Chapter 2: COMPUTER GRAPHICS by F.S HILLs. Initial steps in drawing figures (polygon, rectangle etc)
Objective:

Basic understanding of simple code in OpenGL and initial steps in drawing.

Event-driven programming.

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

*all of the callback functions are defined here*

**glutDisplayFunc(myDisplay);** Whenever the system determines that a screen window should be redrawn it issues a “redraw” event. This happens when the window is first opened, and when the window is exposed by moving another window off of it. Here the function myDisplay() is registered as the callback function for a redraw event.

**glutReshapeFunc(myReshape);** Screen windows can be reshaped by the user, usually by dragging a corner of the window to a new position with the mouse. (Simply moving the window does not produce a reshape event.) Here the function myReshape() is registered with the “reshape” event. As we shall see, myReshape() is automatically passed arguments that report 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 event occurs. myMouse() is automatically passed arguments that describe the mouse location and the nature of the button action.

**glutKeyboardFunc(myKeyboard);** This registers the function myKeyboard() with the event of pressing or releasing some key on the keyboard. myKeyboard() is automatically passed arguments that tell which key was pressed. Conveniently, it is also passed data as to the location of the mouse at the time the key was pressed.
Opening a Window for Drawing.

Code using the OpenGL utility toolkit to open the initial window for drawing.

```c
void main(int argc, char** argv)
{
    glutInit(&argc, argv); // initialize the toolkit
    glutInitDisplayMode(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(myDisplay);
    glutReshapeFunc(myReshape);
    glutMouseFunc(myMouse);
    glutKeyboardFunc(myKeyboard);

    myInit(); // additional initializations as necessary
    glutMainLoop(); // go into a perpetual loop
}
```

`glutInit(&argc, argv);` This function initializes the toolkit. Its arguments are the standard ones for passing command line information; we will make no use of them here.

`glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);` This function specifies how the display should be initialized. The built-in constants GLUT_SINGLE and GLUT_RGB, which are OR’d together, indicate that a single display buffer should be allocated and that colors are specified using desired amounts of red, green, and blue. (Later we will alter these arguments: for example, we will use double buffering for smooth animation.)

`glutInitWindowSize(640,480);` This function specifies that the screen window should initially be 640 pixels wide by 480 pixels high. When the program is running the user can resize this window as desired.

`glutInitWindowPosition(100, 150);` This function specifies that the window’s upper left corner should be positioned on the screen 100 pixels from the left edge and 150 pixels down from the top. When the program is running the user can move this window wherever desired.
glutCreateWindow("my first attempt"); This function actually opens and displays the screen window, putting the title “my first attempt” in the title bar.

The remaining functions in main() register the callback functions as described earlier, perform any initializations specific to the program at hand, and start the main event loop processing. The programmer (you) must implement each of the callback functions as well as myInit().

### Drawing Basic Graphics Primitives.

We want to develop programming techniques for drawing a large number of geometric shapes that make up interesting pictures. The drawing commands will be placed in the callback function associated with a redraw event, such as the myDisplay() function mentioned above.

OpenGL provides tools for drawing all of the output primitives such as points, lines, polylines, and polygons, and these are defined by one of 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 argument of glBegin() determines which object is drawn.
Example: To draw three points

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

```c
glBegin(GL_TRIANGLE_FAN);  // draw triangle
    glColor3f(1.0f,0.0f,0.0f);  // set color to red
    glVertex3f( 0.0f, 30.0f, 0.0f);
    glColor3f(0.0f,1.0f,0.0f);  // set color to green
    glVertex3f(-50.0f, -50.0f, 50.0f);
    glColor3f(1.0f,1.0f,0.0f);  // set color to yellow
    glVertex3f( 50.0f, -50.0f, 50.0f);
    glColor3f(0.0f,0.0f,1.0f);  // set color to blue
    glVertex3f( 50.0f, -50.0f, -50.0f);
    glColor3f(1.0f,1.0f,1.0f);  // set color to white
    glVertex3f(-50.0f, -50.0f, -50.0f);
    glColor3f(0.0f,1.0f,0.0f);  // set color to green
    glVertex3f(-50.0f, -50.0f, 50.0f);
glEnd();
```

```c
glBegin(GL_QUADS);         // draw square
    glColor3f(0.0f,1.0f,0.0f);  // set color to green
    glVertex3f(-50.0f, -50.0f, 50.0f);
    glColor3f(1.0f,1.0f,1.0f);  // set color to white
    glVertex3f(-50.0f, -50.0f, -50.0f);
    glColor3f(0.0f,0.0f,1.0f);  // set color to blue
    glVertex3f( 50.0f, -50.0f, -50.0f);
    glColor3f(1.0f,1.0f,0.0f);  // set color to yellow
    glVertex3f( 50.0f, -50.0f, 50.0f);
glEnd();
```
<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>GL_POINTS</td>
<td>Treats each vertex as a single point. Vertex ( n ) defines point ( n ). ( N ) points are drawn.</td>
</tr>
<tr>
<td>GL_LINES</td>
<td>Treats each pair of vertices as an independent line segment. Vertices ( 2n - 1 ) and ( 2n ) define line ( n ). ( N/2 ) lines are drawn.</td>
</tr>
<tr>
<td>GL_LINE_STRIP</td>
<td>Draws a connected group of line segments from the first vertex to the last. Vertices ( n ) and ( n+1 ) define line ( n ). ( N - 1 ) lines are drawn.</td>
</tr>
<tr>
<td>GL_LINE_LOOP</td>
<td>Draws a connected group of line segments from the first vertex to the last, then back to the first. Vertices ( n ) and ( n + 1 ) define line ( n ). The last line, however, is defined by vertices ( N ) and ( 1 ). ( N ) lines are drawn.</td>
</tr>
<tr>
<td>GL_TRIANGLES</td>
<td>Treats each triplet of vertices as an independent triangle. Vertices ( 3n - 2 ), ( 3n - 1 ), and ( 3n ) define triangle ( n ). ( N/3 ) triangles are drawn.</td>
</tr>
<tr>
<td>GL_TRIANGLE_STRIP</td>
<td>Draws a connected group of triangles. One triangle is defined for each vertex presented after the first two vertices. For odd ( n ), vertices ( n ), ( n + 1 ), and ( n + 2 ) define triangle ( n ). For even ( n ), vertices ( n + 1 ), ( n ), and ( n + 2 ) define triangle ( n ). ( N/2 ) triangles are drawn.</td>
</tr>
<tr>
<td>GL_TRIANGLE_FAN</td>
<td>Draws a connected group of triangles. One triangle is defined for each vertex presented after the first two vertices. Vertices ( 1 ), ( n + 1 ), ( n + 2 ) define triangle ( n ). ( N - 2 ) triangles are drawn.</td>
</tr>
<tr>
<td>GL_QUADS</td>
<td>Treats each group of four vertices as an independent quadrilateral. Vertices ( 4n - 3 ), ( 4n - 2 ), ( 4n - 1 ), and ( 4n ) define quadrilateral ( n ). ( N/4 ) quadrilaterals are drawn.</td>
</tr>
<tr>
<td>GL_QUAD_STRIP</td>
<td>Draws a connected group of quadrilaterals. One quadrilateral is defined for each pair of vertices presented after the first pair. Vertices ( 2n - 1 ), ( 2n ), ( 2n + 2 ), and ( 2n + 1 ) define quadrilateral ( n ). ( N/2 - 1 ) quadrilaterals are drawn. Note that the order in which vertices are used to construct a quadrilateral from strip data is different from that used with independent data.</td>
</tr>
<tr>
<td>GL_POLYGON</td>
<td>Draws a single, convex polygon. Vertices 1 through ( N ) define this polygon.</td>
</tr>
</tbody>
</table>
Points

\[ \text{GL_POINTS} \]

Lines

\[ \text{GL_LINES} \]
\[ \text{GL_LINE_STRIP} \]
\[ \text{GL_LINE_LOOP} \]

Triangles

\[ \text{GL_TRIANGLES} \]
\[ \text{GL_TRIANGLE_FAN} \]
\[ \text{GL_TRIANGLE_STRIP} \]

Rectangles

\[ \text{GL_QUADS} \]
\[ \text{GL_QUAD_STRIP} \]

Polygon

\[ \text{GL_POLYGON} \]

Little more detail…
GL_TRIANGLE_STRIP: draws a series of triangles based on triplets of vertices: \( v_0, v_1, v_2, \) then \( v_2, v_1, v_3, \) then \( v_2, v_3, v_4, \) etc. (in an order so that all triangles are “traversed” in the same way; e.g. counterclockwise).

- GL_TRIANGLE_FAN: draws a series of connected triangles based on triplets of vertices: \( v_0, v_1, v_2, \) then \( v_0, v_2, v_3, \) then \( v_0, v_3, v_4, \) etc.

- GL_QUAD_STRIP: draws a series of quadrilaterals based on foursomes of vertices: first \( v_0, v_1, v_3, v_2, \) then \( v_2, v_3, v_5, v_4, \) then \( v_4, v_5, v_7, v_6, \) (in an order so that all quadrilaterals are “traversed” in the same way; e.g. counterclockwise).

### Format of OpenGL commands.

![Format of OpenGL commands diagram](image)

### OpenGL Data Types

OpenGL works internally with specific data types: for instance, functions such as `glVertex2i()` expect integers of a certain size (32 bits).

<table>
<thead>
<tr>
<th>suffix</th>
<th>data type</th>
<th>typical C or C++ type</th>
<th>OpenGL type name</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>8-bit integer</td>
<td>signed char</td>
<td>GLbyte</td>
</tr>
<tr>
<td>s</td>
<td>16-bit integer</td>
<td>short</td>
<td>GLshort</td>
</tr>
<tr>
<td>i</td>
<td>32-bit integer</td>
<td>int or long</td>
<td>GLint, GLsizei</td>
</tr>
<tr>
<td>f</td>
<td>32-bit floating point</td>
<td>float</td>
<td>GLfloat, GLclampf</td>
</tr>
<tr>
<td>d</td>
<td>64-bit floating point</td>
<td>double</td>
<td>GLdouble, GLclampd</td>
</tr>
<tr>
<td>ub</td>
<td>8-bit unsigned number</td>
<td>unsigned char</td>
<td>GLubyte, GLboolean</td>
</tr>
<tr>
<td>us</td>
<td>16-bit unsigned number</td>
<td>unsigned short</td>
<td>GLushort</td>
</tr>
<tr>
<td>ui</td>
<td>32-bit unsigned number</td>
<td>unsigned int or unsigned long</td>
<td>GLuint, GLenum, GLbitfield</td>
</tr>
</tbody>
</table>

### The OpenGL “State”
OpenGL keeps track of many state variables, such as the current “size” of points, the current color of drawing, the current background color, etc. The value of a state variable remains active until a new value is given.

Establishing the Coordinate System

Our method for establishing our initial choice of coordinate system will seem obscure here, but will become clearer in the next chapter when we discuss windows, viewports, and clipping. Here we just take the few required commands on faith. The myInit() function is a good place to set up the coordinate system.

```c
void myInit(void)
{
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, 640.0, 0, 480.0);
}
```

Putting it together: A Complete OpenGL program

Figure given below shows a complete program that draws the lowly three dots. It is easily extended to draw more interesting objects as we shall see. 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(). As this program is non-interactive, no other callback functions are used.

```c
#include <windows.h> // use as needed for your system
#include <gl/gl.h>
#include <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);
}
//<<<<<<<<<<<<<<<<<<< myDisplay >>>>>>>>>>>>>>>>
void myDisplay(void)
{
```


```c
#include <GL/glut.h>
void init (void)
{
    glClearColor (1.0, 1.0, 1.0, 0.0);
    glMatrixMode (GL_PROJECTION);
    gluOrtho2D (0.0, 200.0, 0.0, 150.0);
}
void lineSegment (void)
{
    glClear (GL_COLOR_BUFFER_BIT);
    glColor3f (0.0, 0.0, 1.0);
    glBegin (GL_LINES);
    glVertex2i (180, 15);
    glVertex2i (10, 145);
    glEnd ( );
    glFlush ( );
}
void main (int argc, char** argv)
{
    glutInit (&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize (640, 480);
    glutInitWindowPosition (100, 150);
    glutCreateWindow ("my first attempt");
    glutDisplayFunc (myDisplay);
    myInit ();
    glutMainLoop ();
}
```

## Draw LINE

```c
void lineSegment (void)
{
    glClear (GL_COLOR_BUFFER_BIT);
    glColor3f (0.0, 0.0, 1.0);
    glBegin (GL_LINES);
    glVertex2i (180, 15);
    glVertex2i (10, 145);
    glEnd ( );
    glFlush ( );
}
```
glutInit (&argc, argv);
glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize (50, 100);
glutCreateWindow("An Example OpenGL Program");
init ( );
glutDisplayFunc (lineSegment);
glutMainLoop ( );
}

Drawing the Sierpinski Gasket.

# include <windows.h> // use as needed for your system
# include <gl/Gl.h>
# include <gl/glut.h>
// <<<<<<<<<<<<< myInit >>>>>>>>>>>>>>>>>>>>
class GLintPoint {
 public:
   GLint x, y;
};
int random (int m)
{
    return rand() % m;
}
void drawDot(int x, int y)
{ // draw dot at integer point (x, y)
gBegin(GL_POINTS);
 glVertex2i(x, y);
 glEnd();
}
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);
}
// <<<<<<<<<<<<< myDisplay >>>>>>>>>>>>>>>>>>>
void Sierpinski(void)
{
GLintPoint T[3]= {{10,10},{300,30},{200, 300}};
int index = random(3); // 0, 1, or 2 equally likely
GLintPoint point = T[index]; // initial point
drawDot(point.x, point.y); // draw initial point
for(int i = 0; i < 1000; i++) // draw 1000 dots
{
    index = random(3);
    point.x = (point.x + T[index].x) / 2;
    point.y = (point.y + T[index].y) / 2;
    drawDot(point.x,point.y);
}
glFlush();

void 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(Sierpinski); // register redraw function
    myInit();
    glutMainLoop(); // go into a perpetual loop
}

Tasks:
1. Practice different colours, polygon, lines quads etc.
2. Draw tic-tac-toe board of different widths (a,b) separately.

![](image)

a). thin lines  b). thick lines

Hint:

```c
 glBegin(GL_LINES);
 glVertex2i(10, 20); // first horizontal line
```
OpenGL provides tools for setting the attributes of lines. A line’s color is set using `glColor3f()`. Width is set using `glLineWidth(4.0)` default thickness is 1.0.