.OpenGL

Definition:

- OpenGL is an application programming interface (API), which is merely a software library for accessing features in graphics hardware.
- OpenGL is designed as a streamlined, hardware-independent interface that can be implemented on many different types of graphics hardware systems, or entirely in software independent of a computer's operating system.
- A sophisticated library that provides these features could certainly be built on top of OpenGL. The OpenGL Utility Library (GLU) provides many of the modeling features, such as quadric surface and NURBS curves and surfaces. GLU is a standard part of every OpenGL implantation.
- It was first developed at Silicon Graphics Computer System in July of 1994.

What OpenGL does not do?

- OpenGL doesn't include functions that for performing windowing tasks or processing user input;
- OpenGL doesn't provide any functionality for describing models of three-dimensional objects, or operations for describing models of 3-D objects, or operations for reading image files (like JPEG files, for example). Instead, you must construct your three dimensional objects from a small set of geometric primitives – points, lines, triangles, and patches.

Information Repositories for OpenGL

- http://www.opengl.org
- http://www.sgi.com/software/openngl/manual.html
- F.S. Hill, Stephen M. Kelly, " Computer Graphics using OpenGL", Prentice Hall
- Dave Shreiner, " OpenGL Programming Guide", Addison-Wesley

What you need

With any system, you start with a good C/C++ compiler and install appropriate OpenGL header files and libraries. Three libraries with associated files are required to use OpenGL:

OpenGL (the basic API tool)

GLU (the OpenGL Utility Library)

GLUT (the OpenGL Utility Toolkit, a windowing tool kit that handles window system operations).

Adding Header files

Place the three files *gl.h, glu.h*, and *glut.h* in a gl subdirectory of the *include* directory of your compliler.

In each application you write the following include satements:

```
#include <gl/Gl.h>
#include <gl/Glu.h>
#include <gl/glut.h>
```

Linking Library Files

Each application be part of a project that is compiled and linked. In addition to the (.c or .cpp) files that you write, add the appropriate OpenGL library (.lib) files to your project so that the linker can find them.

Windows-based Programming

Many modern graphics systems are window based and manage the display of multiple overlapping windows. The user can move windows around the screen by means of mouse and can resize the windows.

Event-driven Programming

Another property of most windows-based programs is that they are *event driven*. This means that the program responds to various events, such as the click of the mouse, the press of a key on a keyboard, or the resizing of a window on screen.

The system automatically manages an event queue, which receives messages stating the certain events have occurred and deals with them on a first-come, first-served, basis. The programmer organizes a program as a collection of *callback functions* that are executed when an even occur. A callback function is created for each type of event that might take place. When the system removes an event from the queue, it simply executes the callback functions associated with the type of that event.

OpenGL comes with a Utility Toolkit, which provides tools to assist with event management. They are four principal types of events, and a "**glut**" function is available for each:

- glutDisplayFunc (myDisplay) : whenever the system determines that a window should be redrawn on the screen, it issues a "redraw" event. This happens when the window is first opened and when the window is exposed by moving other window off of it. Here, the function myDisplay( ) is registered as the call back function for a redraw event.
- glutReshapFunc (myReshape) : Screen windows can be reshaped by the user, usually by dragging a corner of the window to a new position with the mouse. Here the function myReshape( ) is registered with the reshape. The function myReshape( ) is automatically passed arguments that reports the new width and height of the reshaped window.
- glutMouseFunc(myMouse) : when one of the mouse buttons is pressed or released, a mouse event is issued. Here, myMouse( ) is registered as the function to be called when a mouse even occurs. The function myMouse( ) is automatically passed arguments that describe the location of the mouse and the nature of the action initiated by pressing the botton.
- glutKeyboardFunc(myKeyboard) : This command registers the function myKeyboard( ) with the event of pressing or releasing some key on the keyboard. The function myKeyboard( ) is automatically passed arguments that tell which key was pressed. Conventionally, it is also passed data indicating the location of the mouse at the time the key was pressed.

```
void main ( )
{
        initialize things
        create a screen window
        glutDisplayFunc (myDispaly) ;          // register the redraw function
        glutReshapeFunc (myReshape) ;          // register the reshape function
        glutMouseFunc (myMouse) ;              // register the mouse action function
        glutKeyboardFunc (myKeybaord) ;        // register the keyboard action function
        perhaps initialize other things
        glutMainLoop ( ) ;                     // enter the unending main loop
}
```

Figure: A skeleton of an event-driven program using OpenGL

glutMainLoop( ) : when this instruction is executed, the program draws the initial picture and enters an unending loop, in which it simply waits for events to occur. (A program is normally terminated by clicking in the "go away" box that is attached to each window.)

Opening a Window for Drawing

The first task in making pictures is to open a screen window for drawing. In the figure below, the first five function calls the OpenGL Utility Toolkit to open a window for drawing.

```
// appropriate #include go here
Void main( int argc, char** argv)
{
        glutInit( &argc, argv) ;              // initialize the toolkit
        glutInitDispalyMode ( GLUT_SINGLE | GLUT_RGB) ;          // set the display mode
        glutInitWindowSize( 640, 480) ;                  // set window size
        glutInitWindowPosition ( 100, 150 );             // set the window position on screen
        glutCreateWindow ( "my first attempt" ) ;        // open the screen window

        // register the callback functions
         glutDisplayFunc (myDispaly) ;          // register the redraw function
        glutReshapeFunc (myReshape) ;           // register the reshape function
        glutMouseFunc (myMouse) ;               // register the mouse action function
        glutKeyboardFunc (myKeybaord) ;         // register the keyboard action function

        myInit( ) ;                             // additional initializations as necessary
        glutMainLoop ( ) ;                      // enter the unending main loop
```

Figure: Code using the OpenGL Utility Toolkit to open the initial window for drawing

The first five functions initialize and display the screen window in which our program will produce graphics.

- glutInit(&argc, argv) ; This function initializes the OpenGL Utility Toolkit. Its arguments are the standard ones for passing information about command line.

- glutInitDispalyMode (GLUT_SINGLE | GLUT_RGB) ; This function specifies how the display should be initialized. The built-in constants GLUT_SINGLE and GLUT_RGB, which are ORed together, indicate that a single buffer be allocated and that colors are specified using desired amounts of red, green and blue. (Double buffering is used for animation.)
- glutInitWindowSize(640, 480) ; This function specifies that the screen window should initially be 640 pixels by 480 pixels. When the program is running, the user can resize the window as desired.
- glutInitWindowPosition (100, 150); This function specifies that the window's upper left corner should be positioned on the screen 100 pixels over the left and 150 pixels down the top. When the program is running, the user can move this window whenever desired.
- glutCreateWindow ("my first attempt") ; This function actually opens an displays the screen window, putting the title "my first attempt" in the title bar.

The OpenGL code to create the window:

```
#include<windows.h>
#include<gl/Gl.h>
#include<gl/glut.h>

//myInit
void myInit(void)
{

}

void myDisplay(void)
{

}

int main(int argc, char** argv)
{
```

```
    glutInit(&argc, argv);            //initialize the toolkit
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);          // set display mode
    glutInitWindowSize(640, 480);                         // set window size
    glutInitWindowPosition(100, 150);                  // set window position on screen
    glutCreateWindow("my first attempt");               //open the screen window
    glutDisplayFunc(myDisplay);                       // register redraw function
    myInit();
    glutMainLoop();                              // go into perpetual loop
    return 0;


}
```

Output:

Drawing Basic Graphics Primitives

OpenGL provides tools for drawing the output primitives. Most of them, such as points, polylines, and polygons, are defined by one or more *vertices.* To draw such objects in OpenGL, you pass it a list of vertices. The list occurs between the two OpenGL function calls glBegin( ) and glEnd ( ). The arguments of glBegin( ) determines which object is drawn.

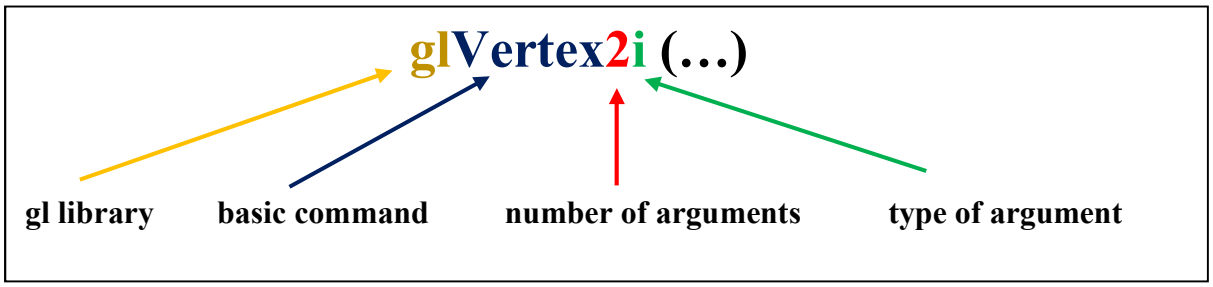
For instances:

Drawing three points on the screen:

```
glBegin( GL_POINTS)
        glVertex2i( 100, 50 ) ;
        glVertex2i( 100, 130 ) ;
        glVertex2i( 150, 130 ) ;
glEnd( ) ;
```

- The constant GL_POINTS is built-into OpenGL. To draw other primitives, you replace GL_POINTS with GL_LINES, GL_POLYGON etc.

Format of glVertx2i(…) :



The prefix "gl" indicates the function from the OpenGL library. Then comes the basic command root followed by the number of arguments being sent to the function, and, finally, the type of arguments (i for an integer ,f or d for a floating-point value.

Following commands can be used to pass the floating value instead of integer:

```
glBegin( GL_POINTS)
        glVertex2d( 100.0, 50.0 ) ;
        glVertex2d( 100.0, 130.0 ) ;
        glVertex2d( 150.0, 130.0 ) ;
glEnd( ) ;
```

OpenGL Data Types

Suffix	Data Types	Typical C or C++ types	OpenGL type name
b	8-bit integer	signed char	GLbyte
s	16-bit integer	short	GLshort
i	32-bit integer	int or long	Glint, GLsizei
f	32-bit floating point	float	GLfloat, GLclampf
d	64-bit floating point	double	GLdouble, GLclampd
ub	8-bit unsigned number	unsigned char	GLubyte, GLboolean
us	16-bit unsinged number	unsinged short	GLushort
ui	32-bit unsinged number	unsigned int or unsinged long	GLuint, GLenum, GLbitfield

If you wish to encapsulate the OpenGL commands for drawing a dot in a generic function such as drawDot( ), you can use the following code:

```
Void drawDot (GLint x, Glint y)
{
        // draw dot at integer point (x, y)
        glBegin(GL_POINTS) ;
        glVertex2i(x, y) ‘
        glEnd( ) ;
```

Figure: Encapsulating OpenGL details in the generic function drawDot( )

The OpenGL "State"

OpenGL keeps track of many state variable, such as the current size of point, the current color of a drawing, the current background color etc. The value of a state variable remains active until a new value is given.

Setting the size of point:

Syntax: glPointSize (floating_number);

Example: glPointSize (4.0) ; // the point is drawn as a square, four pixels on a sie.

Setting the color of a drawing:

Syntax: glColor3f (red, green, blue) ;

Where the value of red, green, and blue vary between 0.0 and 1.0.

Examples:

glColor3f (1.0, 0.0, 0.0); // set drawing color to red

glColor3f (0.0, 1.0, 0.0); // set drawing color to green

glColor3f (0.0, 0.0, 1.0); // set drawing color to blue

glColor3f (0.0, 0.0, 0.0); // set drawing color to black

glColor3f (1.0, 1.0, 1.0); // set drawing color to white

glColor3f (1.0, 1.0, 0.0); // set drawing color to yellow

Setting background color

Syntax: glClearColor (red, green, blue, alpha);

where the value of red, green, and blue vary between 0.0 and 1.0, and, alpha specifies a degree of transparency.

Clearing the entire window to background color

Syntax: glClear (GL_COLOR_BUFFER_BIT);

where, GL_COLOR_BUFFER_BIT is one of the constants built in OpenGL.

Establishing the Coordinate System

- myInit ( ) function is used to set up the coordinate system in the program.
- OpenGL routinely performs a large number of transformations. It uses matrices to do this, and the commands in myInit ( ) manipulate certain matrices to accomplish the desired goal.
- The gluOrtho2D ( ) routine sets the transformation we need for a screen window size (i.e. 640 pixels wide by 480 pixels high, in this program)

```
void myInit (void)
{
        glMatrixMode (Gl_PROJECTION);
        glLoadIdentity ( ) ;
        gluOrtho2D(0, 640.0, 480.0 ) ;
}
```

Putting it all Together: A complete OpenGL program to draw three points

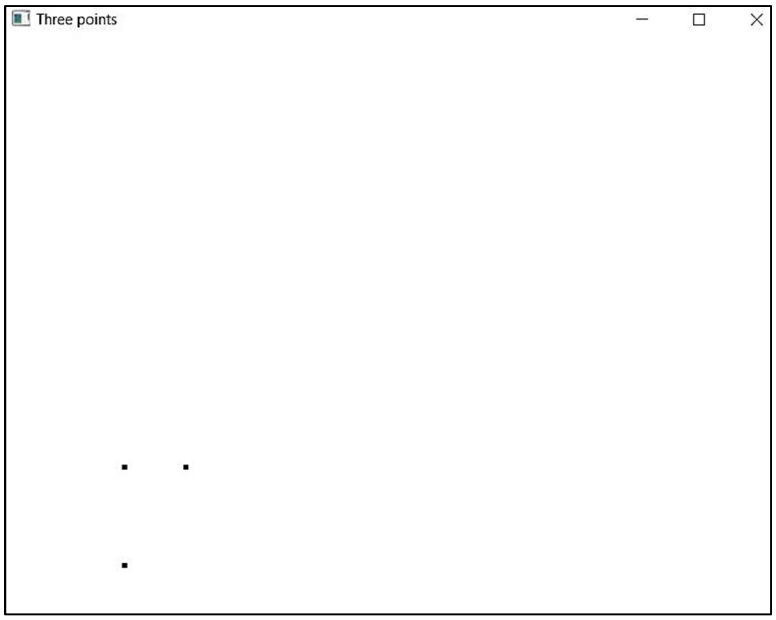
- Here the initialization in myInit ( ) sets up the coordinate system, the point size, the background color, and the drawing color.
- The drawing is encapsulated in the callback function myDisplay ( ). Because this program is not interactive, no other callback functions are used.
- The function glFlush ( ) is called after the dots are drawn, to ensure that all data are completely processed and sent to display. This is important is some systems that operate over the network; data are buffered on the host machine and sent to the remote display only when the buffer becomes full or glFlush( ) is executed.

```
#include<windows.h>
#include<gl/Gl.h>
#include<gl/glut.h>
//myInit
void myInit(void)
```

```
{
   glClearColor(1.0, 1.0, 1.0, 0.0);                    //set white background color
   glColor3f(0.0f, 0.0f, 0.0f)  ;                       //set the drawing color
   glPointSize(4.0);                                    //a 'dot' is 4 by 4 pixels
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity( );
   gluOrtho2D(0.0, 640.0, 0.0, 480.0);
}
void myDisplay(void)
{
   glClear(GL_COLOR_BUFFER_BIT);                        //clear the screen
   glBegin(GL_POINTS);
            glVertex2i(100, 50);                        // draw the three points
            glVertex2i(100, 130);
            glVertex2i(150, 130);
   glEnd();
   glFlush();                                           // send all output to display
}
int main(int argc, char** argv)
{
   glutInit(&argc, argv);            //initialize the toolkit
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);          // set display mode
   glutInitWindowSize(640, 480);                         // set window size
   glutInitWindowPosition(100, 150);                     // set window position on screen
   glutCreateWindow("Three points");                     //open the screen window
   glutDisplayFunc(myDisplay);                           // register redraw function
   myInit();
```

```
    glutMainLoop();                                    // go into perpetual loop
    return 0;
}
```

Output:



Drawing a line

Use GL_LINES as the argument to glBegin ( ), and pass it the two endpoints as vertices. Thus, to draw a line between (40, 100) and (202, 96), use the following code:

```
glBegin ( GL_Lines );                    // use GL_LINES here
        glVertex2i (40, 100) ;
        glVertex2i ( 202, 96) ;
glEnd( ) ;
```

- the function glLineWidth (floating_constant) is used to set the width of a line. For example: glLineWidth(4.0) set the line width of 4 thickness. The default value is 1.0.

Drawing Polylines and Polygons

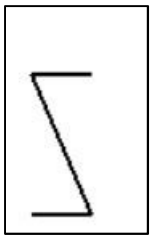
- A polyline is a collection of line segments joined end to end. It is described the ordered list of points, as in the equation

  $p_0 = (x_0, y_0)\ p_1 = (x_1, y_1),\ \ldots\ldots\ p_n = (x_n, y_n)$.

- In OpenGL, a polyline is called a "line strip" and is drawn by specifying the vertices in turn, between glBegin(Gl_LINE_STRIP) and glEnd ( ).
- For example, the given code below produces a polyline.

```
glBegin (GL_LINE_STRIP) ;     // draw an open polyline
        glVertex2i ( 20, 10) ;
        glVertex2i ( 50, 10) ;
        glVertex2i ( 20, 80) ;
        glVertex2i ( 50, 10) ;
glEND ( ) ;
```

Output:

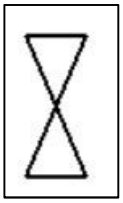


- If it is desired to connect the last point with the first point to make the polyline into a polygon, simple replace GL_LINE_STRIP with GL_LINE_LOOP. The resulting polygon is shown in figure below.
- Polygon drawn suing GL_LINE_LOOP cannot be filled with a color or pattern. To draw filled polygons, you must use gl_Begin (GL_POLYGON).

Code:

```
glBegin (GL_LINE_LOOP) ;     // draw an open polygon
        glVertex2i ( 20, 10) ;
        glVertex2i ( 50, 10) ;
        glVertex2i ( 20, 80) ;
        glVertex2i ( 50, 10) ;
glEND ( ) ;
```

Output:



Drawing Aligned Rectangle:

- A special case of polygon is the **aligned rectangle**, so called because it side are aligned with the coordinate axes.
- OpenGL provide the ready-made function to draw the rectangle.

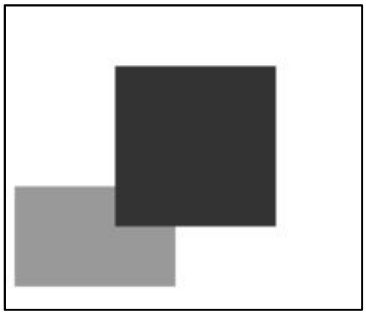
Syntax: glRecti (GLint x1, GLint y1, GLint x2, GLint y2) ;

This function draw a rectangle with opposite corners (x1, y1) and (x2, y2) and fill the rectangle with the current color.

For example:

```
glClearColor ( 1.0, 1.0, 1.0, 0.0 );          // white background
glClear(Gl_COLOR_BUFFER_BIT);                 // clear the window
glCOLOR3f(0.6, 0.6, 0.6);                     // bright gray
glRecti(20, 20, 100, 70);                     // draw a rectangle
glColor3f(0.2, 0.2, 0.2);                     // dark gray
glRecti(70, 50, 150, 130);                    // draw a rectangle
glFlush( ) ;
```

Output:



Code: Drawing a House

```
#include<windows.h>
#include<gl/Gl.h>
#include<gl/glut.h>
//myInit
void myInit(void)
{
   glClearColor(1.0, 1.0, 1.0, 0.0);    //set white background color
   glColor3f(0.0f, 0.0f, 0.0f)  ;       //set the drawing color
   glPointSize(4.0);                    //a 'dot' is 4 by 4 pixels
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity( );
```

```
   gluOrtho2D(0.0, 640.0, 0.0, 480.0);
}
void myDisplay(void)
{
   glClear(GL_COLOR_BUFFER_BIT);       //clear the screen
   glBegin(GL_LINE_LOOP);
            glVertex2i(40, 40);              // draw the shell of house
            glVertex2i(40, 90);
            glVertex2i(70, 120);
            glVertex2i(100, 90);
            glVertex2i(100,40);
   glEnd();
         glBegin(GL_LINE_STRIP);
            glVertex2i(50, 100);              // draw the chimney
            glVertex2i(50, 120);
            glVertex2i(60, 120);
            glVertex2i(60, 110);
   glEnd();
   glBegin(GL_LINE_STRIP);
            glVertex2i(50, 40);              // draw the door
            glVertex2i(50, 60);
            glVertex2i(60, 60);
            glVertex2i(60, 40);
   glEnd();
   glBegin(GL_LINE_LOOP);
            glVertex2i(80, 55);              // draw the window
            glVertex2i(80, 65);
            glVertex2i(90, 65);
            glVertex2i(90, 55);
```

```
    glEnd();
    glFlush();                    // send all output to display
}
int main(int argc, char** argv)
{
    glutInit(&argc, argv);          //initialize the toolkit
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);          // set display mode
    glutInitWindowSize(640, 480);                  // set window size
    glutInitWindowPosition(100, 150);                  // set window position on screen
    glutCreateWindow("my first house attempt");              //open the screen window
    glutDisplayFunc(myDisplay);                  // register redraw function
    myInit();
    glutMainLoop();                        // go into perpetual loop
    return 0;
}
```

Output: