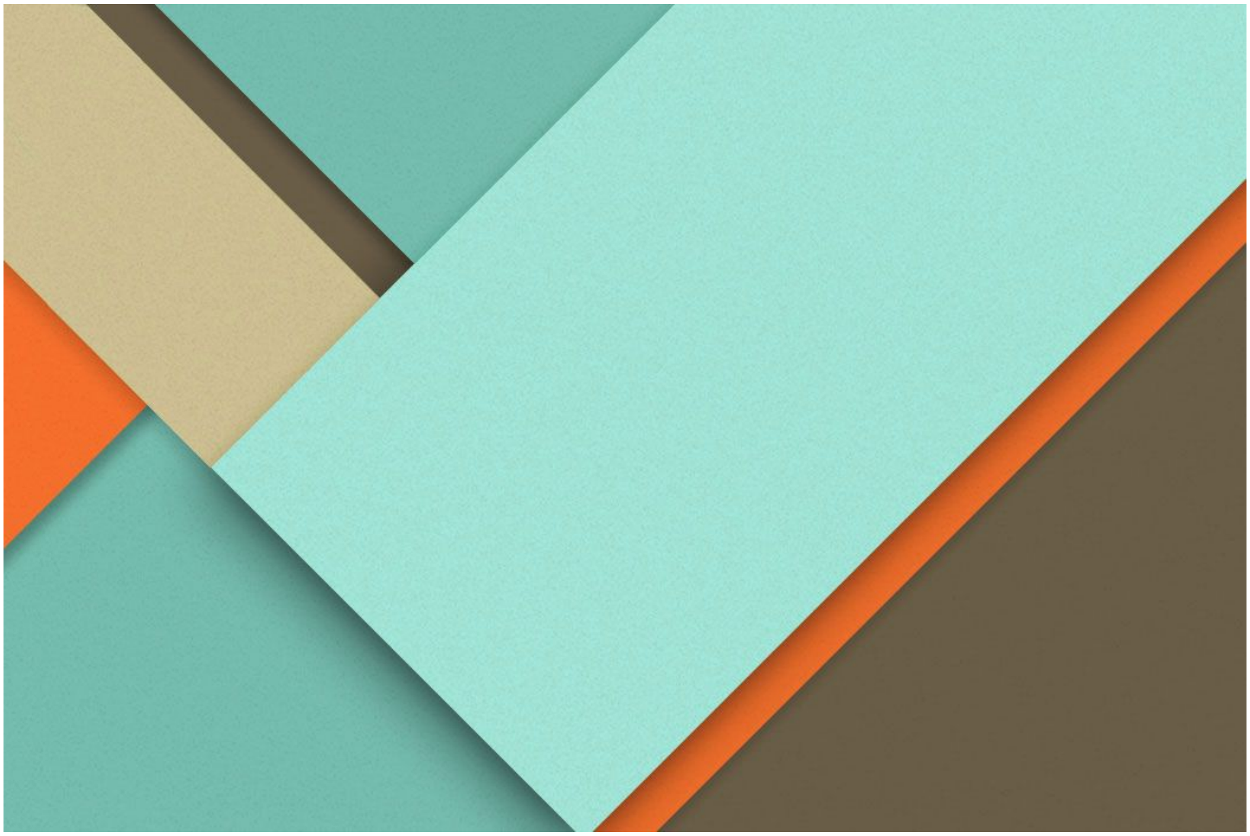


Glut Installation on ubuntu and Tutorial

Rajan Saha Raju
2015331010

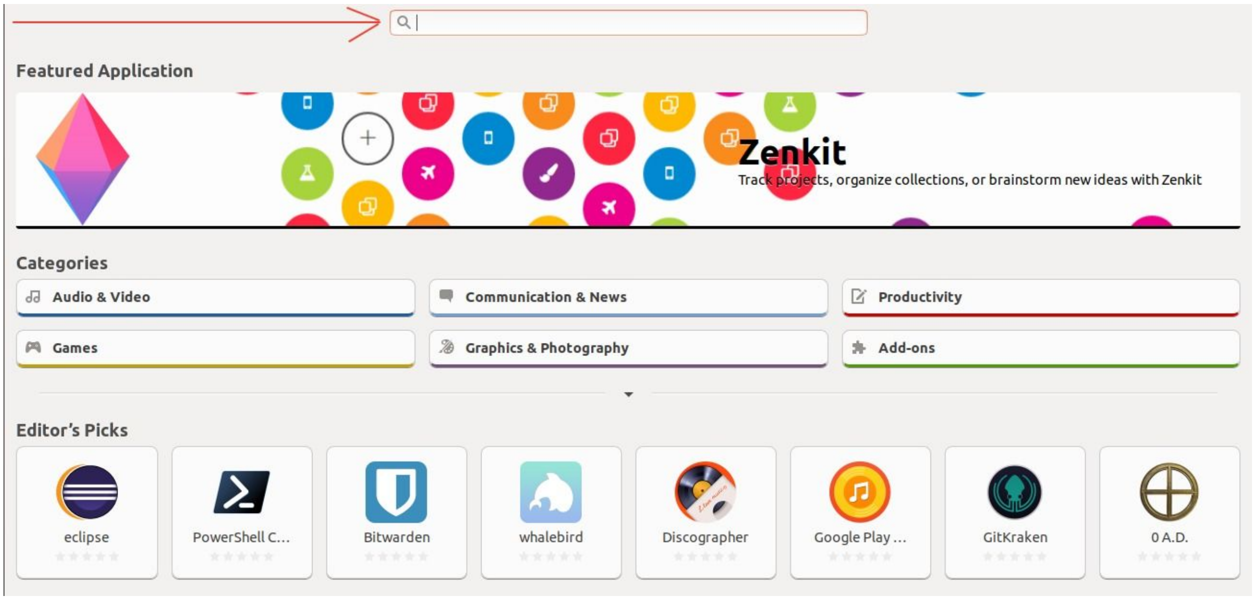
Text Editor Installation

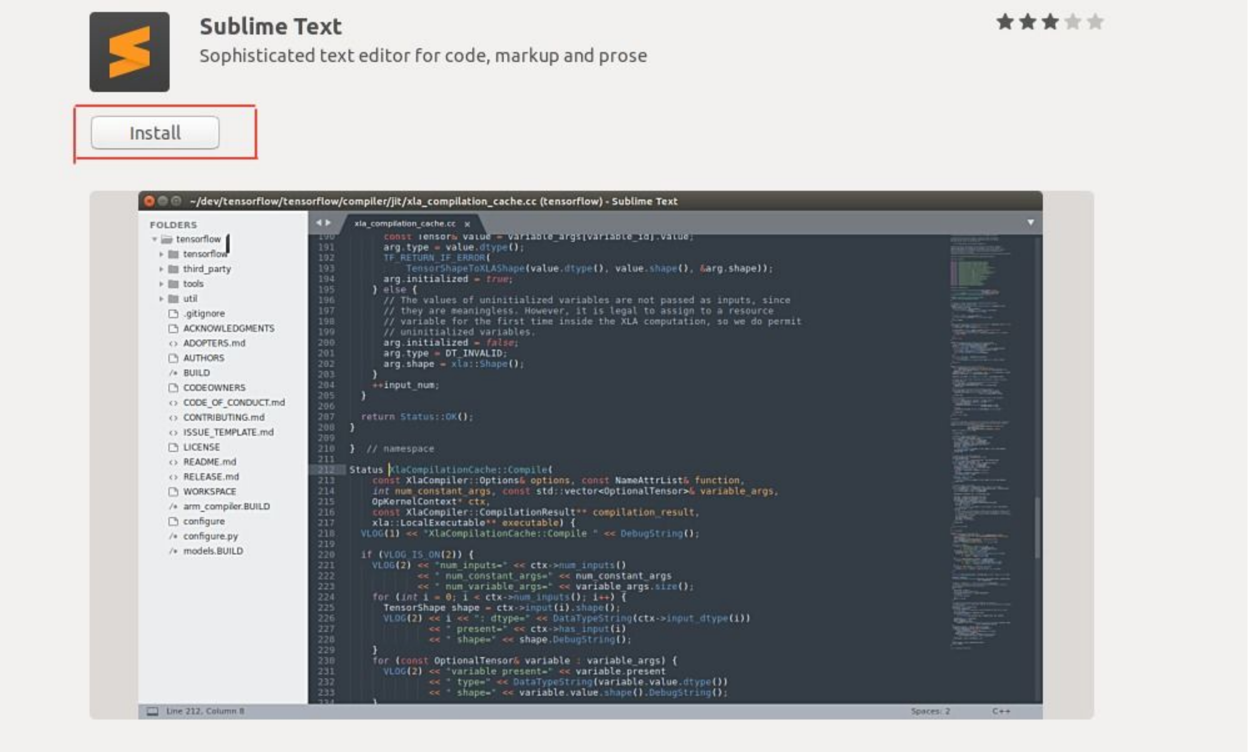
Step 1 :

We need just a text editor like Sublime Text,Atom etc.Here,I'm showing how to install Sublime text.

We'll go to ubuntu software center and search Sublime Text in search bar.After finding Sublime Text, we will click in install button.This will complete installing Sublime Text.

Search Bar :





```
        const Tensor& value = variable_args[variable_id].value;
        arg.type = value.dtype();
        TF_RETURN_IF_ERROR(
            TensorShapeToXLAShape(value.dtype(), value.shape(), &arg.shape));
        arg.initialized = true;
      } else {
        // The values of uninitialized variables are not passed as inputs, since
        // they are meaningless. However, it is legal to assign to a resource
        // variable for the first time inside the XLA computation, so we do permit
        // uninitialized variables.
        arg.initialized = false;
        arg.type = DT_INVALID;
        arg.shape = xla::Shape();
      }
      ++input_num;
    }

    return Status::OK();
  }

}  // namespace

Status XlaCompilationCache::Compile(
    const XlaCompiler::Options& options, const NameAttrList& function,
    int num_constant_args, const std::vector<OptionalTensor>& variable_args,
    OpKernelContext* ctx,
    const XlaCompiler::CompilationResult** compilation_result,
    xla::LocalExecutable** executable) {
  VLOG(1) << "XlaCompilationCache::Compile " << DebugString();

  if (VLOG_IS_ON(2)) {
    VLOG(2) << "num_inputs=" << ctx->num_inputs()
            << " num_constant_args=" << num_constant_args
            << " num_variable_args=" << variable_args.size();
    for (int i = 0; i < ctx->num_inputs(); i++) {
      TensorShape shape = ctx->input(i).shape();
      VLOG(2) << i << ": dtype=" << DataTypeString(ctx->input_dtype(i))
              << " present=" << ctx->has_input(i)
              << " shape=" << shape.DebugString();
    }
    for (const OptionalTensor& variable : variable_args) {
      VLOG(2) << "variable present=" << variable.present
              << " type=" << DataTypeString(variable.value.dtype())
              << " shape=" << variable.value.shape().DebugString();
    }
```

Glut installation

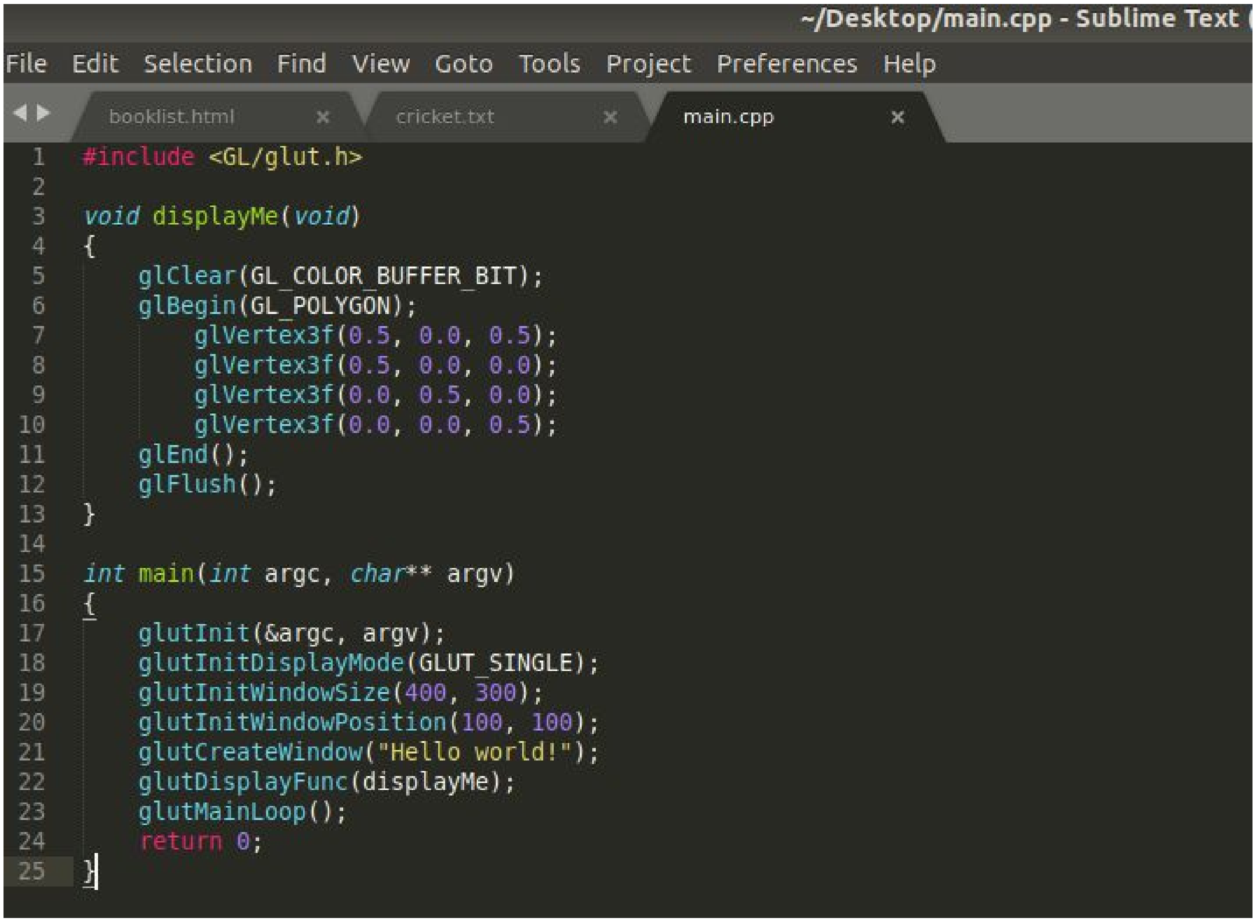
Open a Terminal : Ctrl+Alt+t

Run the following commands to install OpenGL :

```
$ sudo apt-get update
$ sudo apt-get install libglu1-mesa-dev freeglut3-dev mesa-common-dev
```

Creating a cpp file and Open in sublime :

```
$ touch filename.cpp
$ subl filename.cpp
```

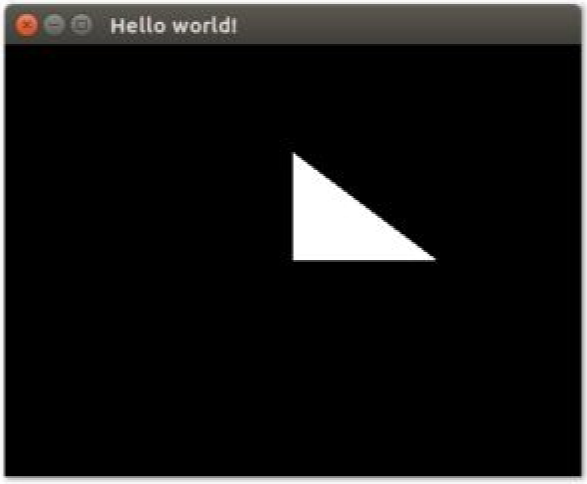


```cpp
#include <GL/glut.h>

void displayMe(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_POLYGON);
        glVertex3f(0.5, 0.0, 0.5);
        glVertex3f(0.5, 0.0, 0.0);
        glVertex3f(0.0, 0.5, 0.0);
        glVertex3f(0.0, 0.0, 0.5);
    glEnd();
    glFlush();
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE);
    glutInitWindowSize(400, 300);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Hello world!");
    glutDisplayFunc(displayMe);
    glutMainLoop();
    return 0;
}
```

Compile and run :

```
$ g++ filename.cpp -o firstOpenGlApp -lglut -lGLU -lGL
$ ./firstOpenGlApp
```

Output :



Documentation

Header file include :

```
#include <GL/glut.h>
#include <GL/glu.h>
```

glutInit :

Initializes GLUT, must be called before other GL/GLUT functions. It takes the same arguments as the `main()`.

```
int main(int argc, char** argv) {
   glutInit(&argc, argv);
}
```

glutCreateWindow :

creates a window with the given title.

```
glutCreateWindow("Graphics Window");
```

glutInitWindowSize :

Specifies the initial window width and height, in pixels.

```
glutInitWindowSize(300,400); /* width = 300 and height = 400 */
```

glutDisplayFunc :

Registers the callback function (or event handler) for handling window-paint event. The OpenGL graphic system calls back this handler when it receives a window repaint request. In the example, we register the function `display()` as the handler.

```
glutDisplayFunc(display); /*calling the display function*/
```

glutInitDisplayMode :

Sets the initial display mode.Can be set multiple mode using bitwise or.

```
glutInitDisplayMode(GLUT_RGBA|GLUT_SINGLE);
```

glutMainLoop :

Enters the infinite event-processing loop, i.e, put the OpenGL graphics system to wait for events (such as repaint), and trigger respective event handlers (such as `display()`).

```
glutMainLoop();
```

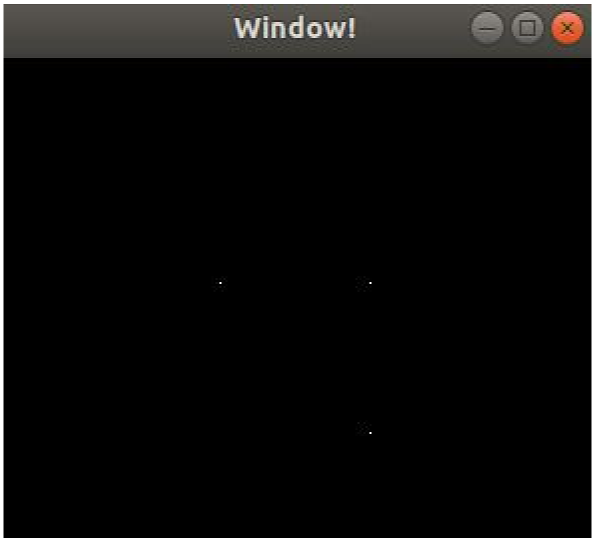
Drawing Points and Line

Points :

glVertex2f(double x,double y) :

```
void display() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glBegin(GL_POINTS);
            glVertex2f(0.5f, 0.5f);
            glVertex2f(0.5f, -0.5f);
            glVertex2f(-0.5f, 0.5f);
        glEnd();
    glutSwapBuffers();
}
```

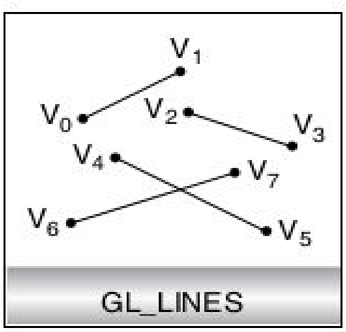
Output :

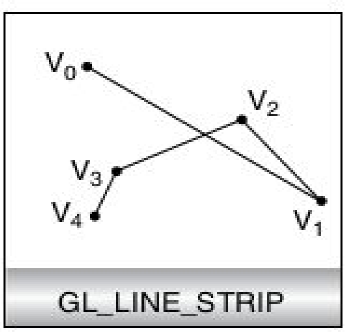


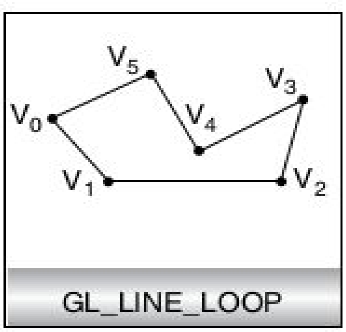
Lines :

Value	Meaning
GL_LINES	**Individual line segments**
GL_LINE_STRIP	**Draws a line segment from V_0 to V_1 , then from V_1 to V_2 , and so on, finally drawing the segment from V_{n-2} to V_{n-1} . V_x denote x th vertice.**

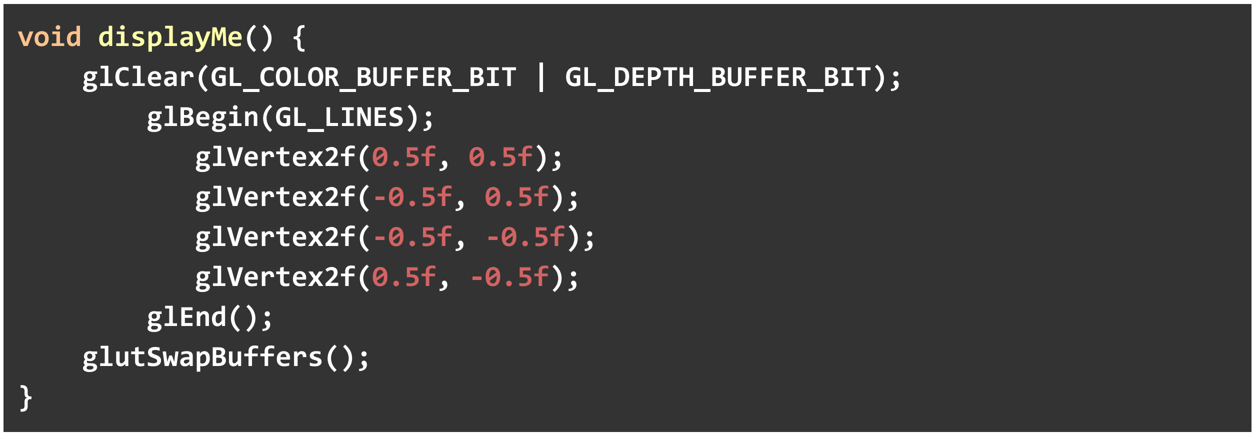
GL_LINE_LOOP	**Same as GL_LINE_STRIP, except that a final line segment is drawn from V_{n-1} to V_0 , completing a loop.**





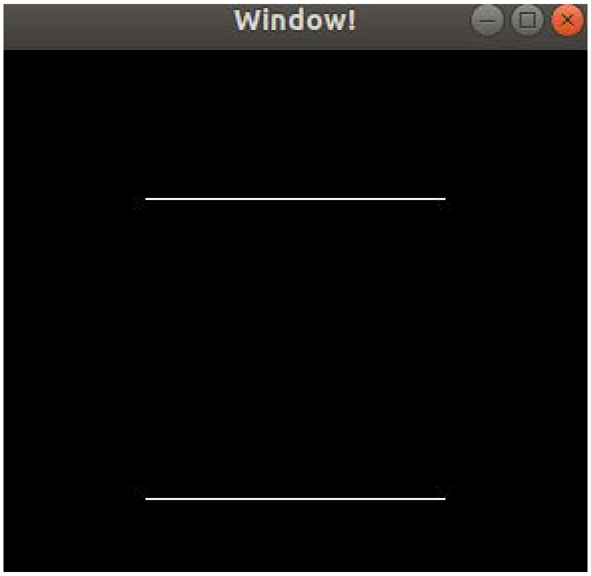


Example : GL_LINES



```
void displayMe() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glBegin(GL_LINES);
           glVertex2f(0.5f, 0.5f);
           glVertex2f(-0.5f, 0.5f);
           glVertex2f(-0.5f, -0.5f);
           glVertex2f(0.5f, -0.5f);
        glEnd();
    glutSwapBuffers();
}
```

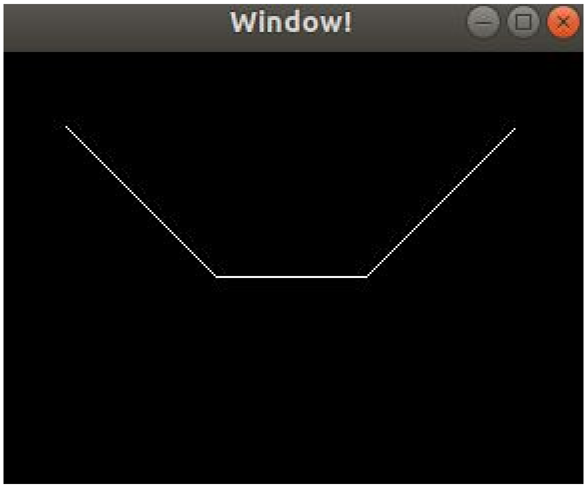
Output :



Example : GL_LINE_STRIP

```
void displayMe() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glBegin(GL_LINE_STRIP);
           glVertex2f(-0.75f, 0.75f);
           glVertex2f(-0.25f, 0.25f);
           glVertex2f(0.25f, 0.25f);
           glVertex2f(0.75f, 0.75f);
        glEnd();
    glutSwapBuffers();
}
```

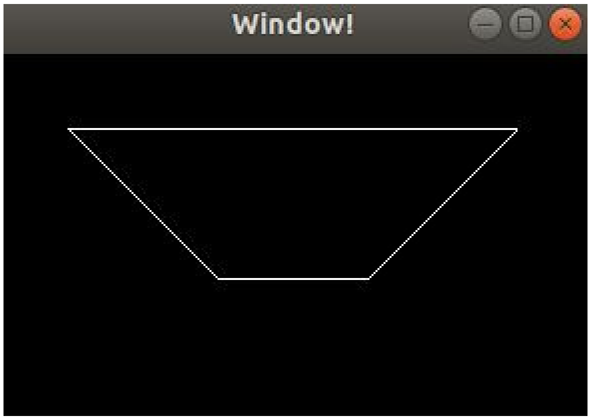
Output :



Example : GL_LINE_LOOP

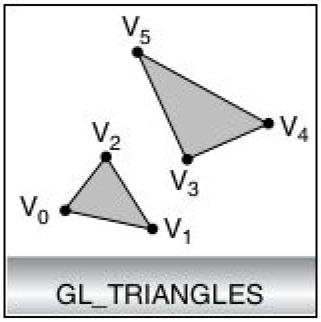
```
void displayMe() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glBegin(GL_LINE_LOOP);
           glVertex2f(-0.75f, 0.75f);
           glVertex2f(-0.25f, 0.25f);
           glVertex2f(0.25f, 0.25f);
           glVertex2f(0.75f, 0.75f);
        glEnd();
    glFlush();
}
```

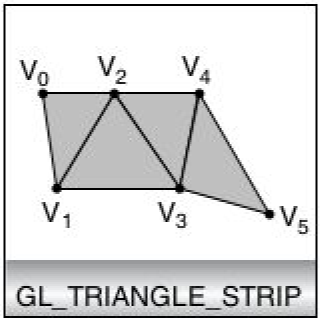
Output :

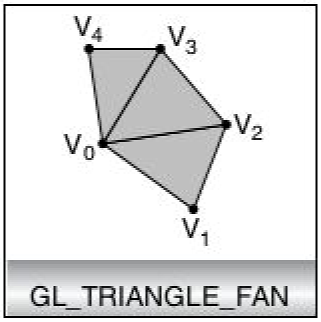


Drawing Triangle

Value	Meaning
GL_TRIANGLES	Draws a series of triangles (three-sided polygons) using vertices V_0 , V_1 , V_2 and V_3 ,V_4 , V_5 , and so on. If n isn't a multiple of 3, the final one or two vertices are ignored.
GL_TRIANGLE_STRIP	Draws a series of triangles (three-sided polygons) using vertices V_0, V_1 , V_2 , then V_2,V_1 ,V_3 (note the order), then V_3 ,V_2 ,V_4 and so on.
GL_TRIANGLE_FAN	Same as GL_TRIANGLE_STRIP, except that the vertices are V_0 , V_1 , V_2 , then V_0 , V_2 , V_3 , then V_0 ,V_3 , V_4 , and so on.



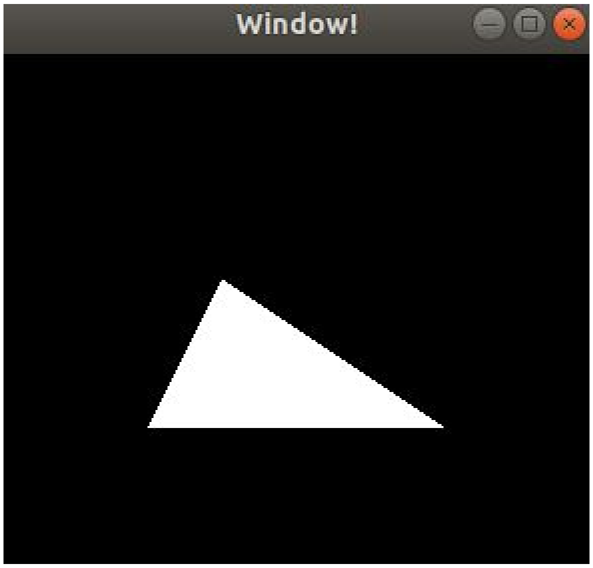




Example : **GL_TRIANGLES**

```
void displayMe(void) {
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_TRIANGLES);
               glVertex2f(-0.25f, 0.25f); // vertex 1
               glVertex2f(-0.5f, -0.25f); // vertex 2
               glVertex2f(0.5f, -0.25f); // vertex 3
    glEnd();
    glFlush();
}
```

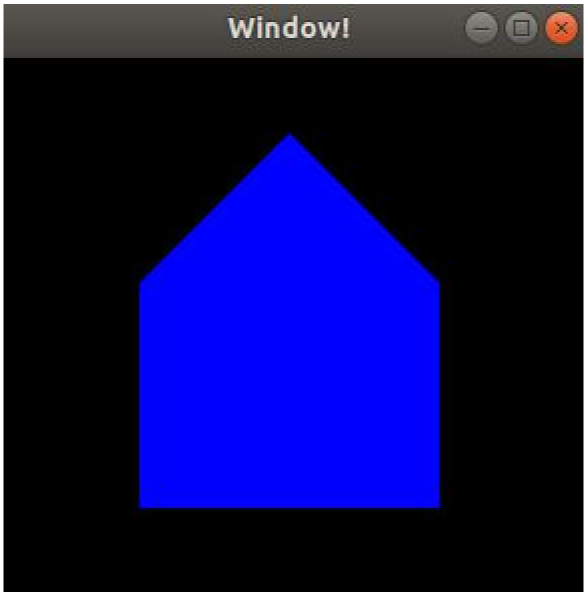
Output :



Example : **GL_TRIANGLE_STRIP**

```
void displayMe(void) {
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0f,0.0f,1.0f); //blue color
    glBegin(GL_TRIANGLE_STRIP); // draw in triangle strips
               glVertex2f(0.0f, 0.75f); // top of the roof
               glVertex2f(-0.5f, 0.25f); // left corner of the roof
               glVertex2f(0.5f, 0.25f); // right corner of the roof
               glVertex2f(-0.5f, -0.5f); // bottom left corner of the house
               glVertex2f(0.5f, -0.5f); //bottom right corner of the house
    glEnd();
    glFlush();
}
```

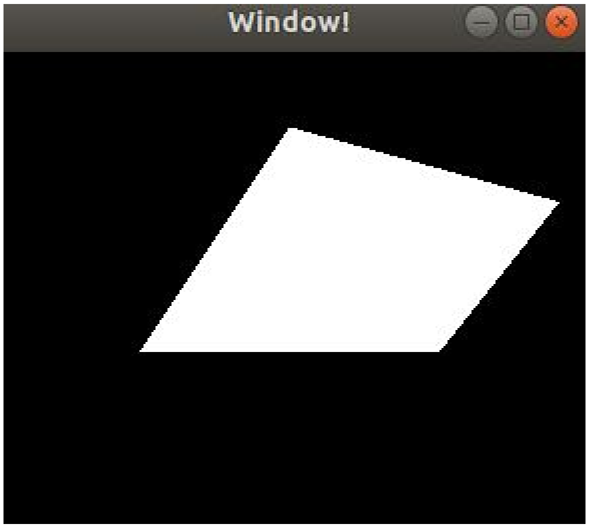
Output :



Example : **GL_TRIANGLE_FAN**

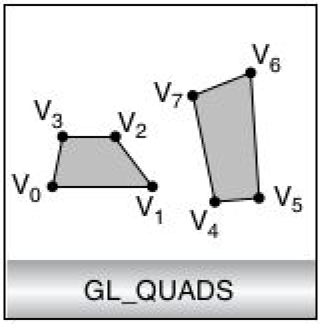
```
void displayMe(void) {
    glClear(GL_COLOR_BUFFER_BIT);

    glBegin(GL_TRIANGLE_FAN);
               glVertex2f(0.0f, 0.75f);
               glVertex2f(-0.5f, 0.0f);
               glVertex2f(0.5f, 0.0f);
               glVertex2f(0.9f, 0.5f);

    glEnd();
    glFlush();
}
```

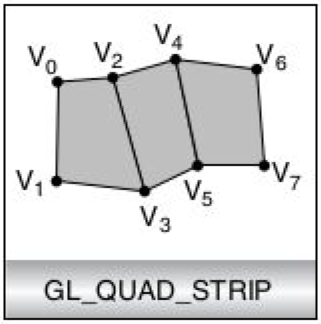
Output :



Drawing quadrilateral

Value	Meaning
GL_QUADS	**Draws a series of quadrilaterals (four-sided polygons) using vertices V_0, V_1 , V_2 , V_3 and V_4 ,V_5 ,V_6 , V_7 and so on. If n isn't a multiple of 4,the final one, two, or three vertices are ignored.**
GL_QUAD_STRIP	**Draws a series of quadrilaterals (four-sided polygons) beginning with V_0 ,V_1 , V_2 ,V_3 , then V_2 , V_3 , V_5 , V_4 and so on . n must be at least 4 before anything is drawn. If n is odd, the final vertex is ignored.**

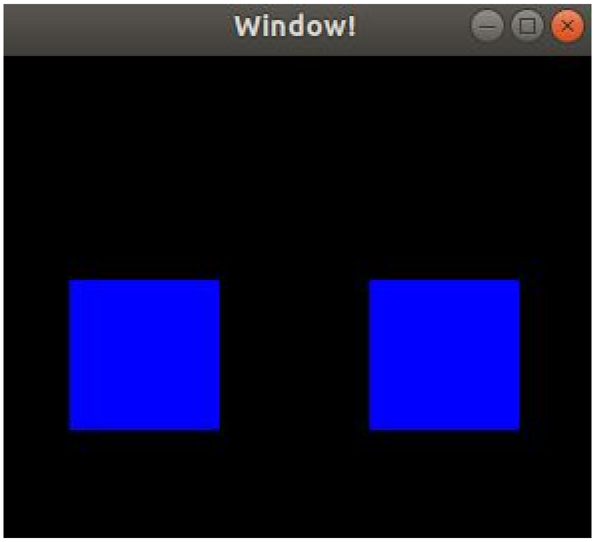




Example : **GL_QUADS**

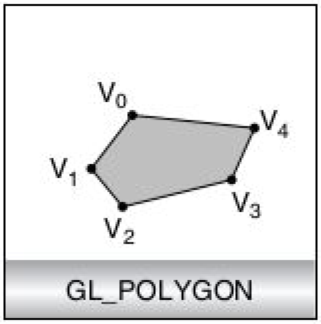
```
void displayMe(void) {
    glClear(GL_COLOR_BUFFER_BIT);

    glBegin(GL_QUADS);
               glVertex2f(-0.25f,0.25f);
               glVertex2f(-0.25f,-0.25f);
               glVertex2f(-0.75f,-0.25f);
               glVertex2f(-0.75f,0.25f);
               glVertex2f(0.25f,0.25f);
               glVertex2f(0.75f,0.25f);
               glVertex2f(0.75f,-0.25);
               glVertex2f(0.25f,-0.25f);
    glEnd();
    glFlush();
}
```

Output :



Drawing Polygon

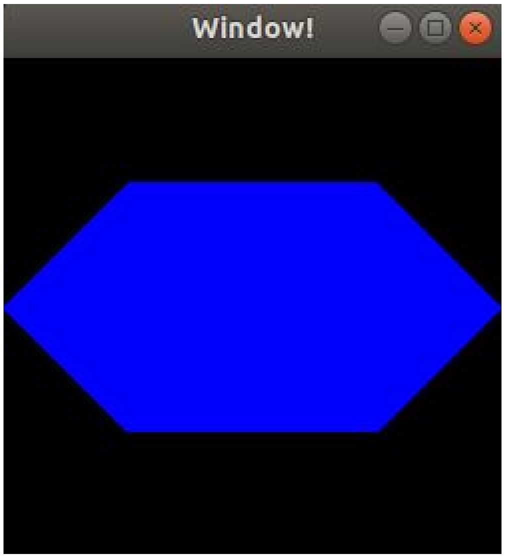
Value	Meaning
GL_POLYGON	**Draws a polygon using the points.In addition, the polygon specified must not intersect itself and must be convex. If the vertices don’t satisfy these conditions, the results are unpredictable.**



Example : **GL_POLYGON**

```
void displayMe(void) {

    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0f,0.0f,1.0f);
    glBegin(GL_POLYGON);//begin drawing of polygon
      glVertex3f(-0.5f,0.5f,0.0f);//first vertex
      glVertex3f(0.5f,0.5f,0.0f);//second vertex
      glVertex3f(1.0f,0.0f,0.0f);//third vertex
      glVertex3f(0.5f,-0.5f,0.0f);//fourth vertex
      glVertex3f(-0.5f,-0.5f,0.0f);//fifth vertex
      glVertex3f(-1.0f,0.0f,0.0f);//sixth vertex
    glEnd();//end drawing of polygon
    glFlush();
}
```

Output :

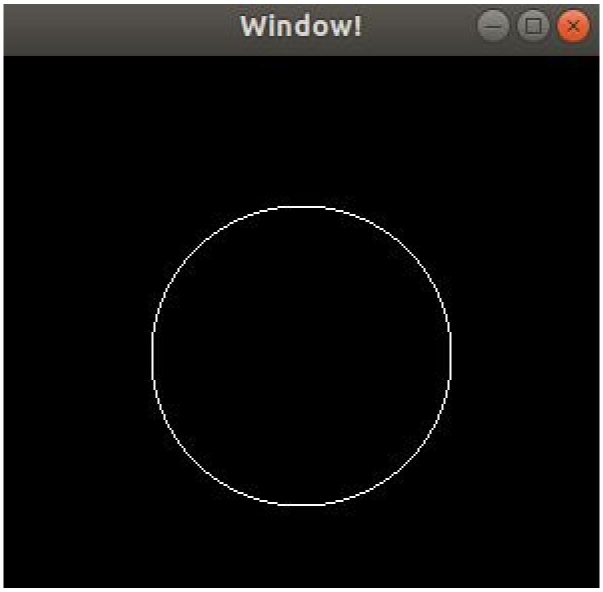


Circle Drawing

Example : Using GL_LINE_LOOP

```
void displayMe(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    GLint circle_points = 1000;
      glBegin(GL_LINE_LOOP);

        for (int i = 0; i < circle_points; i++) {
              double angle = 2*PI*i/circle_points;
              glVertex2f(0.5*cos(angle), 0.5*sin(angle));
        }
      glEnd();
    glFlush();
}
```

Output :

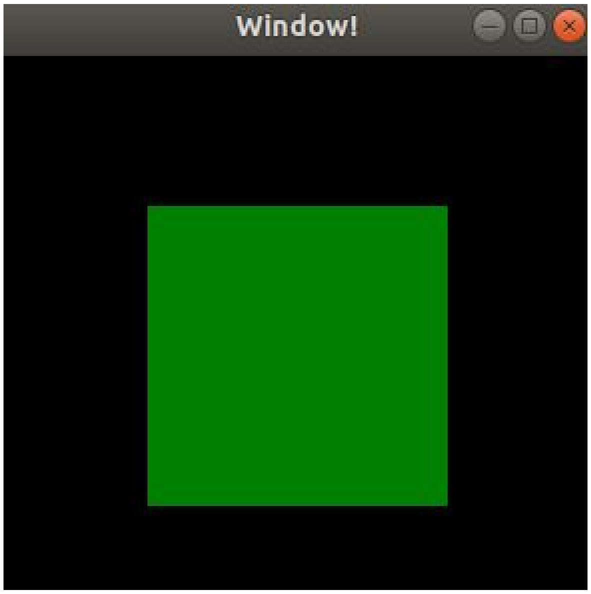


OpenGL Color

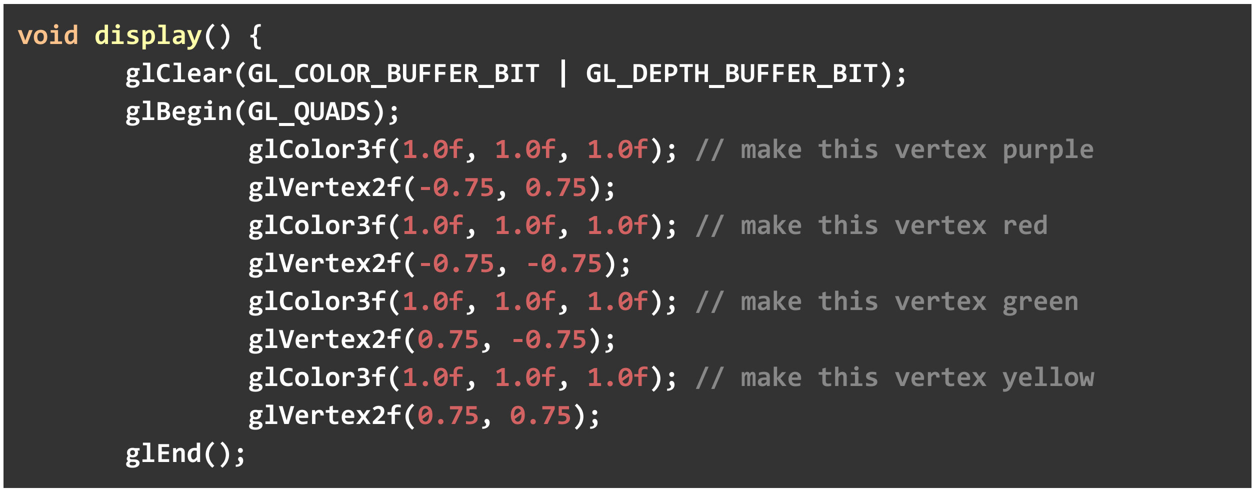
glColor3f(double R,double G,double B) :

Takes 3 arguments: the red, green and blue components of the color you want. After you use glColor3f, everything you draw will be in that color.The range of value of red,green,blue is [0.0 , 1.0].

Example :



glColor3f can be called in between glBegin and glEnd. When it is used this way, it can be used to give each vertex its own color. The resulting rectangle is then shaded with an attractive color gradient.



```
void display() {
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glBegin(GL_QUADS);
                glColor3f(1.0f, 1.0f, 1.0f); // make this vertex purple
                glVertex2f(-0.75, 0.75);
                glColor3f(1.0f, 1.0f, 1.0f); // make this vertex red
                glVertex2f(-0.75, -0.75);
                glColor3f(1.0f, 1.0f, 1.0f); // make this vertex green
                glVertex2f(0.75, -0.75);
                glColor3f(1.0f, 1.0f, 1.0f); // make this vertex yellow
                glVertex2f(0.75, 0.75);
        glEnd();
```

```
	glutSwapBuffers();
}
```

Output :