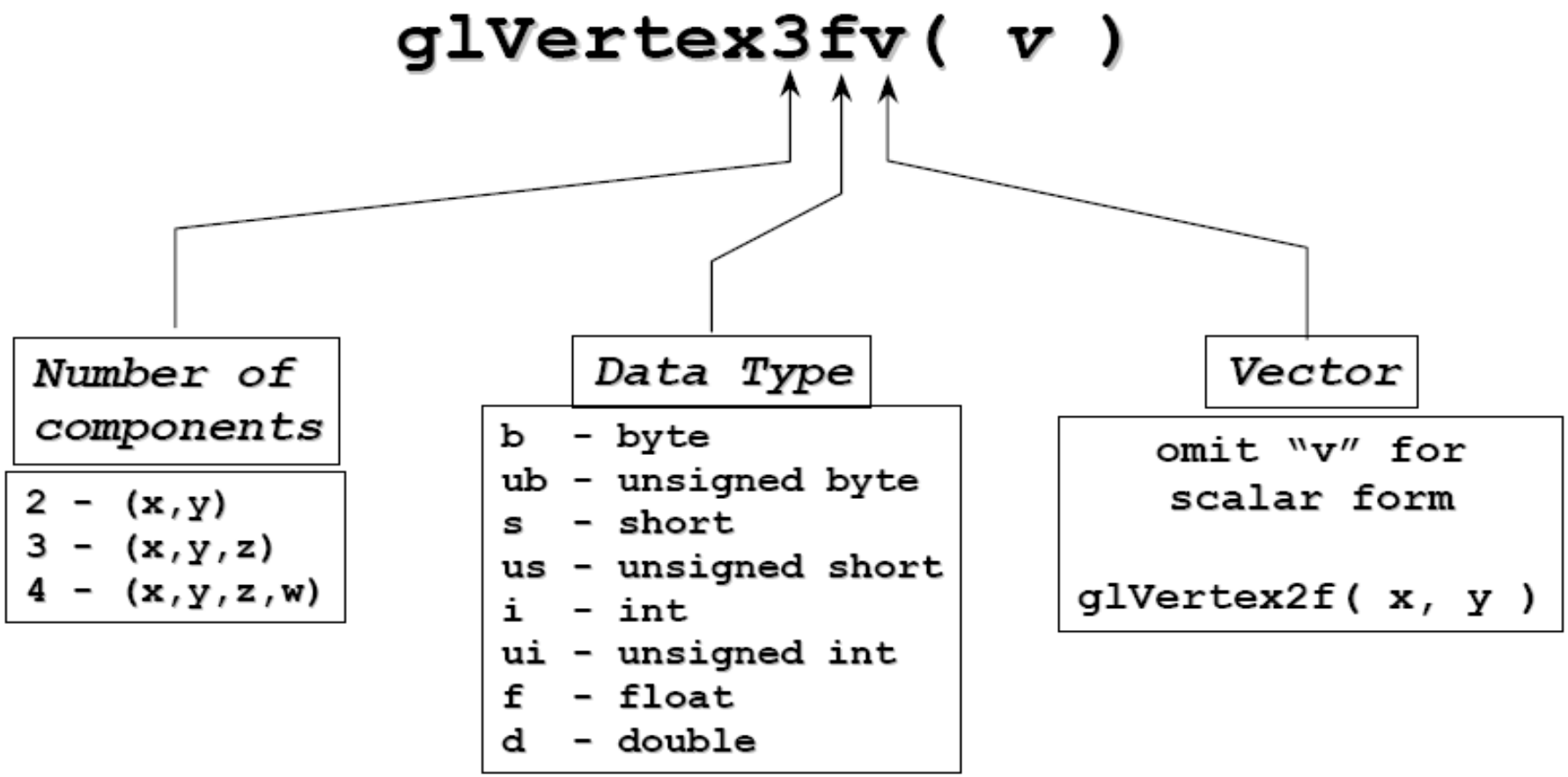
- GL (Graphics Library): Library of 2-D, 3-D drawing primitives and operations
  - API for 3-D hardware acceleration

  GLU (GL Utilities): Miscellaneous functions dealing with camera set-up and higher-level shape descriptions
- GLUT (GL Utility Toolkit): Window-system independent toolkit with numerous utility functions, mostly dealing with user interface

GLUT (GL Utility Toolkit)

- Window-system independent toolkit with numerous utility functions, mostly dealing with user interface
- GLUT provide a portable API for creating window and interacting with I/O devices
- Provides functionality common to all window systems
  - ✓Open a window
  - ✓Get input from mouse and keyboard
  - ✓Menus
  - ✓Event-driven

OpenGL Command format



GLUT Basics

Program Structure

1. Configure and open window (GLUT)
2. Initialize OpenGL (Optional)
3. Register input callback functions (GLUT)
   - Render
   - Resize
   - Input: keyboard, mouse, etc
4. Enter event processing loop (GLUT)

Program Structure

➢Most OpenGL programs have the following structure

- main():
  - ✓defines the callback functions
  - ✓opens one or more windows with the required properties
  - ✓enters event loop (last executable statement)
- init(): sets the state variables
  - ✓Viewing
  - ✓Attributes
- callbacks
  - ✓Display function
  - ✓Input and window functions

Callback functions

➢Callbacks are user-defined functions designed to react on specific events:

- ✓Whenever OpenGL decided it needs to redraw window contents
- ✓What to do when a user resizes a window.
- ✓Handle mouse motions…
- ✓React on keyboard,
- ✓What to do during idle period (no input from user),

Callback functions

- For OpenGL to become aware of your callbacks, you need to register them within it before you start drawing things.
- Some of the callbacks are mandatory, such as display, so that OpenGL know how to render your graphics.
- Programming interface for event-driven input
- Define a callback function for each type of event the graphics system recognizes
- This user-supplied function is executed when the event occurs
  - ✓GLUT example: **glutMouseFunc(mymouse)**

GLUT Callback Functions

- Contents of window need to be refreshed
  glutDisplayFunc()
- Window is resized or moved
  glutReshapeFunc()
- Key action
  glutKeyboardFunc()
- Mouse button action
  glutMouseFunc()
- Mouse moves while a button is pressed
  glutMotionFunc()
- Mouse moves regardless of mouse button state
  glutPassiveMouseFunc()
- Called when nothing else is going on
  glutIdleFunc()
- glutMainLoop()
  - –Runs forever waiting for an event. When one occurs, it is handled by the appropriate callback function.

Register Callback Functions

Set up any callback function you're going to use

```
void main (int argc, char **argv)
{
    ……
    glutDisplayFunc ( display );          // display callback
    glutReshapeFunc ( resize );           // window resize callback
    glutKeyboardFunc ( key );             // keyboard callback

    ……
}
```

Window Resize Callback

It's called when the window is resized or moved

```
void resize(int w, int h)
{
    ……
    display();
}
```

Rendering Callback

- Callback function where all our drawing is done
- Every GLUT program must have a display callback
- glutDisplayFunc(***my_display_func***);

```
/* this part is in main.c*/
void my_display_func (void )
{
    glClear( GL_COLOR_BUFFER_BIT );
    glBegin( GL_TRIANGLE );
      glVertex3fv( v[0] );
      glVertex3fv( v[1] );
      glVertex3fv( v[2] );
    glEnd();
    glFlush();
}
```

Idle Callback

- Use for animation and continuous update
  - Can use *glutTimerFunc* or *timed callbacks* for animations
- glutIdleFunc(***idle***);

```
void idle( void )
{
  /* change something */
  t += dt;
  glutPostRedisplay();
}
```

User Input Callbacks

- ✓ Process user input
- ✓ glutKeyboardFunc(my_key_events);

```
void my_key_events (char key, int x, int y )
{
   switch ( key ) {
     case 'q' : case 'Q' :
       exit ( EXIT_SUCCESS);
       break;
     case 'r' : case 'R' : rotate =
       GL_TRUE; break;
   }
}
```

Mouse Callback

- ✓ Captures mouse press and release events
- ✓ glutMouseFunc(my_mouse);

```
void myMouse(int button, int state, int x, int y)
{
      if (button == GLUT_LEFT_BUTTON && state ==
        GLUT_DOWN)
      {
      …
      }
}
```

Events in OpenGL

Event	Example	OpenGL Callback Function
Keypress	KeyDown KeyUp	glutKeyboardFunc
Mouse	leftButtonDown leftButtonUp	glutMouseFunc
Motion	With mouse press Without	glutMotionFunc glutPassiveMotionFunc
Window	Moving Resizing	glutReshapeFunc
System	Idle Timer	glutIdleFunc glutTimerFunc
Software	What to draw	glutDisplayFunc

Color Models: RGB

- ✓ Additive color
- ✓ Used in display screen. Pixels emit three kinds of light: Red, Green and Blue
- ✓ We choose Red, Green and Blue to be our primary colors.
- ✓ No set of 3 primary colors can generate all possible colors.
- ✓ But, Red, Green and Blue are close enough.

OpenGL RGB and RGBA modes

- ✓ "A" stands for alpha, refers to transparency.
- ✓ Alpha = 1.0 : Fully opaque
- ✓ Alpha = 0.0 : Fully transparent
- ✓ In RGB mode, alpha is assumed to be 1.0.
- ✓ Example:

```
glColor3f(0.5,1.0,0.6);              // RGB mode, fully opaque

glColor4f(0.5,1.0,0.6,0.3);  // RGBA mode, alpha set to 0.3
```

Color and grayscale

- ✓ Grayscale means from black to white (only vary in shade)
- ✓ Color means deviation from gray scale.
- ✓ In OpenGL, color is specified in RGB.

```
glColor3f(r,g,b);
```

where r, g and b are floating point numbers between 0.0 and 1.0, for example:

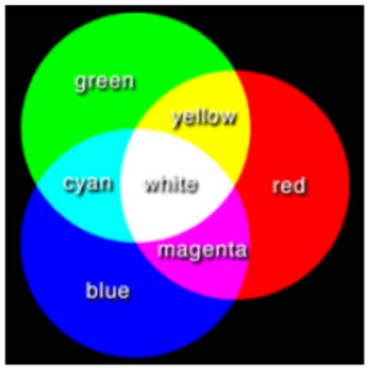
```
glColor3f(0.5,0.1,0.9);
```

This tells display to emit 0.5 intensity red light together with 0.1 intensity green light together with 0.9 intensity blue light.

Note: For grayscale, r=g=b.

OpenGL RGB Colors

Color Component R	G	B	Color Common Name
0	0	0	Black
0	0	1	Blue
0	1	0	Green
0	1	1	Cyan
1	0	0	Red
1	0	1	Magenta
1	1	0	Yellow
1	1	1	White



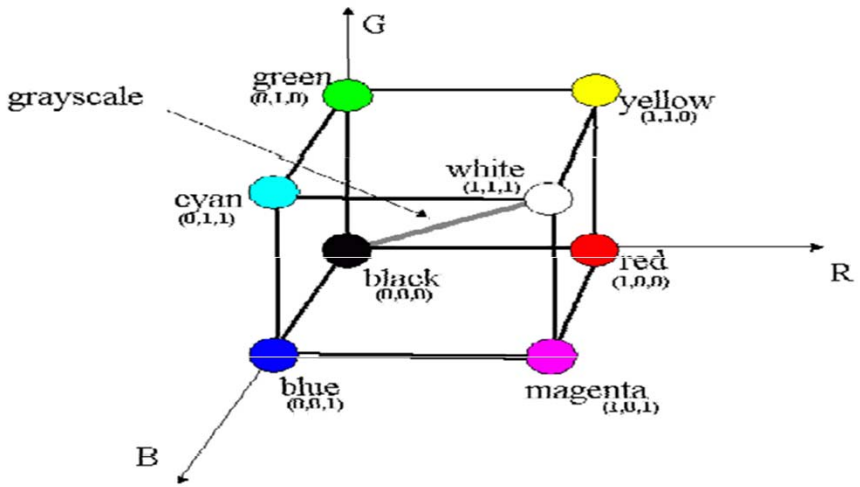
Drawing Attributes: Color

- ✓ **glColor3f(GLfloat r, GLfloat g, GLfloat b)** sets the drawing color
  - **glColor3d(), glColor3ui()** can also be used
  - Remember OpenGL is a state machine
  - Once set, the attribute applies to all subsequent defined objects until it is set to some other value
  - **glColor3fv()** takes a flat array as input
- ✓ There are more drawing attributes than color
  - Point size: **glPointSize()**
  - Line width: **glLinewidth()**
  - Dash or dotted line: **glLineStipple()**
  - Polygon pattern: **glPolygonStipple()**

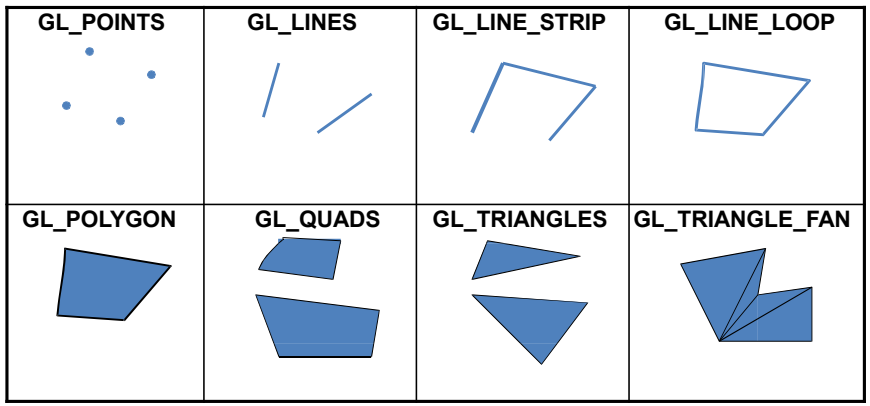
Color Functions

- ✓ **glColor3f(red value, green value, blue value) ;**
  - Used to specify the wanted color
  - Has three float parameter;
- ✓ **glClearColor(red value, green value, blue value, alpha value) ;**
  - Used to specify the initial background color.
  - Has four float parameters
  - Alpha value is used to determine the color of two overlapped objects
- ✓ **glClear (GL_COLOR_BUFFER_BIT);**
  - Used to set the bit value in the color buffer (refresh buffer) to the color indicated in the glClearColor function.

Color Values



2D Geometric Primitives



All geometric primitives are specified by vertices

Types

GL_POINTS

GL_LINES : each successive pair for a ling segment

GL_LINE_STRIP: vertices defining a sequence of line segments

GL_LINE_LOOP: GL_LINE_STRIP + the last vertex connects to the first

GL_POLYGON : sequence of vertices of polygon, filled

GL_QUADS: each successive group of four vertices for a quadrilaterals

GL_TRIANGLES: each successive group of three vertices for a triangle

GL_TRIANGLE_FAN: first three vertices for the first triangle and each subsequent vertex with the first vertex and the previous vertex for the next triangle

Specifying Geometric Primitives

```
glBegin( type );
    glVertex*(…);
    ……
    glVertex*(…);
glEnd();
```

type determines how vertices are combined

Geometry Commands

✓**glBegin(GLenum type)**
marks the beginning of a vertex-data list that describes a geometric primitives

✓**glEnd (void)**
marks the end of a vertex-data list

✓**glVertex*(…)**
specifies vertex for describing a geometric object

Example

```
void drawSquare ()
{
   glClear(GL_COLOR_BUFFER_BIT);
   glBegin(GL_POLYGON);
         glVertex2f ( 0.0, 0.0 );
         glVertex2f ( 1.0, 0.0 );
         glVertex2f ( 1.1, 1.1 );
         glVertex2f ( 0.0, 1.0 );
    glEnd();
    glFlush();          // force the renderer to
                        // output the results
}
   }
```

myDisplay()

```
void myDisplay(){
        glClear(GL_COLOR_BUFFER_BIT);
        glColor3f(1.0, 0.0, 0.0);       // set color to red
        glBegin(GL_POLYGON);
            glVertex2f(0.90, 0.50);
            glVertex2f(0.50, 0.90);
            glVertex2f(0.10, 0.50);
            glVertex2f(0.50, 0.10);
        glEnd();
       glColor3f(0.0, 0.0, 1.0);       // set color to blue
      glRectf(0.25, 0.25, 0.75, 0.75);  // draw a rectangle
//        glFlush();                // force OpenGL to render
        glutSwapBuffers();      // swap buffers
}
```

Drawing: Miscellaneous

✓**glColor():** Range is [0, 1] for each color channel

✓**glRect(x1, y1, x2, y2)** specifying opposite corners of rectangle is equivalent to GL_POLYGON with four vertices listed (i.e., filled)

✓Can set persistent attributes outside of **glBegin()/ glEnd()**

- **glPointSize(GLfloat size)**
- **glLineWidth(GLfloat width)**

Geometric Primitives: Points, Lines and Polygons Example

```
#ifdef __FLAT_____
#include <windows.h>
#endif
#include <gl/glut.h>


// The initialization
function

void init(void)
{
  glutInitWindowSize( glutGet(
            GLUT_SCREEN_WIDTH
            )/3, glutGet(
            GLUT_SCREEN_HEIGH
            T)/3 );
  glutInitWindowPosition( 0, 0 );
  glutInitDisplayMode(GLUT_D
  OUBLE | GLUT_RGB);
  glutCreateWindow(  Rendering
  Primitives) ;
  glClearColor(1.0, 1.0, 1.0, 0.0) ;
  glShadeModel(GL_FLAT) ;
}
```

Geometric Primitives: Points, Lines and Polygons Example

```
// The display callback function
void display(void)
{
  static float v[] = { 0.1, 0.1 };
  glClear(GL_COLOR_BUFFER_BIT) ;
  glColor3f(0.0, 0.0, 0.0 ); // Set the point color
  to black
  glPointSize( 3.5 );
  glBegin( GL_POINTS);
      glVertex2fv( v );
      glVertex2f( 0.05, 0.2);
      glVertex2f( 0.05, 0.3);
      glVertex2f( 0.05, 0.4);
      glVertex2f( 0.05, 0.5);
      glVertex2f( 0.1, 0.2);
      glVertex2f( 0.1, 0.3);
      glVertex2f( 0.1, 0.4);
      glVertex2f( 0.1, 0.5);
      glVertex2i( 0, 0 );
      glVertex2f( -0.1, -0.1);
```

Geometric Primitives: Points, Lines and Polygons Example

```
        glVertex2f( -0.05,-0.2);
        glVertex2f( -0.05,-0.3);
        glVertex2f( -0.05,-0.4);
        glVertex2f( -0.05,-0.5);
        glVertex2f( -0.1,-0.2);
        glVertex2f( -0.1,-0.3);
        glVertex2f( -0.1,-0.4);
        glVertex2f( -0.1,-0.5);
glEnd();

glBegin( GL_LINES );
        glColor3f(0.0, 0.0, 1.0); // Set the point color to blue
        glVertex2f( 0.5, 0.5 );
        glVertex2f( 0.1, 0.1);
glEnd();
```

Geometric Primitives: Points, Lines and Polygons Example

```
glBegin( GL_QUADS );
        glColor3f( 1.0, 0.0, 0.0); // Draw the quad in red
        glVertex2f( -0.8, -0.8);
        glVertex2f( -0.5, -0.3);
        glVertex2f( -0.2, -0.9);
        glVertex2f( -0.4, -1.0);
glEnd();
glBegin( GL_POLYGON );
        glColor3f( 0.0, 0.0, 0.0);
        glVertex2f( -0.9, 0.6);
        glVertex2f( -0.8, 0.55);
        glVertex2f( -0.7, 0.6);
        glVertex2f( -0.6,0.8);
        glVertex2f( -0.85, 0.9);
glEnd();
glColor3f(0.2, 0.9, 0.1);
glRectf(-.1, .6, .3, .9);
glutSwapBuffers();
}
```

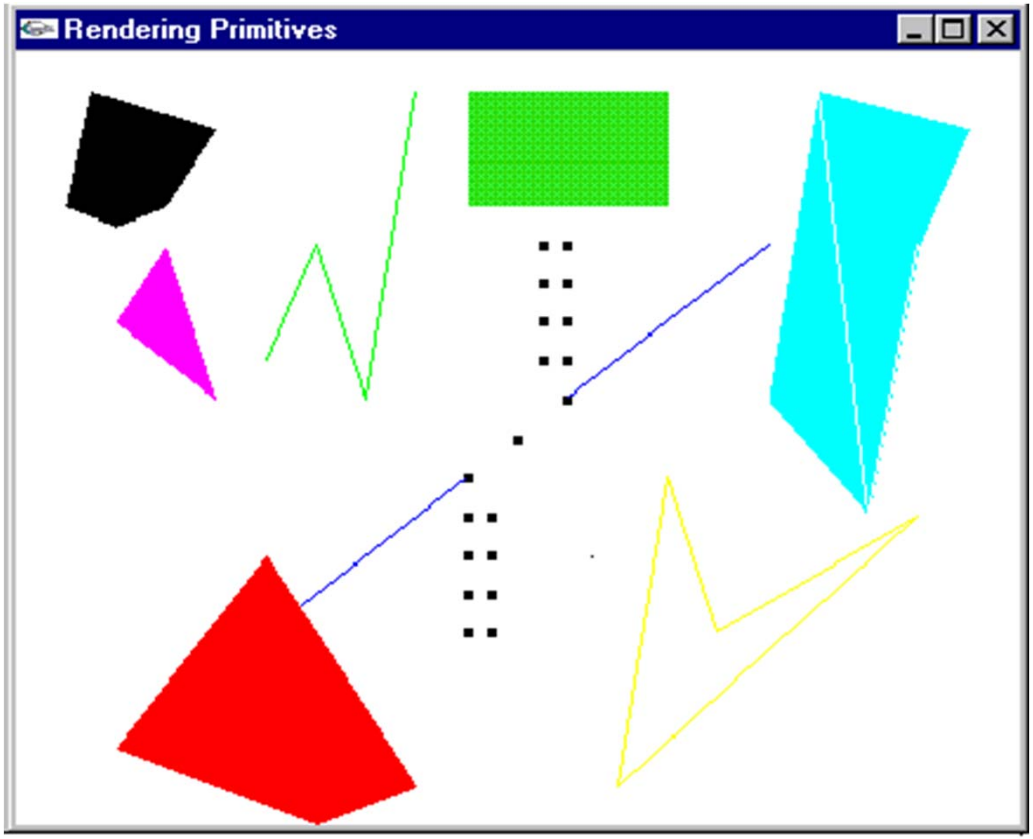
Geometric Primitives: Points, Lines and Polygons Example

```
// The main function
int main(int argc, char** argv)
{
  glutInit(&argc, argv) ;

  init();
  glutDisplayFunc(display) ;
  glutMainLoop() ;

  return 0 ;
}
```

Geometric Primitives: Points, Lines and Polygons Example



Lights

- ✓Create and select a *lighting model.*
- ✓Define material properties for the objects in the scene.
- ✓And most important … **Enable the lights**:
  - *glEnable(GL_LIGHTING);*
  - *glEnable(GL_LIGHTING);*

Creating & Positioning Lights

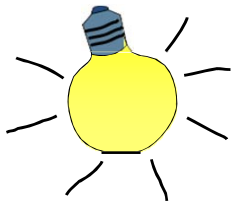
void ***glLight****(light, pname, param);*

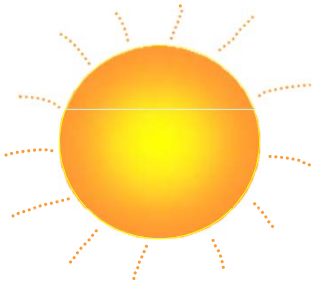
- Creates the light specified by light, which can be GL_LIGHT0, ... , or GL_LIGHT7
- The characteristic of the light being set is defined by pname, which specifies a named parameter
- param indicates the values to which the pname characteristic is set.
- *pname* can get one of several values:

GL_AMBIENT, GL_DIFFUSE & GL_SPECULAR define the light RGBA values for each of the light components.

Directional and Positional Lights

- ✓Directional light source is positioned at infinity (like the sun).
- ✓Positional light source is positioned near the scene and its exact position determines its effect.





Creating & Positioning Lights

- ✓when *GL_POSITION* is passed as an argument to glLight*() four values (x,y,z,w) are passed as parameters.
  - **W** determines the type of light we are defining:
  - 0 ⇔ directional - (x,y,z) is the direction.
  - 1 ⇔ positional - (x,y,z) is the position.

- ✓*GL_CONSTANT_ATTENUATION, GL_LINEAR_ATTENUATION, GL_QUADRATIC_ATTENUATION.*
  - These define the attenuation of the light. Usually disabled for *directional* lights.
- ✓*GL_SPOT_DIRECTION, GL_SPOT_EXPONENT, GL_SPOT_CUTOFF.*
  - These define spotlights and spotlight properties.

Multiple Light Sources

✓You can define several light sources by calling glLight*() several times with different light names:

- glLightfv(GL_LIGHT0, GL_AMBIENT, light0_ambient);
- glLightfv(GL_LIGHT1, GL_AMBIENT, light1_ambient);

Lighting Model

✓*void **glLightModel***(pname, param);*

- GL_LIGHT_MODEL_AMBIENT defines the global ambient RGBA values.
- GL_LIGHT_MODEL_LOCAL_VIEWER determines whether we are using a local or infinite viewpoint.
- GL_LIGHT_MODEL_TWO_SIDE determines whether we are using two sided lighting.

✓*void **glMaterial***(face, pname, param);*

- Specifies a current material property for use in lighting calculations.
- Face can be GL_FRONT, GL_BACK, or GL_FRONT_AND_BACK to indicate which face of the object the material should be applied to.