

Chapter 7

Introduction to OpenGL

Credit hours:



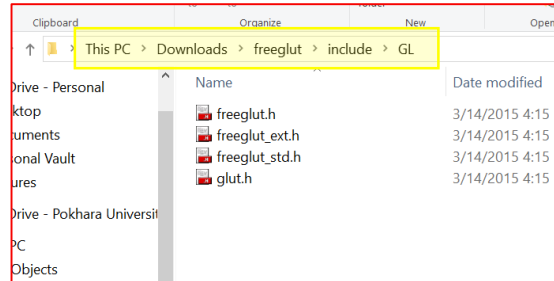
Contents

- 7.1 Overview and Basic Architecture of OpenGL
- 7.2 GL Related Libraries
 - 7.2.1 Drawing Basic Output Primitives and Polygons
 - 7.2.2 Call Back Functions and Input Handling
 - 7.2.3 Basic Transformations
 - 7.2.4 Projections and Lighting



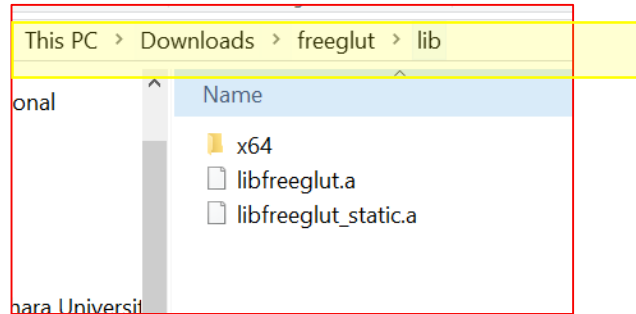
Initial setup with GLUT

- Copy these files from specified source to specified destination:



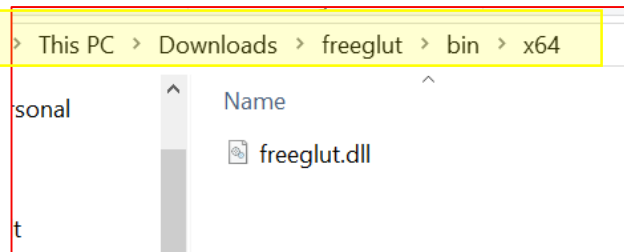
This PC > Local Disk (C:) > Program Files (x86) > Dev-Cpp > MinGW64 > x86_64-w64-mingw32 > include > GL

Name	Date modified	Type	Size
freeglut.h	3/14/2015 4:15 PM	C Header File	1 KB
freeglut_ext.h	3/14/2015 4:15 PM	C Header File	11 KB
freeglut_std.h	3/14/2015 4:15 PM	C Header File	28 KB
gl.h	9/24/2013 4:07 AM	C Header File	46 KB
glaux.h	9/24/2013 4:07 AM	C Header File	7 KB
glcorearb.h	9/24/2013 4:07 AM	C Header File	207 KB



This PC > Local Disk (C:) > Program Files (x86) > Dev-Cpp > MinGW64 > x86_64-w64-mingw32 > lib

Name	Date modified	Type	Size
libfreeglut.a	3/14/2015 4:26 PM	A File	103 KB
libfreeglut_static.a	3/14/2015 4:27 PM	A File	482 KB
libfsusd.a	9/24/2013 4:08 AM	A File	4 KB
libftpctrs2.a	9/24/2013 4:08 AM	A File	4 KB
libftomib.a	9/24/2013 4:08 AM	A File	4 KB



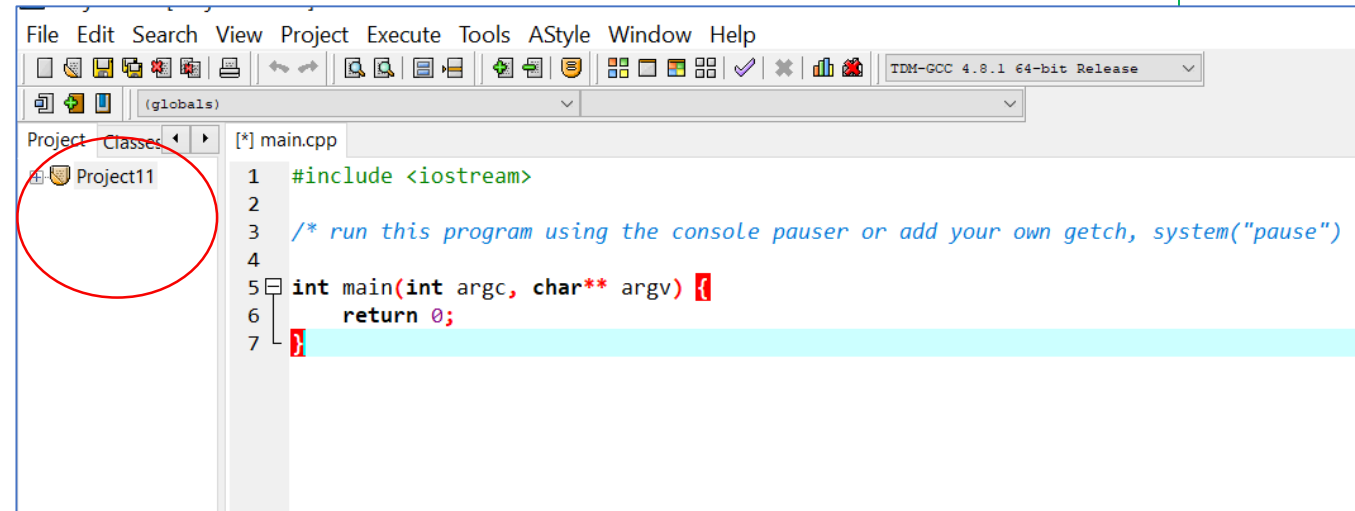
This PC > Local Disk (C:) > Windows > System32

Name
freeglut.dll
frprov.dll
fsavailux.exe

- For setting Free GLUT in dev C++: <https://www.youtube.com/watch?v=f-ppLnLYJYo>

Creating new files in DevC++ for OpenGL programs

- Open **File** tab → New → Project → Console Application
 - Put Project Name
- Right Click Project and inside Project Options → Parameters,
 - Add these linkers and press OK
 - lopengl32
 - lfreeglut
 - lglu32
- Then the program in the main.cpp file



7.1 Introduction to OpenGL

- OpenGL, or Open Graphics Library, is **not a programming language** itself but rather a cross-platform API (Application Programming Interface) designed for rendering 2D and 3D graphics.
- It **provides a set of functions** that developers can use to interact with a computer's GPU (Graphics Processing Unit), allowing for efficient rendering of images and graphics.
- Developed by the Khronos Group, OpenGL is widely used in computer graphics for applications such as video games, simulations, and visualizations.
- It **serves as a bridge between the application and the graphics hardware**, abstracting the complexities of various hardware architectures.



- OpenGL is a low-level graphics library specification. It **makes available a small set of geometric primitives** - points, lines, polygons, images, and bitmaps, to the programmer.
- OpenGL provides a set of commands that allow the specification of geometric objects in two or three dimensions, using the provided primitives, together with commands that control how these objects are rendered (drawn).
- There are several libraries that allow you to simplify your programming tasks, including the following:
 - **OpenGL Utility Library (GLU)**: contains several routines that use lower-level OpenGL commands to perform such tasks as **setting up matrices for specific viewing orientations and projections and rendering surfaces**.
 - **OpenGL Utility Toolkit (GLUT)**: is a **window-system-independent toolkit**, written by Mark Kilgard, to hide the complexities of differing window APIs



Key features of OpenGL:

1. **Cross-Platform Compatibility:**

Enables applications to run on various operating systems.

2. **Hardware Acceleration:**

Leverages GPU capabilities for efficient graphics rendering.

3. **Modular Graphics Pipeline:**

Defines stages for precise control over rendering processes.

4. **Shader Support:**

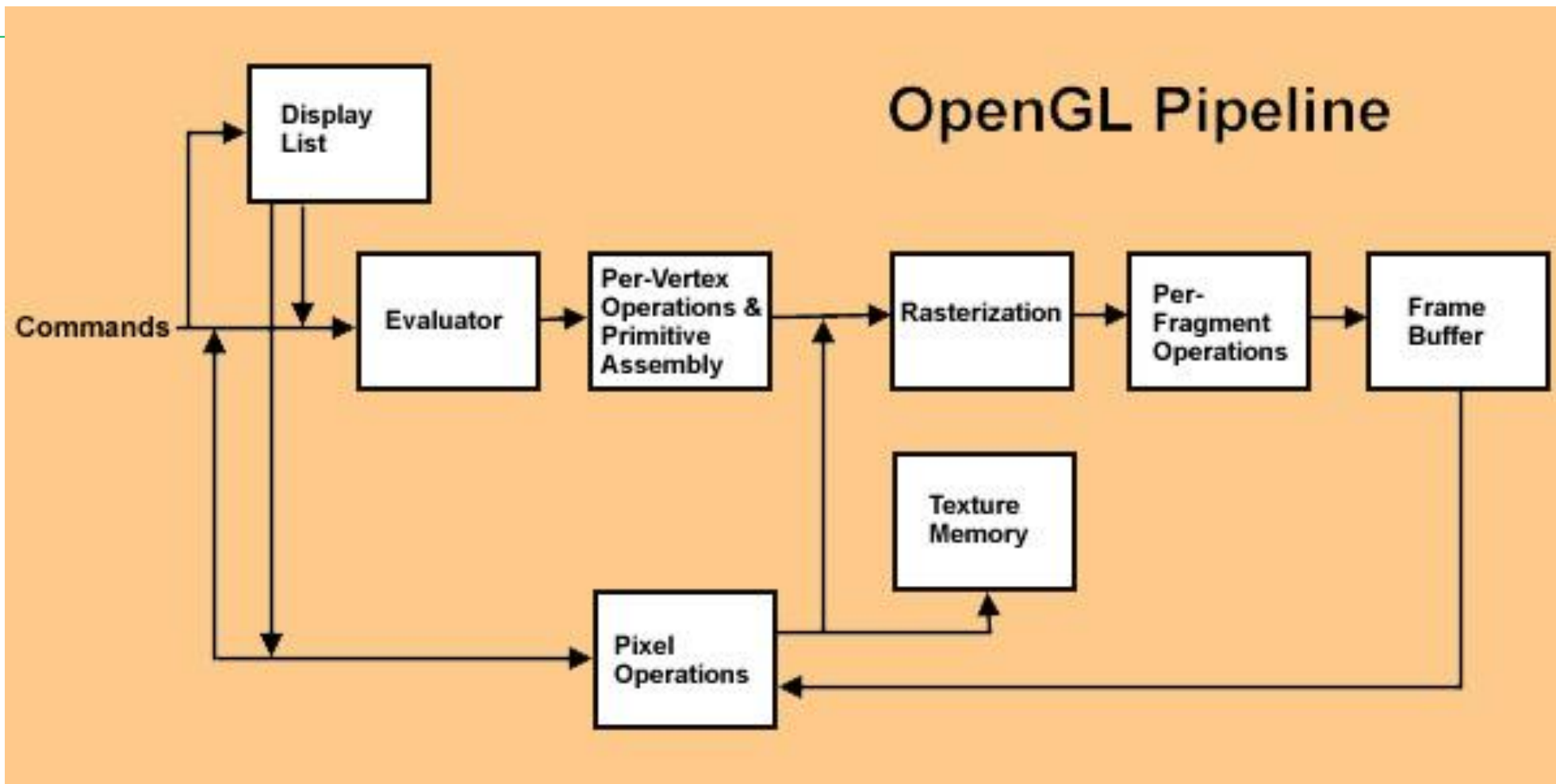
Allows programming custom shaders for visual effects.

5. **Community and Longevity:**

Active community support and ongoing development.

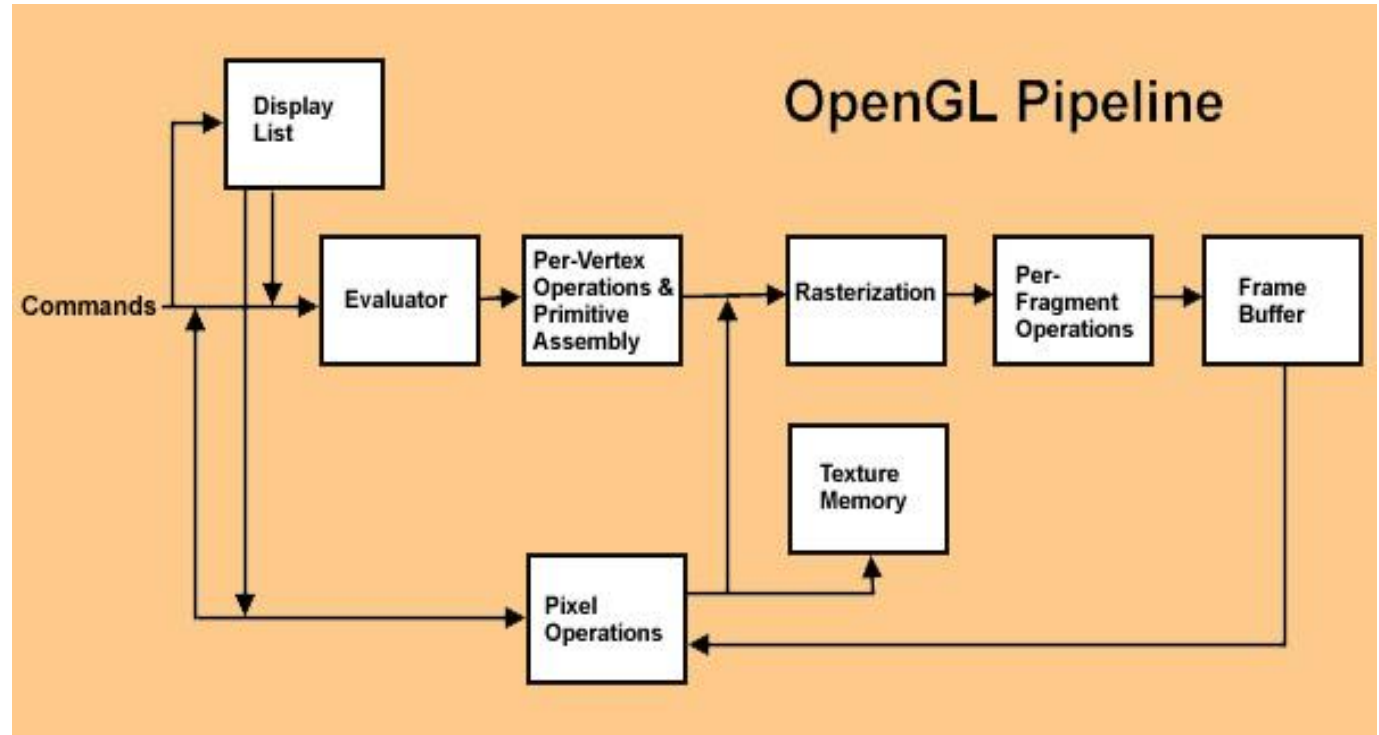


Basic Architecture of OpenGL



1. **Commands** may either be accumulated in **display lists**, or processed immediately through the pipeline. Display lists allow for greater optimization and command reuse, but not all commands can be put in display lists.

2. The first stage in the pipeline is the **evaluator**. This stage effectively takes any polynomial evaluator commands and evaluates them into their corresponding vertex and attribute commands.



3. The second stage is the **per-vertex operations**, including transformations, lighting, primitive assembly, clipping, projection, and viewport mapping.

4. The third stage is **rasterization**. This stage produces fragments, which are series of framebuffer addresses and values, from the viewport-mapped primitives as well as bitmaps and pixel rectangles.

5. The fourth stage is the **per-fragment operations**. Before fragments go to the framebuffer, they may be subjected to a series of conditional tests and modifications, such as blending or z-buffering.

6. Parts of the **framebuffer** may be fed back into the pipeline as pixel rectangles. Texture memory may be used in the rasterization process when texture mapping is enabled.

7.2 GL Related Libraries

- There are several libraries for OpenGL.
- Some of them are:
- A number of libraries exist like:
 - OpenGL,
 - OpenGL Utility Library,
 - OpenGL Utility Toolkit and
 - OpenGL Extension to the X Window System and
 - Wrappers for X functions
- to simplify programming tasks.



7.2 GL Related Libraries

- There are several libraries for OpenGL. Some of them are:

1. Core OpenGL (GL):

- consists of hundreds of commands, which **begin with a prefix "gl"**
- The Core OpenGL models an object via a set of geometric primitives such as point, line and polygon.
- Some of the GL functions are:
 - glColor(),
 - glVertex(),
 - glTranslate()
 - glRotate()
 - glEnd();
 - glEnable();
 - glDisable();
- OpenGL defined constants begin with GL_ and use all capital letters and underscores to separate words.
 - GL_COLOR_BUFFER_BIT



7.2 GL Related Libraries

2. OpenGL Utility Library (GLU):

- **built on-top of the core OpenGL to provide important utilities.**
- The OpenGL Utility Library (GLU) contains several routines that use lower-level OpenGL commands to perform such tasks as setting up matrices for specific viewing orientations and projections, performing polygon tessellation, and rendering surfaces.
- This library is provided as part of every OpenGL implementation.
- GLU commands **start with a prefix "glu"**
- Some GLU functions or commands are:
 - `gluPersepctive();`
 - `gluLookAt();`

7.2 GL Related Libraries

3. OpenGL Utilities Toolkit (GLUT):

- The OpenGL Utility Toolkit (GLUT) is a window system independent toolkit written by Mark Kilgard to hide the complexities of differing window system APIs.
- OpenGL contains only rendering commands and no commands for opening windows or reading events from keyboard or mouse.
- GLUT provides several routines for opening windows, detecting input and creating complicated 3D objects like sphere, torus and teapot.
- GLUT is an auxiliary library that makes easy to show the output of OpenGL application.
- It handles window creation, OS system calls like Mouse buttons, movement, keyboard, Callbacks, etc.
- GLUT commands **start with a prefix of "glut"**
- Some of the GLUT routines are:
 - `glutCreatewindow(),`
 - `glutMouseFunc`
 - `glutInit(&argc, argv);`
 - `glutDisplayFunc();`
 - `glutMainLoop();`



OpenGL Program Structure

- The basic OpenGL programs have a similar structure as follows:

main ()

- Define the callback functions
- Opens one or more windows with required properties.
- Enter event loops
- In main(), we should setup GL and GLUT stuff

init ()

- It initialize any OpenGL state and other program variable.
- Viewings
- Attributes

Callback function

- Initialize the registered Callback functions
- Display function
- Input and window functions



1. Drawing a Line:

```
#include <windows.h> // for MS Windows
#include <GL\glut.h> // GLUT, include glu.h and gl.h
void drawLine();
void myInit();

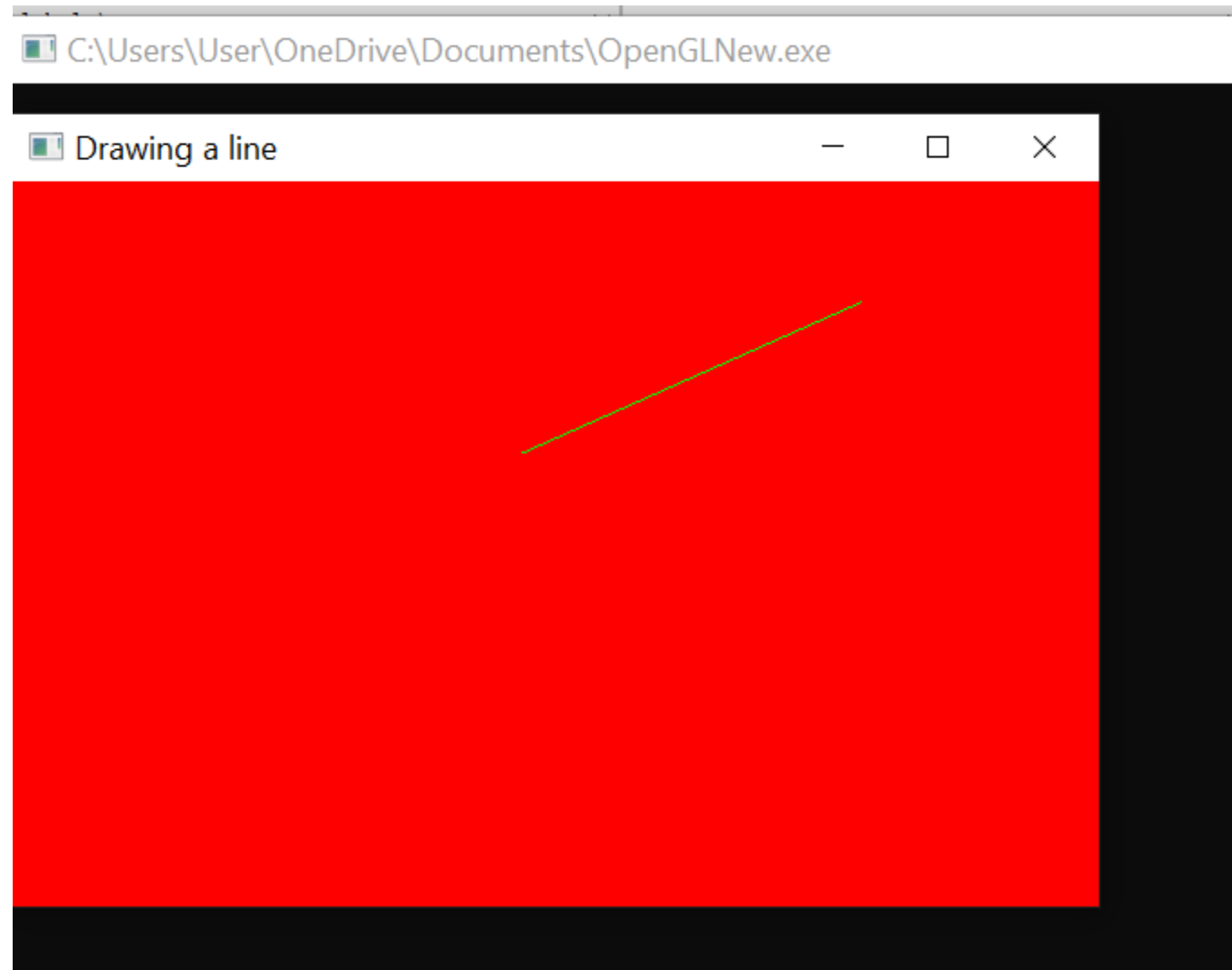
//This is main() function
int main(int argc, char** argv)
{
    glutInit(&argc, argv); //initialize the toolkit
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB); //set
the display mode
    glutInitWindowSize(600, 400); //set window size
    glutInitWindowPosition(100, 150); //set window position
    glutCreateWindow("Drawing a line"); //open the screen
window
    glutDisplayFunc(drawLine); //register redraw function
    myInit(); //additional initializations are necessary
    glutMainLoop(); //go into a perpetual loop
    return 0;
}
```

```
//This function for initialization
void myInit()
{
    glClearColor(1.0, 0.0, 0.0, 0.0); //set red background
color
    glColor3f(0.0, 1.0, 0.0); //set the drawing
color
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, 320, 0, 240);
}

//This function to draw line
void drawLine()
{
    glClear(GL_COLOR_BUFFER_BIT); //clear the screen
    glBegin(GL_LINES);
        glVertex2f(250, 200);
        glVertex2f(150, 150);
    glEnd();
    glFlush(); //send all output to display
}
```



- Output:



2. Drawing a Point

```
#include <windows.h> // for MS Windows
#include <GL\glut.h> // GLUT, include glu.h and gl.h
void drawPoint();
void myInit();

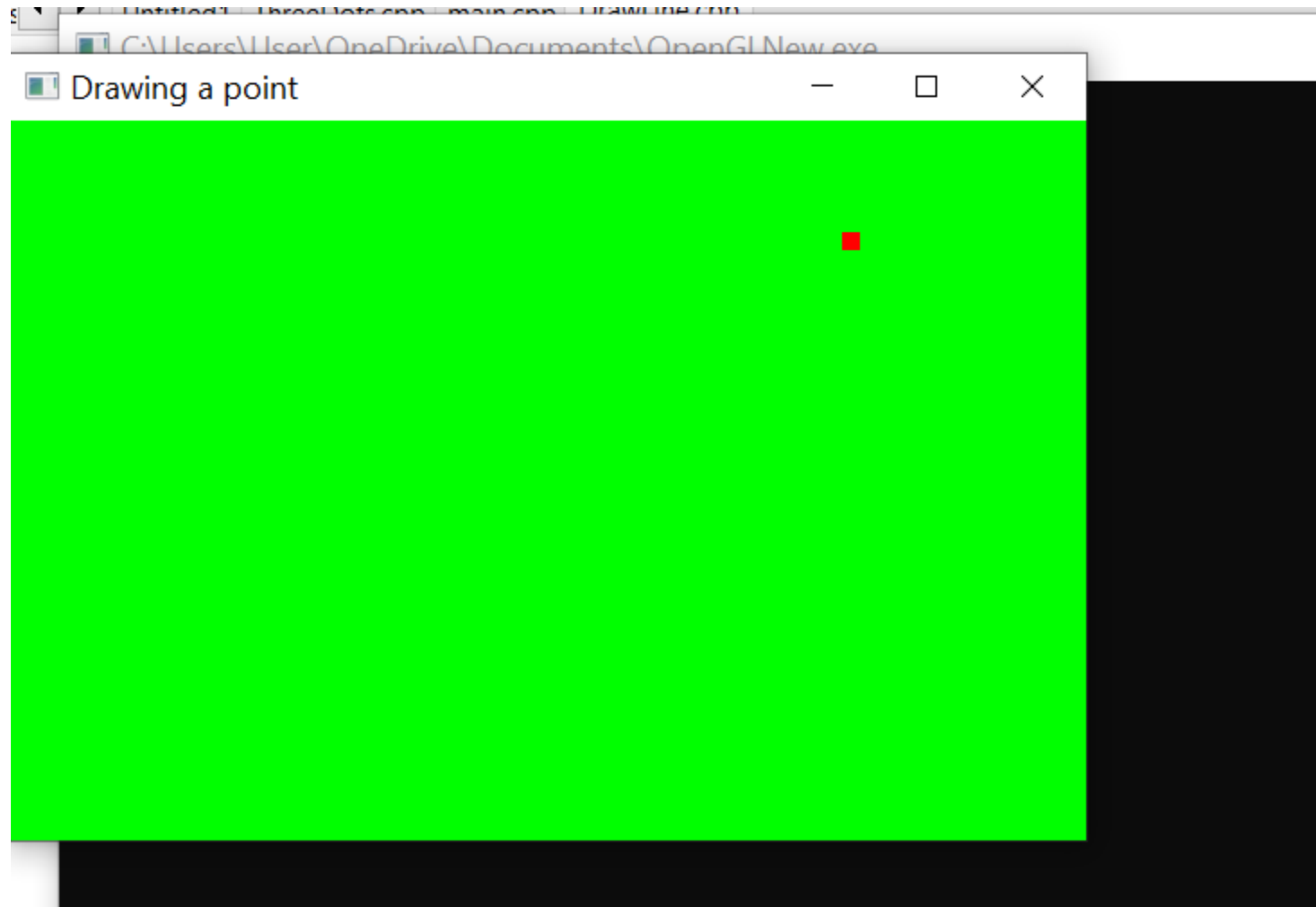
//This is main() function
int main(int argc, char** argv)
{
    glutInit(&argc, argv); //initialize the toolkit
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB); //set
the display mode
    glutInitWindowSize(600, 400); //set window size
    glutInitWindowPosition(100, 150); //set window position
    glutCreateWindow("Drawing a Point"); //open the screen
window
    glutDisplayFunc(drawPoint); //register redraw function
    myInit(); //additional initializations are necessary
    glutMainLoop(); //go into a perpetual loop
    return 0;
}
```

```
//This function for initialization
void myInit()
{
    glClearColor(0.0, 1.0, 0.0, 0.0); //set green
background color
    glColor3f(1.0, 0.0, 0.0); //set the drawing color red
    glPointSize(10.0); // adot is 10x10 pixels
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, 320, 0, 240);
}

//This function to draw line
void drawPoint()
{
    glClear(GL_COLOR_BUFFER_BIT); //clear the screen
    glBegin(GL_POINTS);
        glVertex2f(250, 200);
    glEnd();
    glFlush(); //send all output to display
}
```

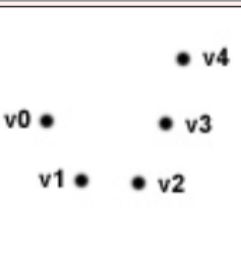
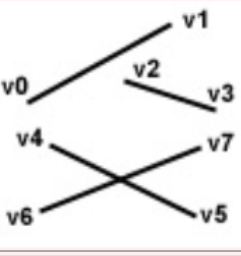
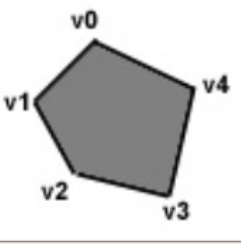


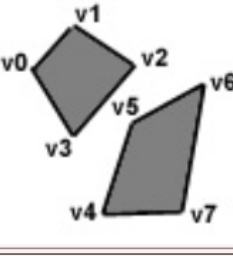
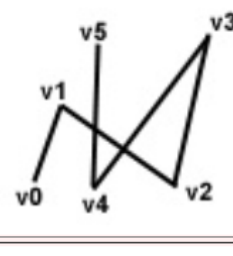
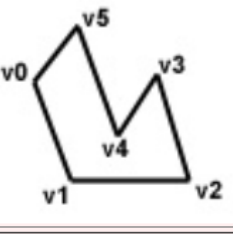
- Output:

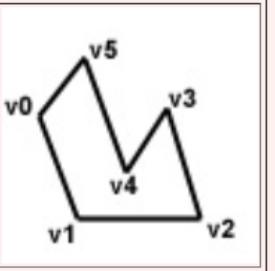
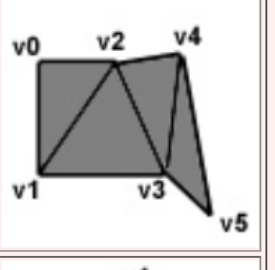
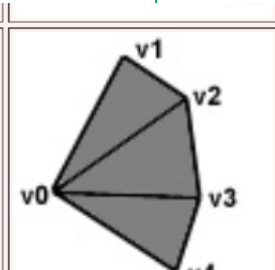
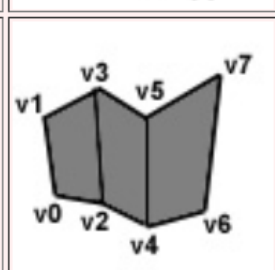


7.2.1 Drawing Basic Output Primitives and Polygons

- There are ten primitive types, as follows:

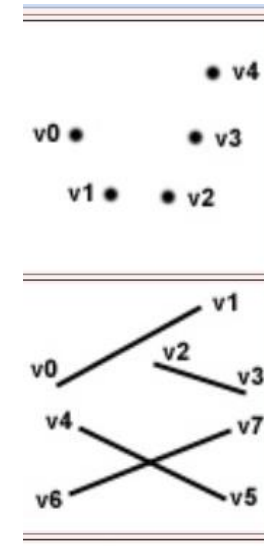
Points	individual points	
Lines	pairs of vertices interpreted as individual line segments	
Polygon	boundary of a simple, convex polygon	

Quads	quadruples of vertices interpreted as four-sided polygons	
Line Strip	series of connected line segments	
Line Loop	same as above, with a segment added between last and first vertices	

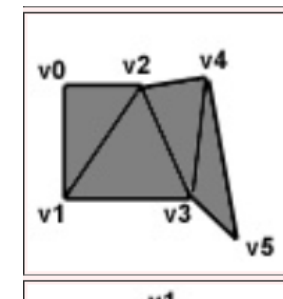
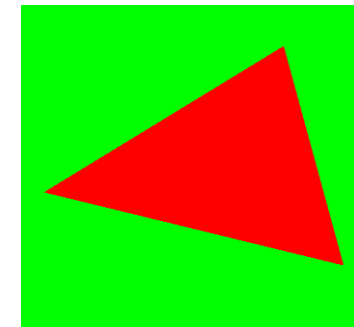
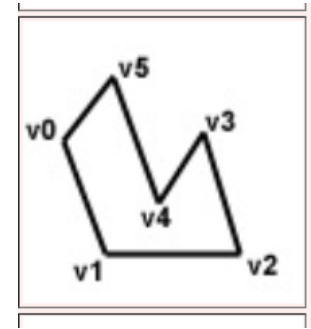
Line Loop	same as above, with a segment added between last and first vertices	
Triangle Strip	linked strip of triangles	
Triangle Fan	linked fan of triangles	
Quad Strip	linked strip of quadrilaterals	

- Those functions in a graphics package that we use to describe the various picture components are called the graphics output primitives.

Primitives	Code	Purpose
1. Points:	<code>glBegin(GL_POINTS); glVertex2f(x, y); glEnd();</code>	This code draws a single point at coordinates (x, y).
2. Lines:	<code>glBegin(GL_LINES); glVertex2f(x1, y1); glVertex2f(x2, y2); glEnd();</code>	This code draws a line segment between points (x1, y1) and (x2, y2).
3. Line Strip:	<code>glBegin(GL_LINE_STRIP); glVertex2f(x1, y1); glVertex2f(x2, y2); glVertex2f(x3, y3); glEnd();</code>	This code connects a series of points with line segments, forming a continuous line strip.

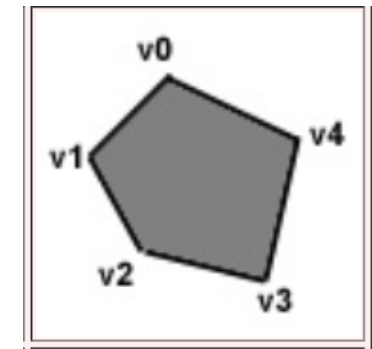
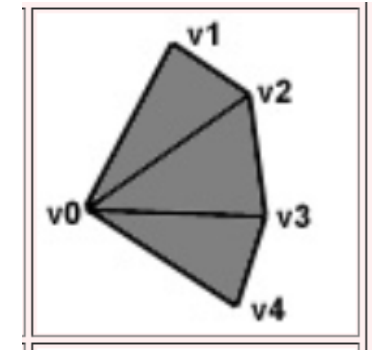


Primitives	Code	Purpose
4. Line Loop:	<pre>glBegin(GL_LINE_LOOP); glVertex2f(x1, y1); glVertex2f(x2, y2); glVertex2f(x3, y3); glEnd();</pre>	Similar to a line strip, but it automatically connects the last point to the first, forming a closed loop.
5. Triangles:	<pre>glBegin(GL_TRIANGLES); glVertex2f(x1, y1); glVertex2f(x2, y2); glVertex2f(x3, y3); glEnd();</pre>	This code draws a filled triangle with vertices at (x1, y1), (x2, y2), and (x3, y3).
6. Triangle Strip:	<pre>glBegin(GL_TRIANGLE_STRIP); glVertex2f(x1, y1); glVertex2f(x2, y2); glVertex2f(x3, y3); glVertex2f(x4, y4); glEnd();</pre>	Draws a series of connected triangles sharing edges, forming a strip.



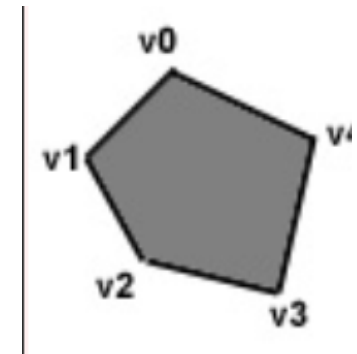
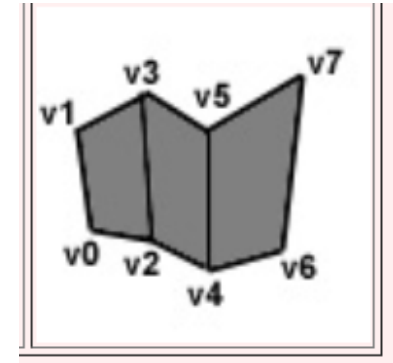
•

Primitives	Code	Purpose
7. Triangle Fan:	<pre>glBegin(GL_TRIANGLE_FAN); glVertex2f(x1, y1); glVertex2f(x2, y2); glVertex2f(x3, y3); glVertex2f(x4, y4); glEnd();</pre>	Similar to a triangle strip, but the first vertex is the center of a fan-like arrangement of triangles.
8. Quads:	<pre>glBegin(GL_QUADS); glVertex2f(x1, y1); glVertex2f(x2, y2); glVertex2f(x3, y3); glVertex2f(x4, y4); glEnd();</pre>	Draws a filled quadrilateral with vertices at (x1, y1), (x2, y2), (x3, y3), and (x4, y4).



•

Primitives	Code	Purpose
9. Quad Strip:	<pre>glBegin(GL_QUAD_STRIP); glVertex2f(x1, y1); glVertex2f(x2, y2); glVertex2f(x3, y3); glVertex2f(x4, y4); glEnd();</pre>	Connects a series of quadrilaterals sharing edges, forming a strip.
10. Polygon:	<pre>glBegin(GL_POLYGON); glVertex2f(x1, y1); glVertex2f(x2, y2); glVertex2f(x3, y3); // Add more vertices as needed glEnd();</pre>	Draws a filled polygon with an arbitrary number of vertices.



Workshop:

- **Write Programs with OpenGL to display:**
 - Triangle
 - Quadrilateral



7.2.2 Callback Functions and Input Handling

- The Callback function is user-defined function used to react on specific event like to redraw window, to react on keyboard, to handle mouse motions, etc.
- We have to register callback functions before to use it,
- Eg: when glut gets a keydown event, it uses the glutKeyboardFunc() callback routine to find out what to do with a key press.
- The glutKeyboardFunc() deals with events generated by keys which have an ASCII code.
- Some of the callback functions are:
 - glutMouseFunc() : this handles mouse events
 - glutDisplayFunc(): identifies the function to be executed. Every GLUT program must have a display callback.
 - glutPostRedisplay(): this can be used to overcome the limitation of DisplayFunc()
 - glutReshapeFunc();
 - glutMotionFunc();



Rendering Callback

- Callback function where all our drawing is done
- `glutDisplayFunc(my_display);`

```
void my_display (void )
{
    glClear( GL_COLOR_BUFFER_BIT );
    glBegin( GL_TRIANGLE );
    glVertex3fv( v[0] );
    glVertex3fv( v[1] );
    glVertex3fv( v[2] );
    glEnd();
}
```

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)
    {
        ...
    }
}
```

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;
    }
}
```

Events in OpenGL

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



7.2.3 Basic Transformations

1. Translation
2. Rotation
3. Scaling



1. Translation

1. Translation:

- Translation moves an object from one location to another.
- In OpenGL, translation is achieved by modifying the model-view matrix.
- Function used is:
 - `glTranslate{fd}(x, y, z);`
 - Eg:
 - `glTranslatef(0.0, 0.0, -6.0);`
- This move an object by the given x-, y-, z-values



3. Translate a point by (50, 50, 0)

```
#include <windows.h> // for MS Windows
#include <GL\glut.h> // GLUT, include glu.h and gl.h
void drawPolygon();
void myInit();

//This is main() function
int main(int argc, char** argv)
{
    glutInit(&argc, argv); //initialize the toolkit
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB); //set
the display mode
    glutInitWindowSize(600, 400); //set window size
    glutInitWindowPosition(100, 150); //set window position
    glutCreateWindow("Drawing a Polygon"); //open
the screen window
    glutDisplayFunc(drawPolygon); //register redraw function
    myInit(); //additional initializations are necessary
    glutMainLoop(); //go into a perpetual loop
    return 0;
}
```

```
//This function for initialization
void myInit()
{
    glClearColor(0.0, 1.0, 0.0, 0.0); //set green background color
    glColor3f(1.0, 0.0, 0.0); //set the drawing color red
    //glPointSize(10.0); // a dot is 10x10 pixels
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, 320, 0, 240);
}

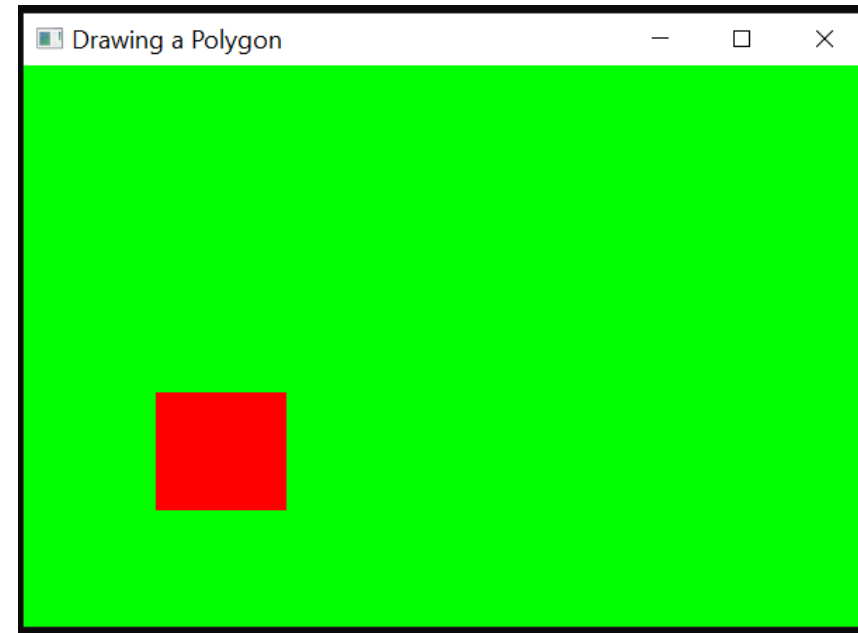
//This function to translate Polygon
void drawPolygon()
{
    glClear(GL_COLOR_BUFFER_BIT); //clear the screen
    glTranslatef(50, 50, 0); //translate the polygon to tx=50, ty=50
    glBegin(GL_POLYGON);
        glVertex2f(0, 0);
        glVertex2f(50, 0);
        glVertex2f(50, 50);
        glVertex2f(0, 50);
    glEnd();
    glFlush(); //send all output to display
}
```



- Before translation



- After translation



2. Rotation

2. Rotation:

- Rotation changes the orientation of an object.
- OpenGL provides functions to rotate objects around a specified axis.
- Function used is:
 - `glRotate{fd}(x, y, z);`
 - Eg:
 - `glRotatef(45, 0.0, 1.0);`
- This rotates an object in a counterclockwise direction about the vector (x,y,z).



4. Rotate a point by (45°, 0, 0, 1)

```
#include <windows.h> // for MS Windows
#include <GL\glut.h> // GLUT, include glu.h and gl.h
void drawPolygon();
void myInit();

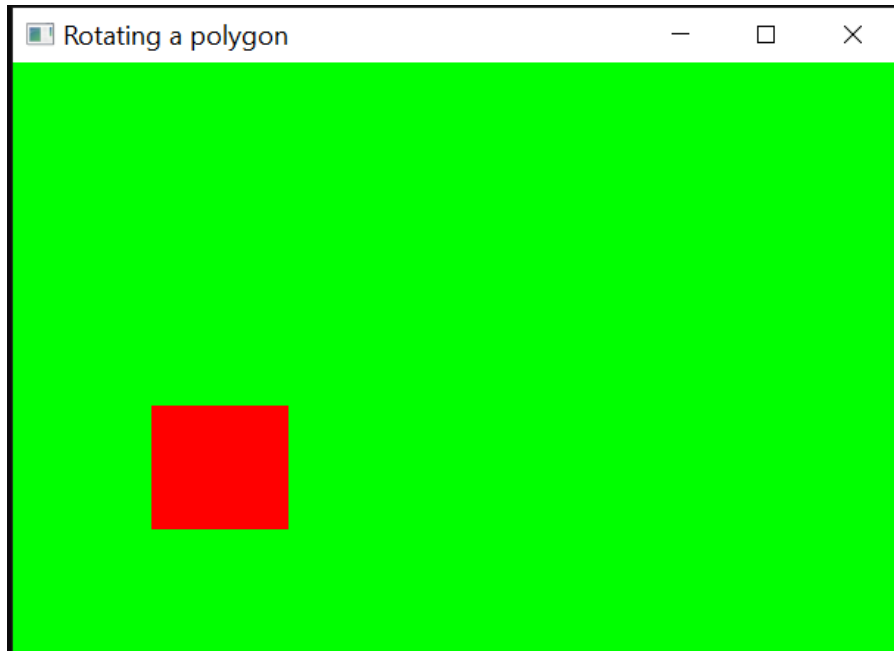
//This is main() function
int main(int argc, char** argv)
{
    glutInit(&argc, argv);           //initialize the toolkit
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB); //set
the display mode
    glutInitWindowSize(600, 400); //set window size
    glutInitWindowPosition(100, 150); //set window position
    glutCreateWindow("Rotating a Polygon"); //open
the screen window
    glutDisplayFunc(drawPolygon); //register redraw function
    myInit(); //additional initializations are necessary
    glutMainLoop(); //go into a perpetual loop
    return 0;
}
```

```
//This function for initialization
void myInit()
{
    glClearColor(0.0, 1.0, 0.0, 0.0); //set green background color
    glColor3f(1.0, 0.0, 0.0); //set the drawing color red
    //glPointSize(10.0); // a dot is 10x10 pixels
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, 320, 0, 240);
}

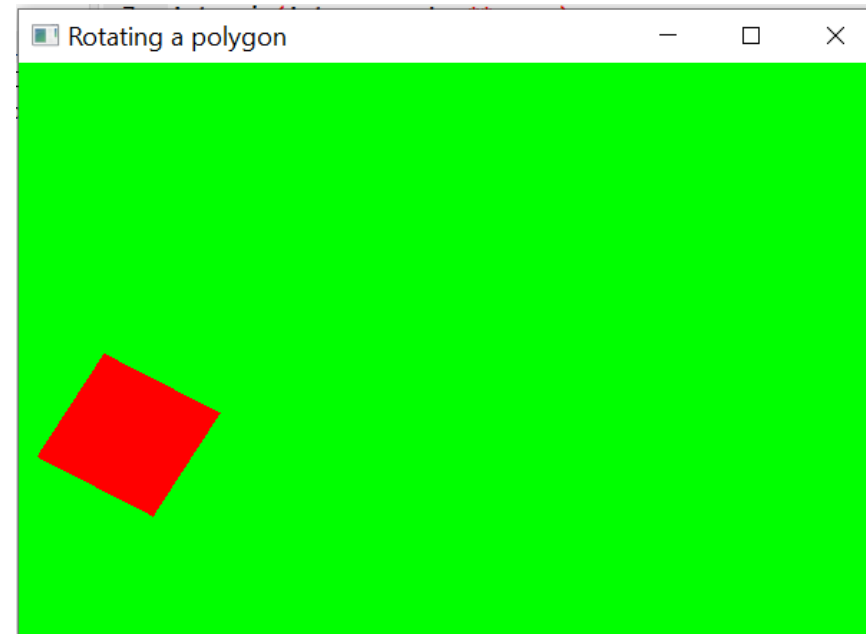
//This function to translate Polygon
void drawPolygon()
{
    glClear(GL_COLOR_BUFFER_BIT); //clear the screen
    glTranslatef(50, 50, 0); //translate the polygon to tx=50, ty=50
    glRotatef(60, 0, 0, 1); //Rotate the polygon by 45 degree
anticlockwise.
    glBegin(GL_POLYGON);
        glVertex2f(0, 0);
        glVertex2f(50, 0);
        glVertex2f(50, 50);
        glVertex2f(0, 50);
    glEnd();
    glFlush(); //send all output to display
}
```



- Before Rotation



- After Rotation



3. Scaling

3. Scaling:

- Scaling changes the size of an object.
- It is performed by modifying the model-view matrix.
- Function used is:
 - `glScalef(x, y, z);`
 - Eg:
 - Ex. `glScalef(2.0, -0.5, 1.0);`
 - This Multiply the x-, y-, z-coordinate of every point in the object by the corresponding argument x, y, or z.



5. Scaling a polygon by (2, 0.5, 1)

```
#include <windows.h> // for MS Windows
#include <GL\glut.h> // GLUT, include glu.h and gl.h
void drawPolygon();
void myInit();

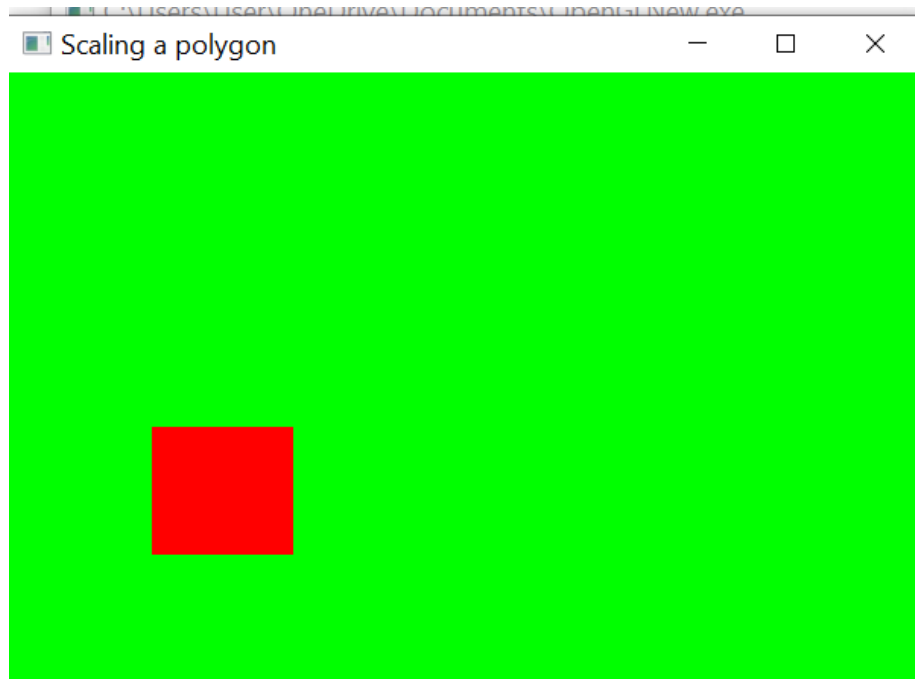
//This is main() function
int main(int argc, char** argv)
{
    glutInit(&argc, argv);           //initialize the toolkit
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB); //set
the display mode
    glutInitWindowSize(600, 400); //set window size
    glutInitWindowPosition(100, 150); //set window position
    glutCreateWindow("Scaling a Polygon"); //open
the screen window
    glutDisplayFunc(drawPolygon); //register redraw function
    myInit(); //additional initializations are necessary
    glutMainLoop(); //go into a perpetual loop
    return 0;
}
```

```
//This function for initialization
void myInit()
{
    glClearColor(0.0, 1.0, 0.0, 0.0); //set green background color
    glColor3f(1.0, 0.0, 0.0); //set the drawing color red
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, 320, 0, 240);
}

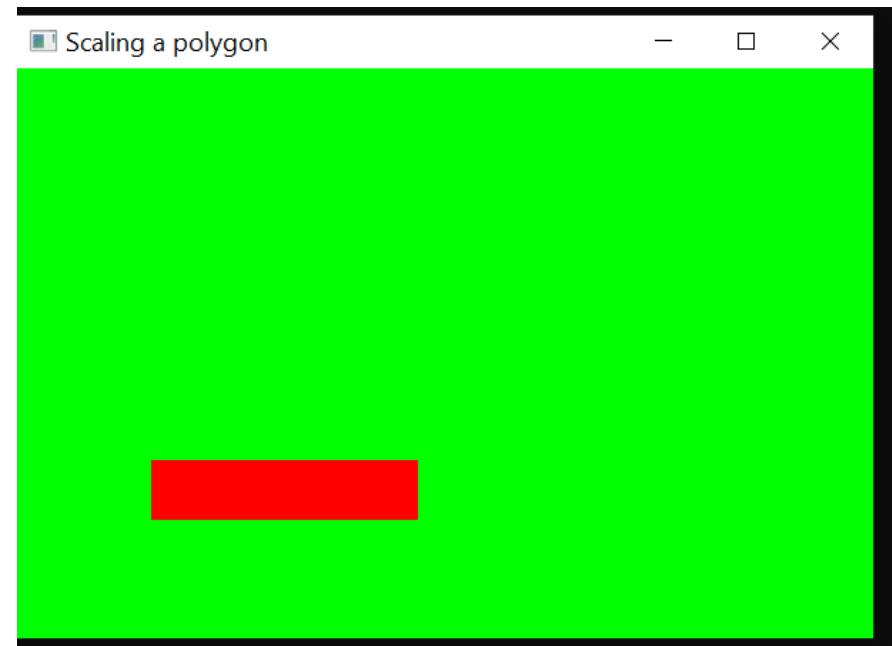
//This function to translate Polygon
void drawPolygon()
{
    glClear(GL_COLOR_BUFFER_BIT); //clear the screen
    glTranslatef(50, 50, 0); //translate the polygon to tx=50, ty=50
    glScalef(2.0f, 0.5f, 1.0f); // Scale by a factor of 2 in the X-axis
and 0.5 in the Y-axis
    glBegin(GL_POLYGON);
        glVertex2f(0, 0);
        glVertex2f(50, 0);
        glVertex2f(50, 50);
        glVertex2f(0, 50);
    glEnd();
    glFlush(); //send all output to display
}
```



- Before Scaling



- After Scaling



7.2.4 Projections and Lighting

- Projection:
 - Orthographic projection
 - Specular projection
- Lighting:
 - Ambient
 - Diffuse
 - Specular

Orthographic Projections

- Orthographic, or parallel, projections consist of those that involve no perspective correction.
 - There is no adjustment for distance from the camera made in these projections, meaning objects on the screen will appear the same size no matter how close or far away they are.
- Traditionally this type of projection was included in OpenGL for uses in CAD, or Computer Aided Design.
- Some uses of orthographic projections are making 2D games, or for creating isometric games.
- To setup this type of projection we use the OpenGL provided `glOrtho()` function.
 - **`glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far);`**
- left and right specify the x-coordinate clipping planes, bottom and top specify the y-coordinate clipping planes, and near and far specify the distance to the z-coordinate clipping planes. Together these coordinates provide a box shaped viewing volume.
- Since orthographic projections are commonly used in 2D scenes the Utility Library provides an additional routine to set them up for scenes that won't be using the z-coordinate.
 - **`gluOrtho2D(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top);`**



Perspective Projections

- Although orthographic projections can be interesting, perspective projections create more realistic looking scenes, so that's what you will most likely be using most often. In perspective projections, as an object gets farther from the viewer it will appear smaller on the screen- an effect often referred to as foreshortening. The viewing volume for a perspective projection is a frustum, which looks like a pyramid with the top cut off, with the narrow end toward the user.

There is a few different ways you can setup the view frustum, and thus the perspective projection. The first we will look at is as follows:

```
void glFrustum(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far);
```

- Using glFrustum enables you to specify an asymmetrical frustum, which can be very useful in some instances, but isn't what you typically want to do. For a different solution we again turn to the Utility Library:
- ```
void gluPerspective(GLdouble fovy, GLdouble aspect, GLdouble near, GLdouble far);
```



# Lighting

- In OpenGL, to enable or disable lighting, users need to call **glEnable(GL\_LIGHTING)** or **glDisable(GL\_LIGHTING)**.
- When lighting is enabled, colors that are specified by **glColor\*()** will become invalid.
- Light position is set using **GL\_POSITION**. Light components are set using **GL\_AMBIENT**, **GL\_DIFFUSE**.





# Reference for OpenGL learning:

- [https://www3.ntu.edu.sg/home/ehchua/programming/opengl/cg\\_introduction.html#zz-1](https://www3.ntu.edu.sg/home/ehchua/programming/opengl/cg_introduction.html#zz-1).
- [https://genuinenotes.com/wp-content/uploads/2020/03/UNIT\\_10-OpenGL.pdf](https://genuinenotes.com/wp-content/uploads/2020/03/UNIT_10-OpenGL.pdf)
- <https://slideplayer.com/slide/6929111/>
- <https://learnopengl.com/Lighting/Basic-Lighting>

**End of Chapter**