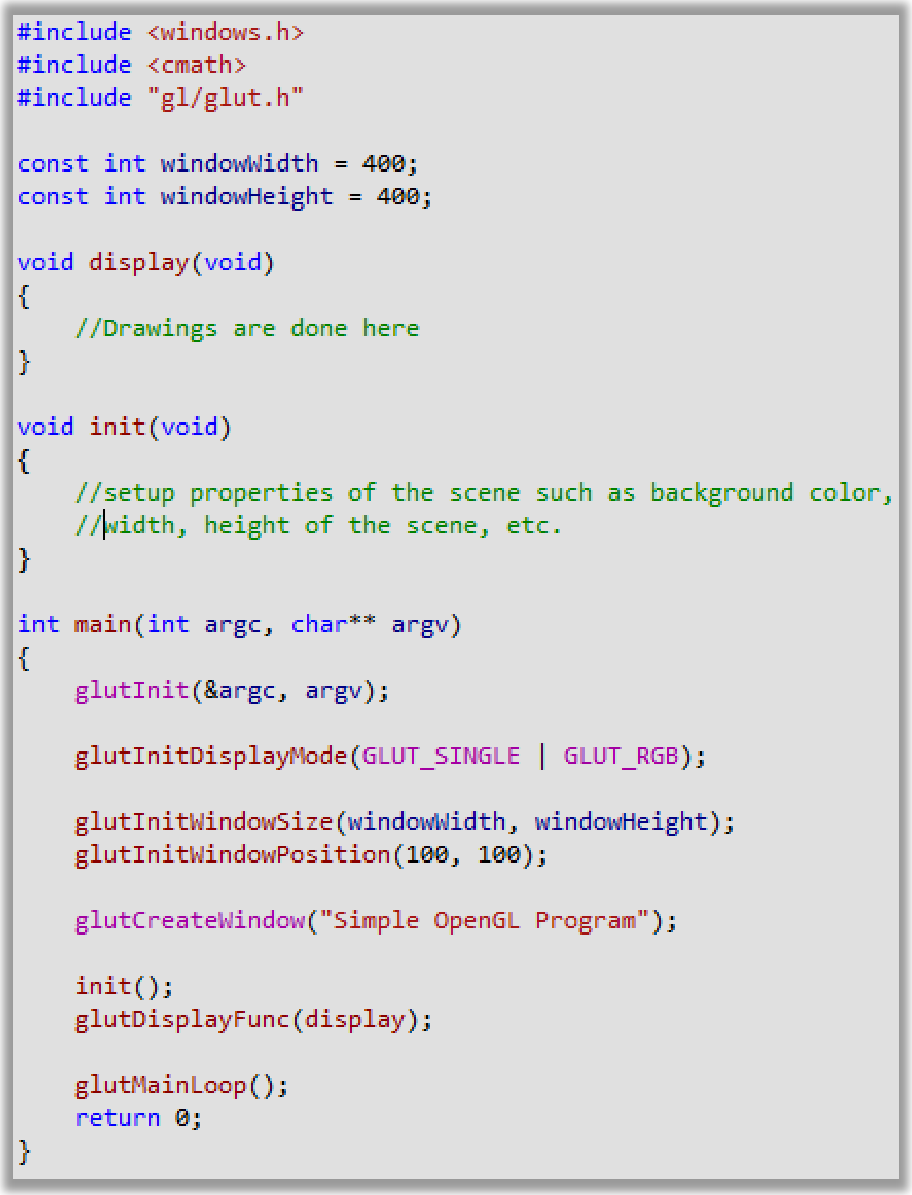
CompSci 373 Tutorial

OpenGL

What is OpenGL?

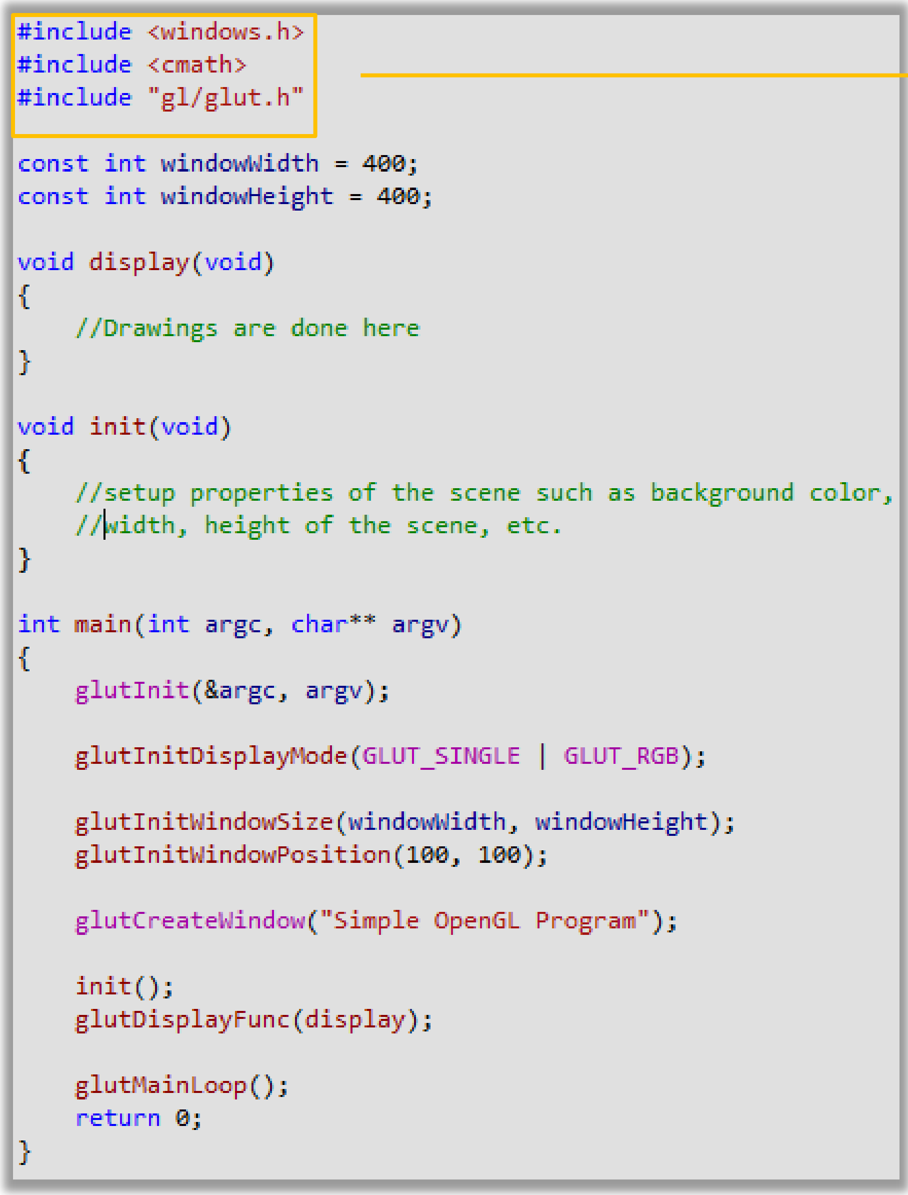
- What is OpenGL?
  - A cross-platform API (library) to produce 2D and 3D computer graphics application
- How can we use it?
  - It provides a set of primitive but powerful high-level rendering command. All drawing must be done through these.

Structure of an OpenGL Program



```
#include <windows.h>
#include <cmath>
#include "gl/glut.h"

const int windowWidth = 400;
const int windowHeight = 400;

void display(void)
{
    //Drawings are done here
}

void init(void)
{
    //setup properties of the scene such as background color,
    //width, height of the scene, etc.
}

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

    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);

    glutInitWindowSize(windowWidth, windowHeight);
    glutInitWindowPosition(100, 100);

    glutCreateWindow("Simple OpenGL Program");

    init();
    glutDisplayFunc(display);

    glutMainLoop();
    return 0;
}
```

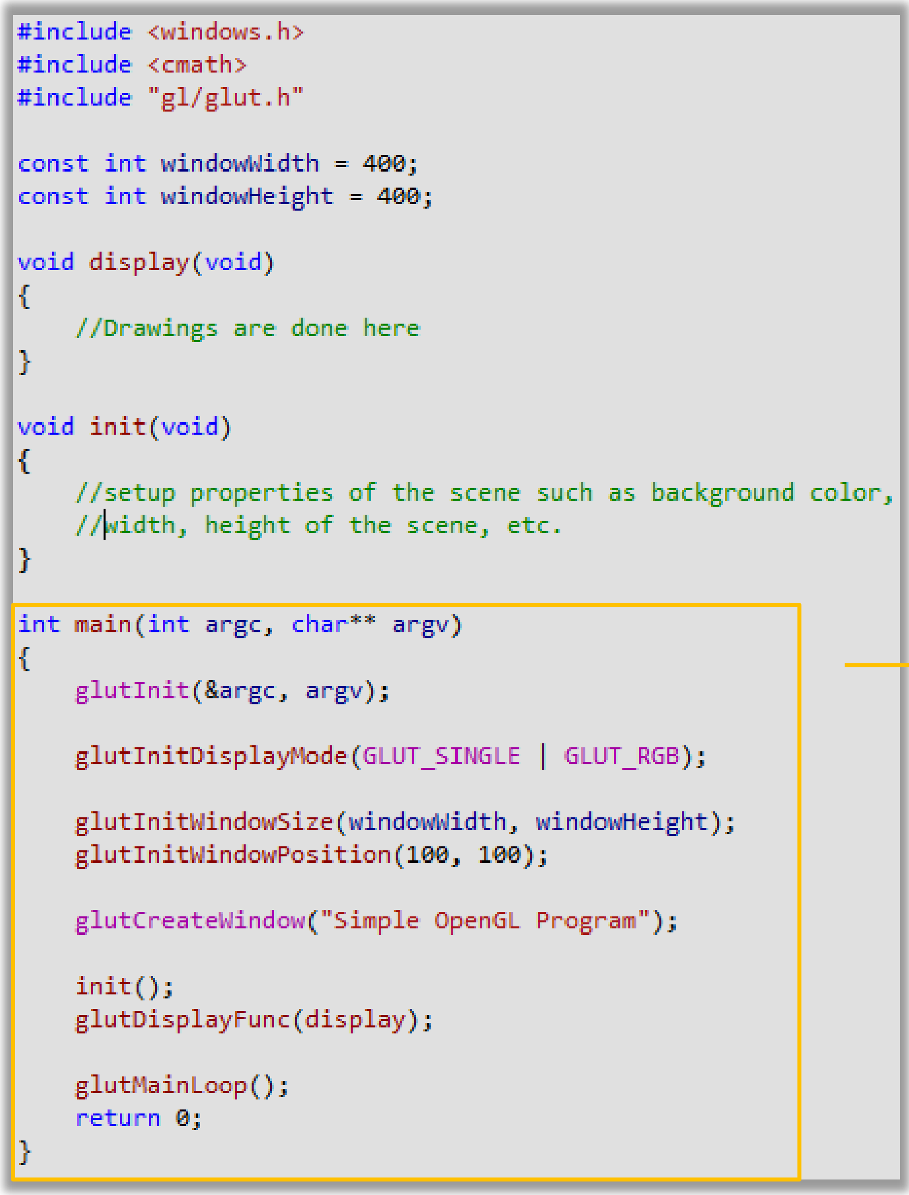
Structure of an OpenGL Program



```
#include <windows.h>
#include <cmath>
#include "gl/glut.h"

const int windowWidth = 400;
const int windowHeight = 400;

void display(void)
{
    //Drawings are done here
}

void init(void)
{
    //setup properties of the scene such as background color,
    //width, height of the scene, etc.
}

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

    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);

    glutInitWindowSize(windowWidth, windowHeight);
    glutInitWindowPosition(100, 100);

    glutCreateWindow("Simple OpenGL Program");

    init();
    glutDisplayFunc(display);

    glutMainLoop();
    return 0;
}
```

File Headers to be included

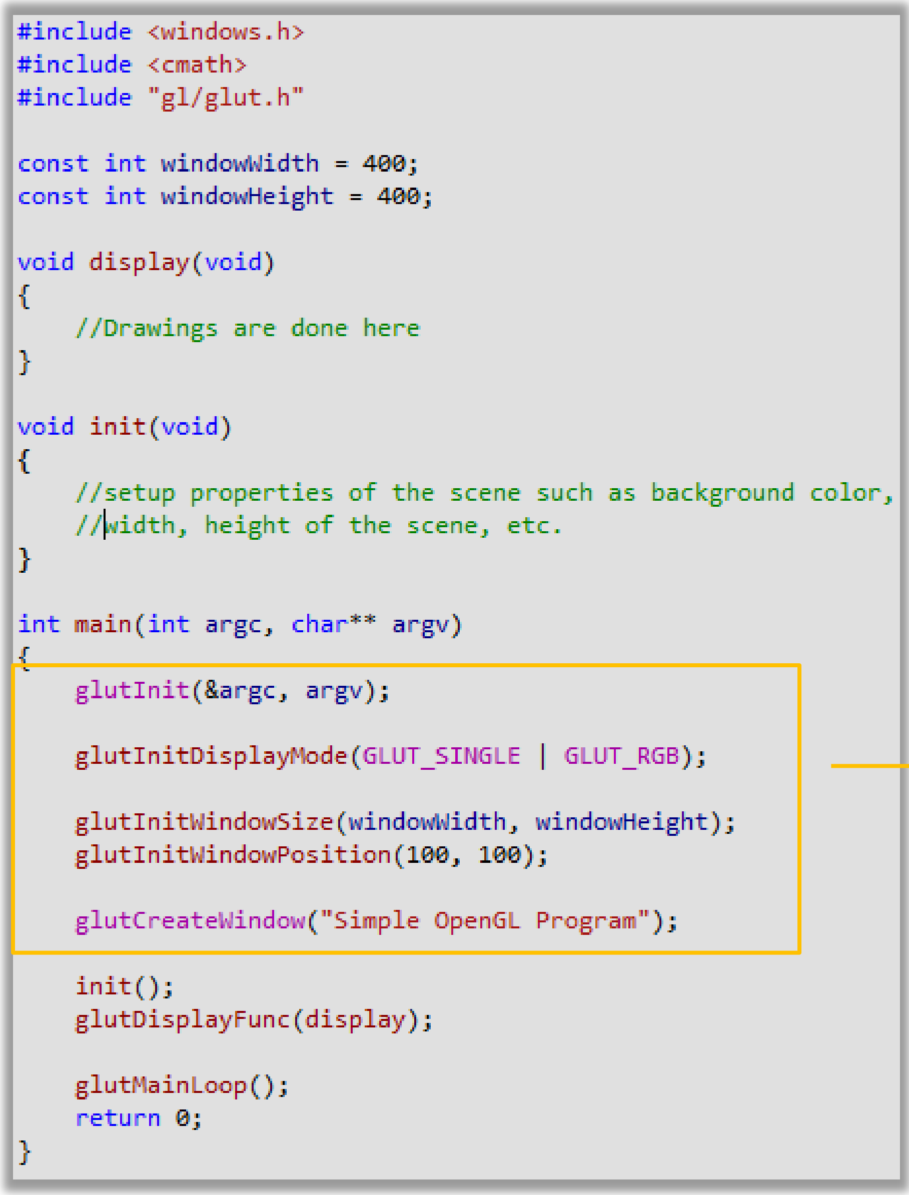
Structure of an OpenGL Program



```
#include <windows.h>
#include <cmath>
#include "gl/glut.h"

const int windowWidth = 400;
const int windowHeight = 400;

void display(void)
{
    //Drawings are done here
}

void init(void)
{
    //setup properties of the scene such as background color,
    //width, height of the scene, etc.
}

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

    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);

    glutInitWindowSize(windowWidth, windowHeight);
    glutInitWindowPosition(100, 100);

    glutCreateWindow("Simple OpenGL Program");

    init();
    glutDisplayFunc(display);

    glutMainLoop();
    return 0;
}
```

Main(): Program's entry point

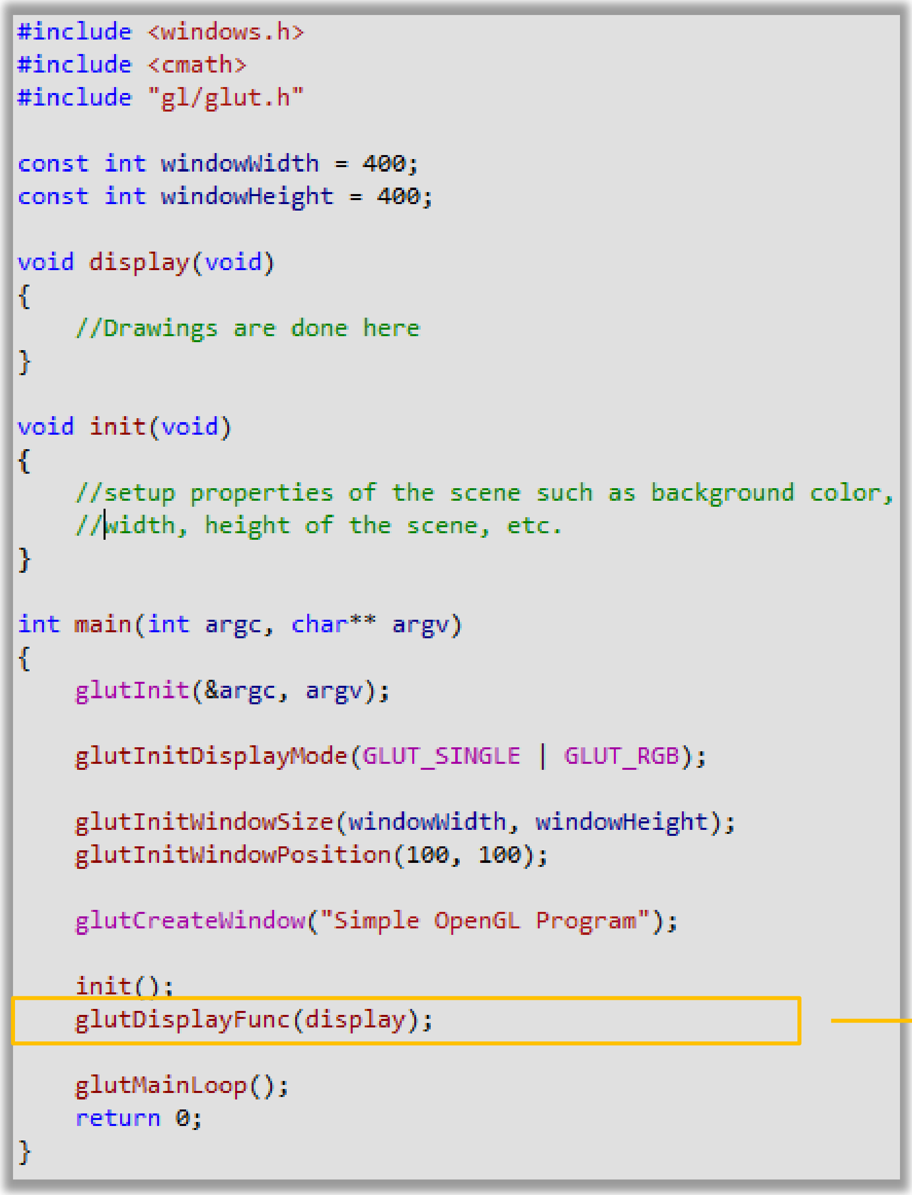
Structure of an OpenGL Program



```
#include <windows.h>
#include <cmath>
#include "gl/glut.h"

const int windowWidth = 400;
const int windowHeight = 400;

void display(void)
{
    //Drawings are done here
}

void init(void)
{
    //setup properties of the scene such as background color,
    //width, height of the scene, etc.
}

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

    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);

    glutInitWindowSize(windowWidth, windowHeight);
    glutInitWindowPosition(100, 100);

    glutCreateWindow("Simple OpenGL Program");

    init();
    glutDisplayFunc(display);

    glutMainLoop();
    return 0;
}
```

Setup the display window

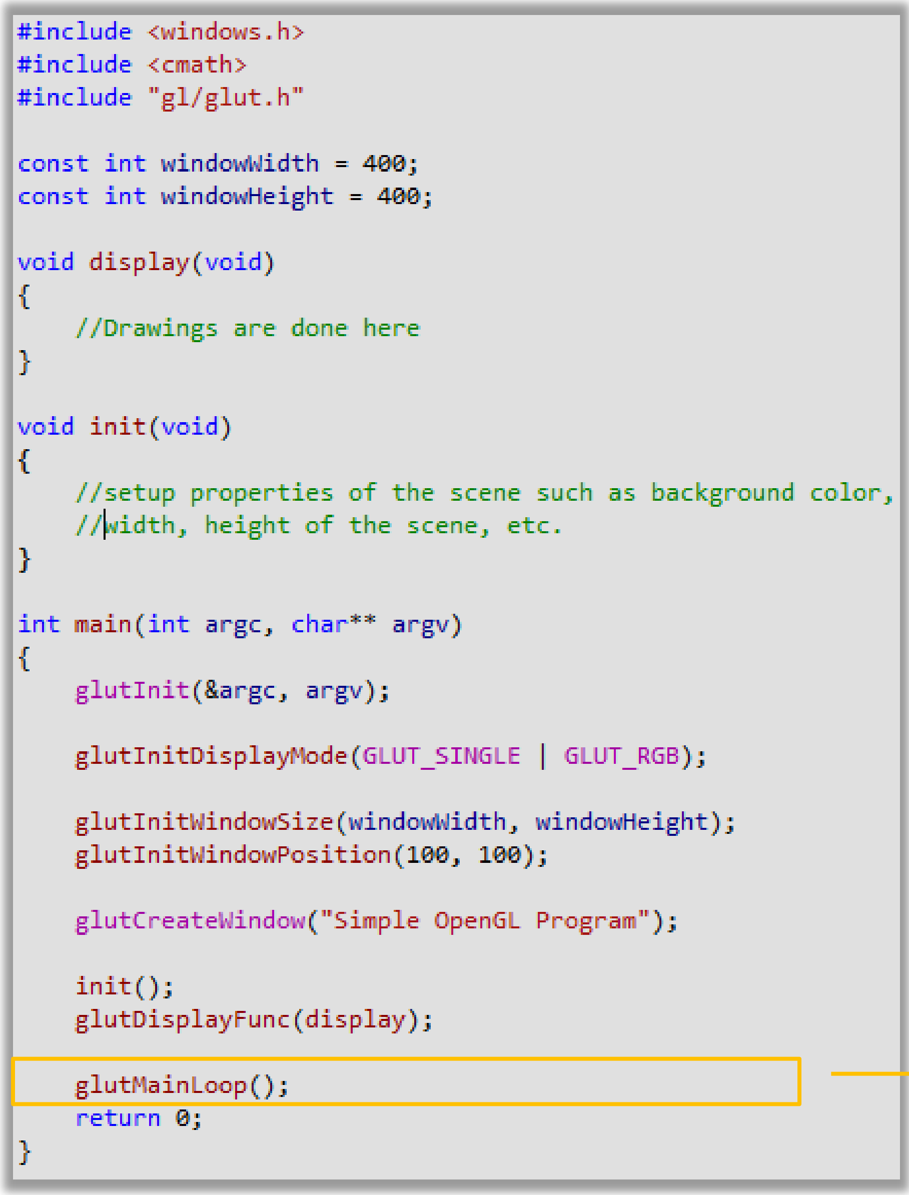
Structure of an OpenGL Program



```
#include <windows.h>
#include <cmath>
#include "gl/glut.h"

const int windowWidth = 400;
const int windowHeight = 400;

void display(void)
{
    //Drawings are done here
}

void init(void)
{
    //setup properties of the scene such as background color,
    //width, height of the scene, etc.
}

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

    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);

    glutInitWindowSize(windowWidth, windowHeight);
    glutInitWindowPosition(100, 100);

    glutCreateWindow("Simple OpenGL Program");

    init();
    glutDisplayFunc(display);

    glutMainLoop();
    return 0;
}
```

Tell OpenGL that the display() is where we do our drawing

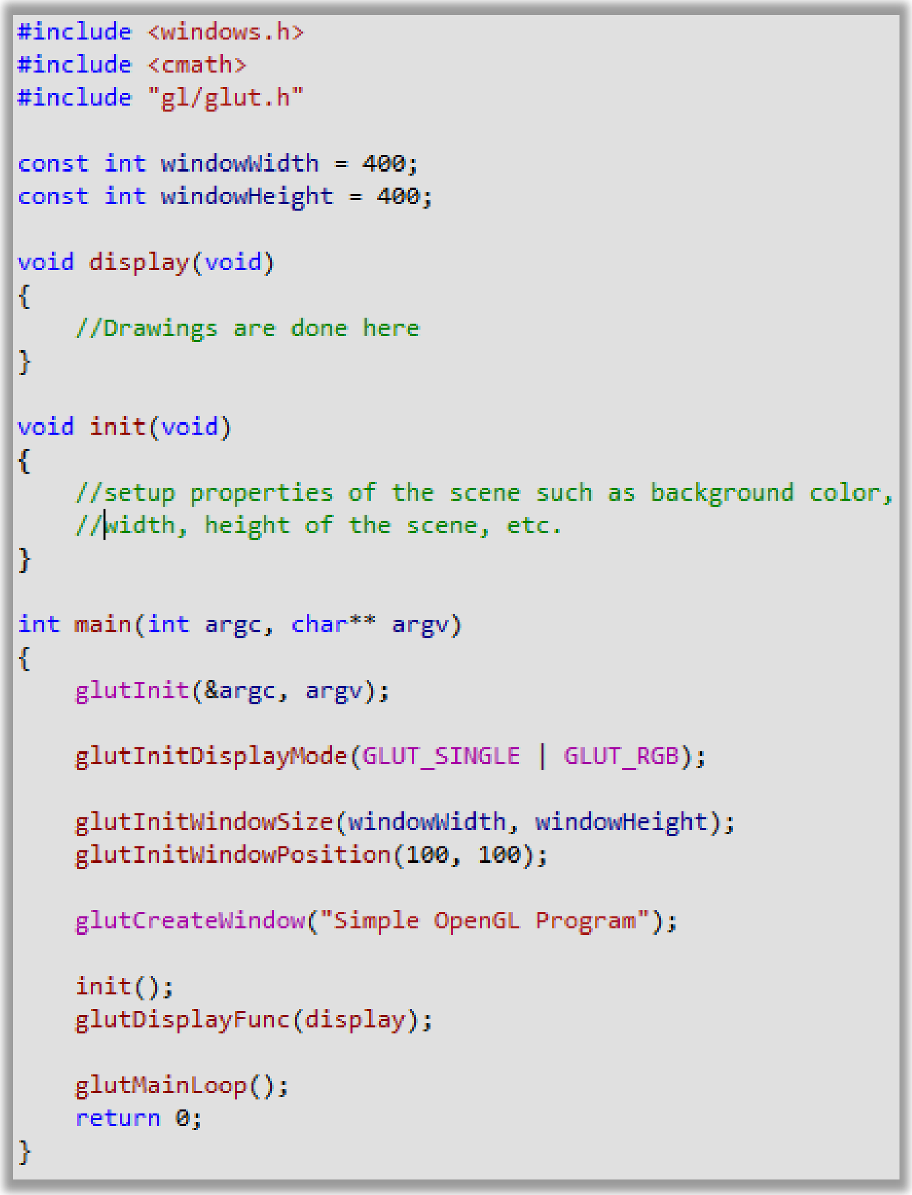
Structure of an OpenGL Program



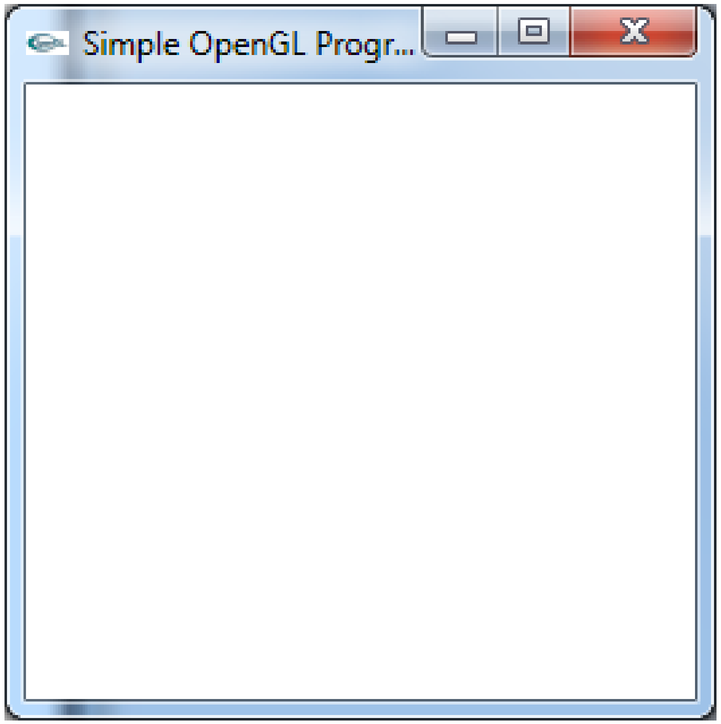
```
#include <windows.h>
#include <cmath>
#include "gl/glut.h"

const int windowWidth = 400;
const int windowHeight = 400;

void display(void)
{
    //Drawings are done here
}

void init(void)
{
    //setup properties of the scene such as background color,
    //width, height of the scene, etc.
}

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

    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);

    glutInitWindowSize(windowWidth, windowHeight);
    glutInitWindowPosition(100, 100);

    glutCreateWindow("Simple OpenGL Program");

    init();
    glutDisplayFunc(display);

    glutMainLoop();
    return 0;
}
```

Tell OpenGL that our program is ready and will stand there waiting for input events

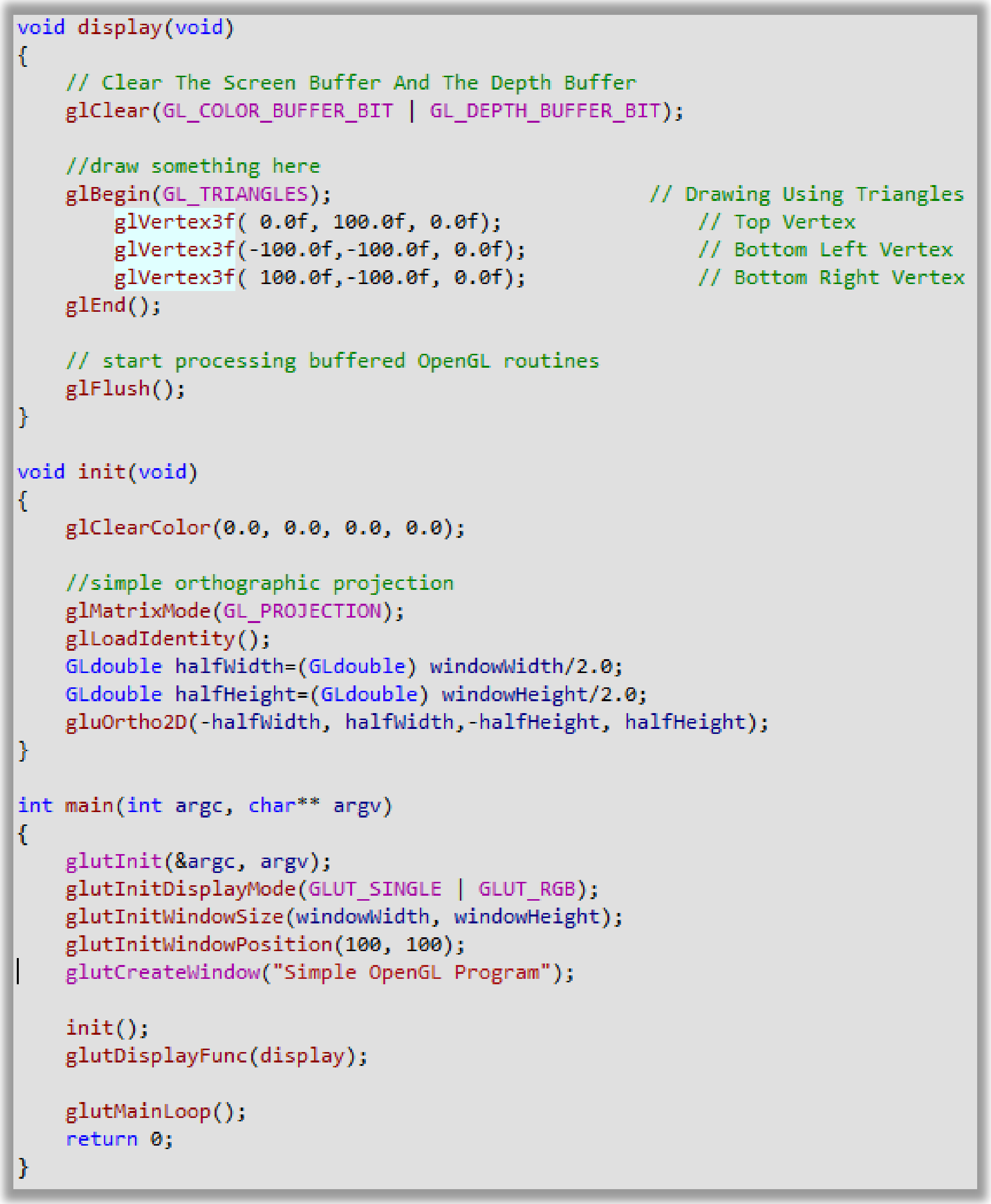
Structure of an OpenGL Program



```
#include <windows.h>
#include <cmath>
#include "gl/glut.h"

const int windowWidth = 400;
const int windowHeight = 400;

void display(void)
{
    //Drawings are done here
}

void init(void)
{
    //setup properties of the scene such as background color,
    //width, height of the scene, etc.
}

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

    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);

    glutInitWindowSize(windowWidth, windowHeight);
    glutInitWindowPosition(100, 100);

    glutCreateWindow("Simple OpenGL Program");

    init();
    glutDisplayFunc(display);

    glutMainLoop();
    return 0;
}
```

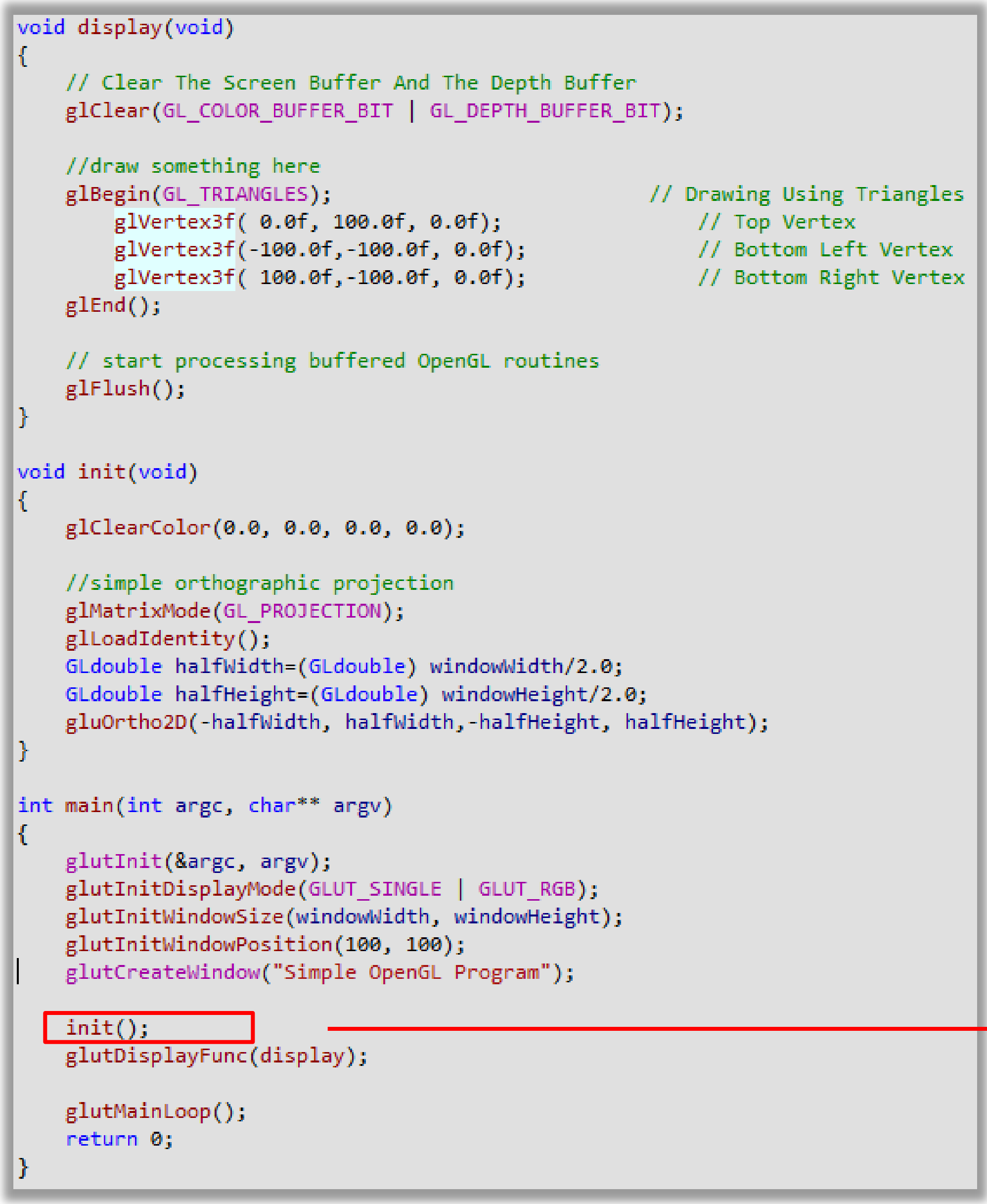


A Simple OpenGL Program



```
void display(void)
{
    // Clear The Screen Buffer And The Depth Buffer
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    //draw something here
    glBegin(GL_TRIANGLES);                          // Drawing Using Triangles
        glVertex3f( 0.0f, 100.0f, 0.0f);                // Top Vertex
        glVertex3f(-100.0f,-100.0f, 0.0f);              // Bottom Left Vertex
        glVertex3f( 100.0f,-100.0f, 0.0f);              // Bottom Right Vertex
    glEnd();

    // start processing buffered OpenGL routines
    glFlush();
}

void init(void)
{
    glClearColor(0.0, 0.0, 0.0, 0.0);

    //simple orthographic projection
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    GLdouble halfWidth=(GLdouble) windowWidth/2.0;
    GLdouble halfHeight=(GLdouble) windowHeight/2.0;
    gluOrtho2D(-halfWidth, halfWidth,-halfHeight, halfHeight);
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(windowWidth, windowHeight);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Simple OpenGL Program");

    init();
    glutDisplayFunc(display);

    glutMainLoop();
    return 0;
}
```

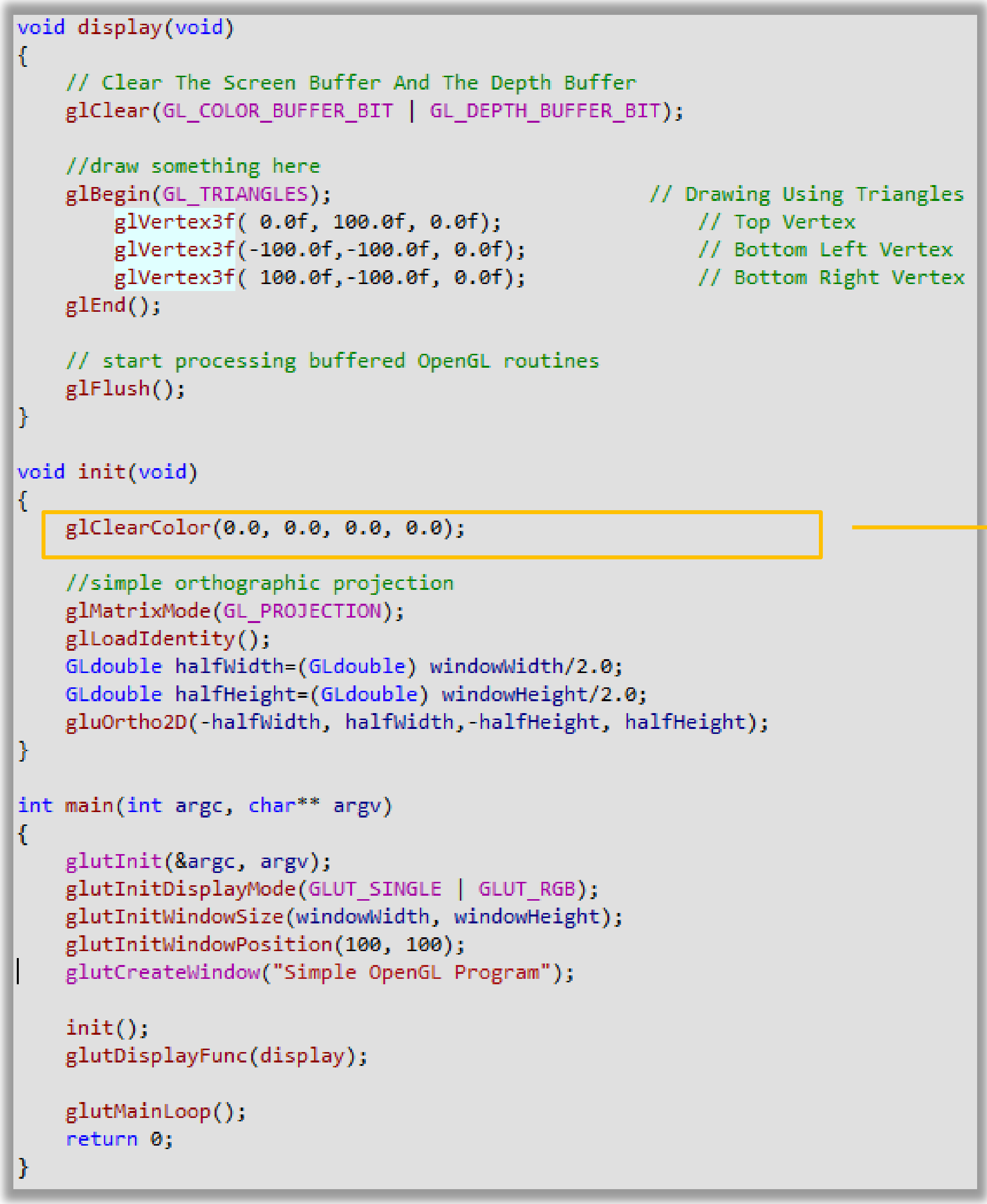
A Simple OpenGL Program



```
void display(void)
{
    // Clear The Screen Buffer And The Depth Buffer
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    //draw something here
    glBegin(GL_TRIANGLES);                          // Drawing Using Triangles
        glVertex3f( 0.0f, 100.0f, 0.0f);                // Top Vertex
        glVertex3f(-100.0f,-100.0f, 0.0f);              // Bottom Left Vertex
        glVertex3f( 100.0f,-100.0f, 0.0f);              // Bottom Right Vertex
    glEnd();

    // start processing buffered OpenGL routines
    glFlush();
}

void init(void)
{
    glClearColor(0.0, 0.0, 0.0, 0.0);

    //simple orthographic projection
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    GLdouble halfWidth=(GLdouble) windowWidth/2.0;
    GLdouble halfHeight=(GLdouble) windowHeight/2.0;
    gluOrtho2D(-halfWidth, halfWidth,-halfHeight, halfHeight);
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(windowWidth, windowHeight);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Simple OpenGL Program");

    init();
    glutDisplayFunc(display);

    glutMainLoop();
    return 0;
}
```

Setup necessary properties

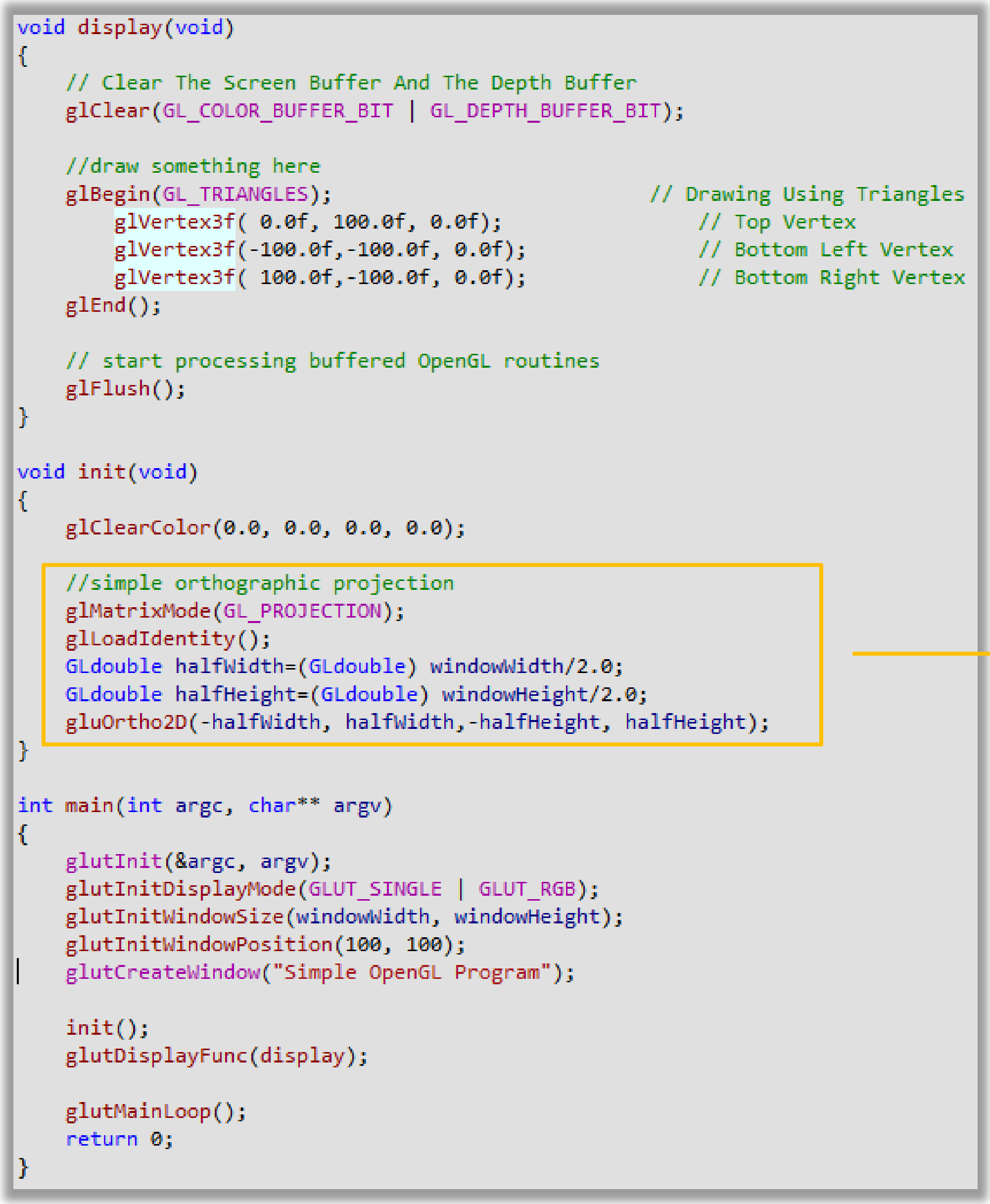
A Simple OpenGL Program



```
void display(void)
{
    // Clear The Screen Buffer And The Depth Buffer
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    //draw something here
    glBegin(GL_TRIANGLES);                          // Drawing Using Triangles
        glVertex3f( 0.0f, 100.0f, 0.0f);                // Top Vertex
        glVertex3f(-100.0f,-100.0f, 0.0f);              // Bottom Left Vertex
        glVertex3f( 100.0f,-100.0f, 0.0f);              // Bottom Right Vertex
    glEnd();

    // start processing buffered OpenGL routines
    glFlush();
}

void init(void)
{
    glClearColor(0.0, 0.0, 0.0, 0.0);

    //simple orthographic projection
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    GLdouble halfWidth=(GLdouble) windowWidth/2.0;
    GLdouble halfHeight=(GLdouble) windowHeight/2.0;
    gluOrtho2D(-halfWidth, halfWidth,-halfHeight, halfHeight);
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(windowWidth, windowHeight);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Simple OpenGL Program");

    init();
    glutDisplayFunc(display);

    glutMainLoop();
    return 0;
}
```

Specify the background color (RGBA)

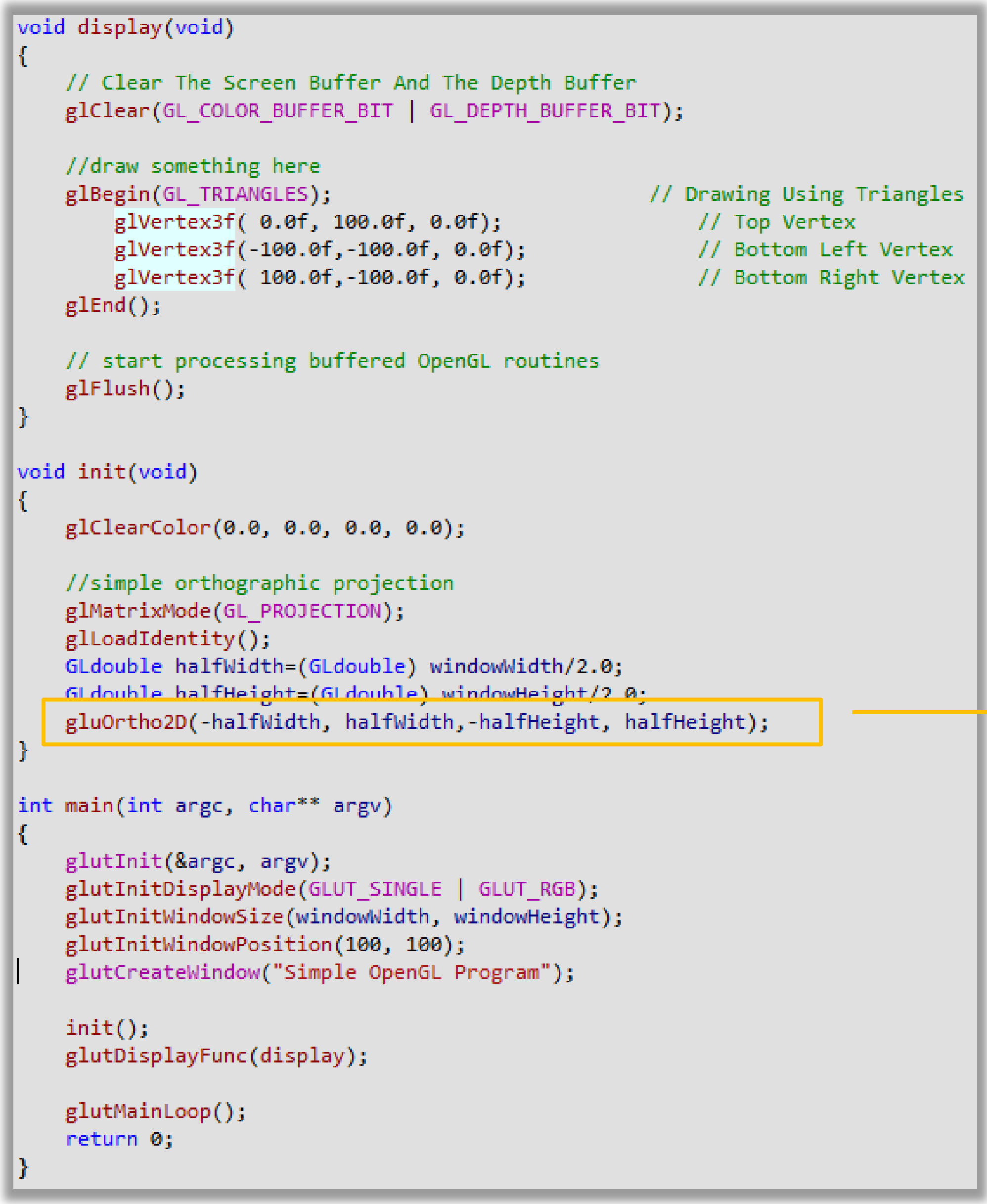
A Simple OpenGL Program



```
void display(void)
{
    // Clear The Screen Buffer And The Depth Buffer
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    //draw something here
    glBegin(GL_TRIANGLES);                          // Drawing Using Triangles
        glVertex3f( 0.0f, 100.0f, 0.0f);                // Top Vertex
        glVertex3f(-100.0f,-100.0f, 0.0f);              // Bottom Left Vertex
        glVertex3f( 100.0f,-100.0f, 0.0f);              // Bottom Right Vertex
    glEnd();

    // start processing buffered OpenGL routines
    glFlush();
}

void init(void)
{
    glClearColor(0.0, 0.0, 0.0, 0.0);

    //simple orthographic projection
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    GLdouble halfWidth=(GLdouble) windowWidth/2.0;
    GLdouble halfHeight=(GLdouble) windowHeight/2.0;
    gluOrtho2D(-halfWidth, halfWidth,-halfHeight, halfHeight);
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(windowWidth, windowHeight);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Simple OpenGL Program");

    init();
    glutDisplayFunc(display);

    glutMainLoop();
    return 0;
}
```

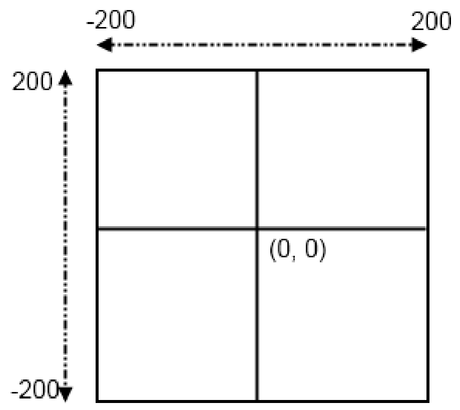
Specify a simple orthographic projection

A Simple OpenGL Program

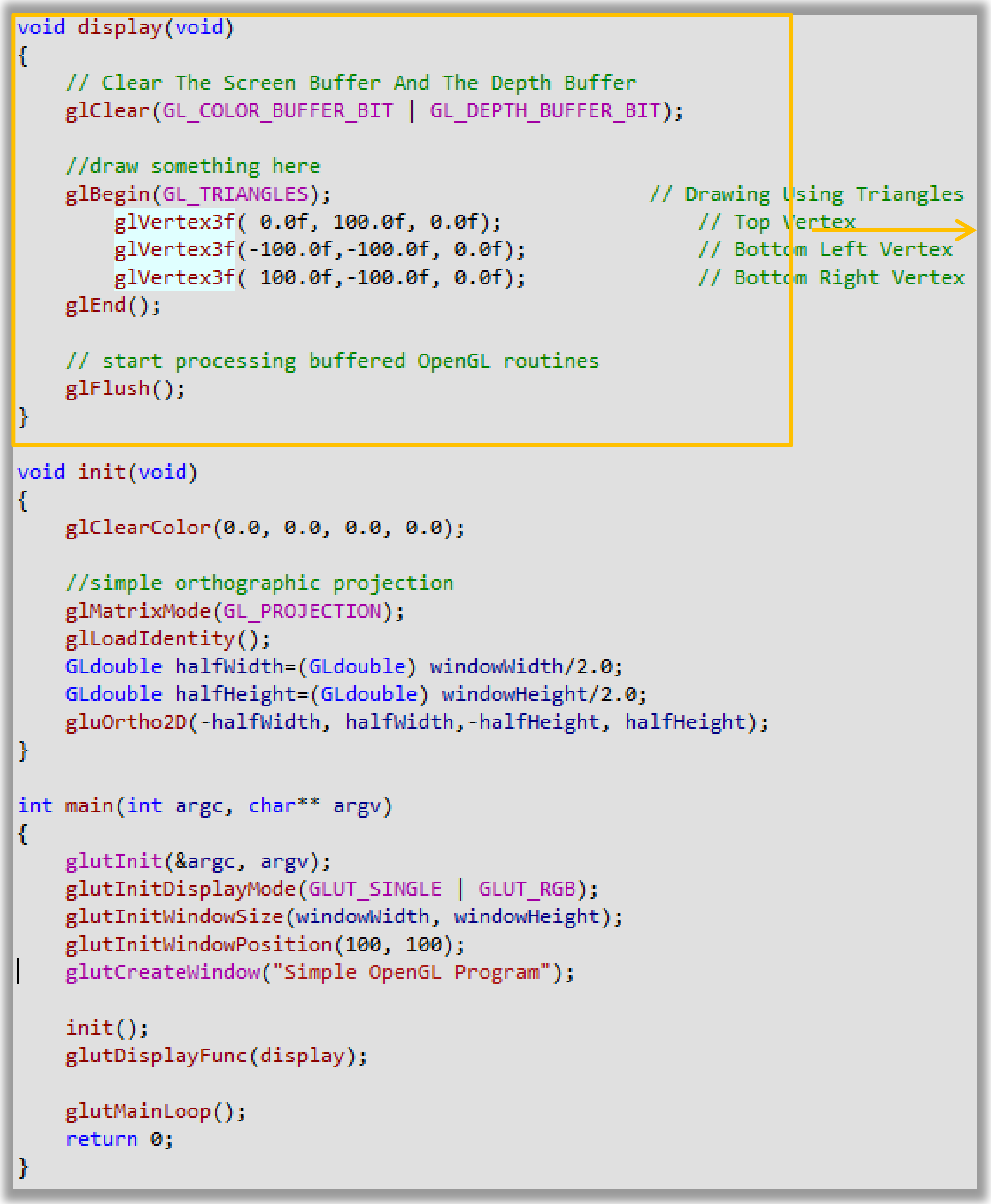


```
void display(void)
{
    // Clear The Screen Buffer And The Depth Buffer
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    //draw something here
    glBegin(GL_TRIANGLES);                          // Drawing Using Triangles
        glVertex3f( 0.0f, 100.0f, 0.0f);                // Top Vertex
        glVertex3f(-100.0f,-100.0f, 0.0f);              // Bottom Left Vertex
        glVertex3f( 100.0f,-100.0f, 0.0f);              // Bottom Right Vertex
    glEnd();

    // start processing buffered OpenGL routines
    glFlush();
}

void init(void)
{
    glClearColor(0.0, 0.0, 0.0, 0.0);

    //simple orthographic projection
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    GLdouble halfWidth=(GLdouble) windowWidth/2.0;
    GLdouble halfHeight=(GLdouble) windowHeight/2.0;
    gluOrtho2D(-halfWidth, halfWidth,-halfHeight, halfHeight);
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(windowWidth, windowHeight);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Simple OpenGL Program");

    init();
    glutDisplayFunc(display);

    glutMainLoop();
    return 0;
}
```

$\longrightarrow$ Specify the screen coordinates



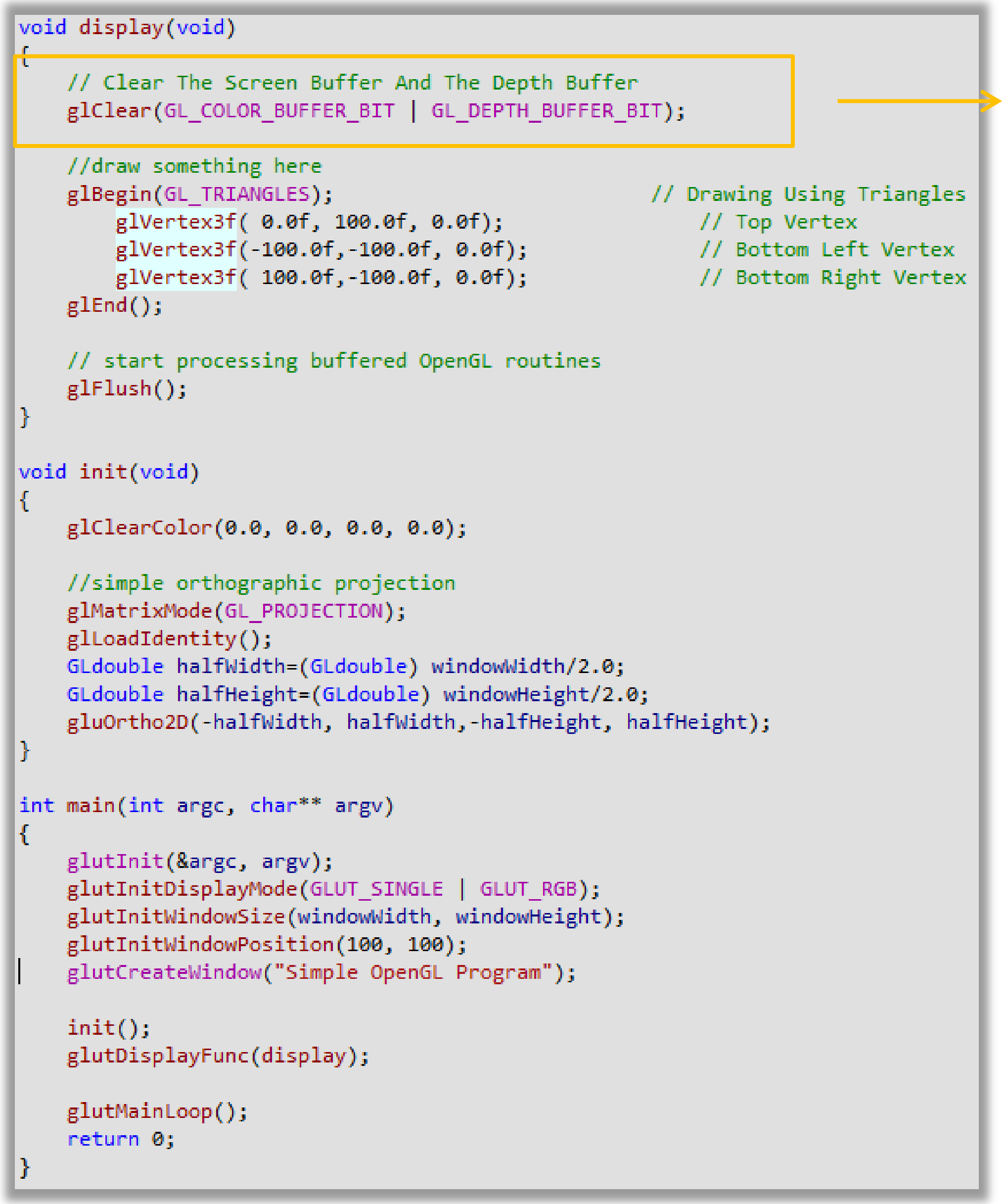
A Simple OpenGL Program



```
void display(void)
{
    // Clear The Screen Buffer And The Depth Buffer
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    //draw something here
    glBegin(GL_TRIANGLES);                          // Drawing Using Triangles
        glVertex3f( 0.0f, 100.0f, 0.0f);                // Top Vertex
        glVertex3f(-100.0f,-100.0f, 0.0f);              // Bottom Left Vertex
        glVertex3f( 100.0f,-100.0f, 0.0f);              // Bottom Right Vertex
    glEnd();

    // start processing buffered OpenGL routines
    glFlush();
}

void init(void)
{
    glClearColor(0.0, 0.0, 0.0, 0.0);

    //simple orthographic projection
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    GLdouble halfWidth=(GLdouble) windowWidth/2.0;
    GLdouble halfHeight=(GLdouble) windowHeight/2.0;
    gluOrtho2D(-halfWidth, halfWidth,-halfHeight, halfHeight);
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(windowWidth, windowHeight);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Simple OpenGL Program");

    init();
    glutDisplayFunc(display);

    glutMainLoop();
    return 0;
}
```

The actual drawings in here

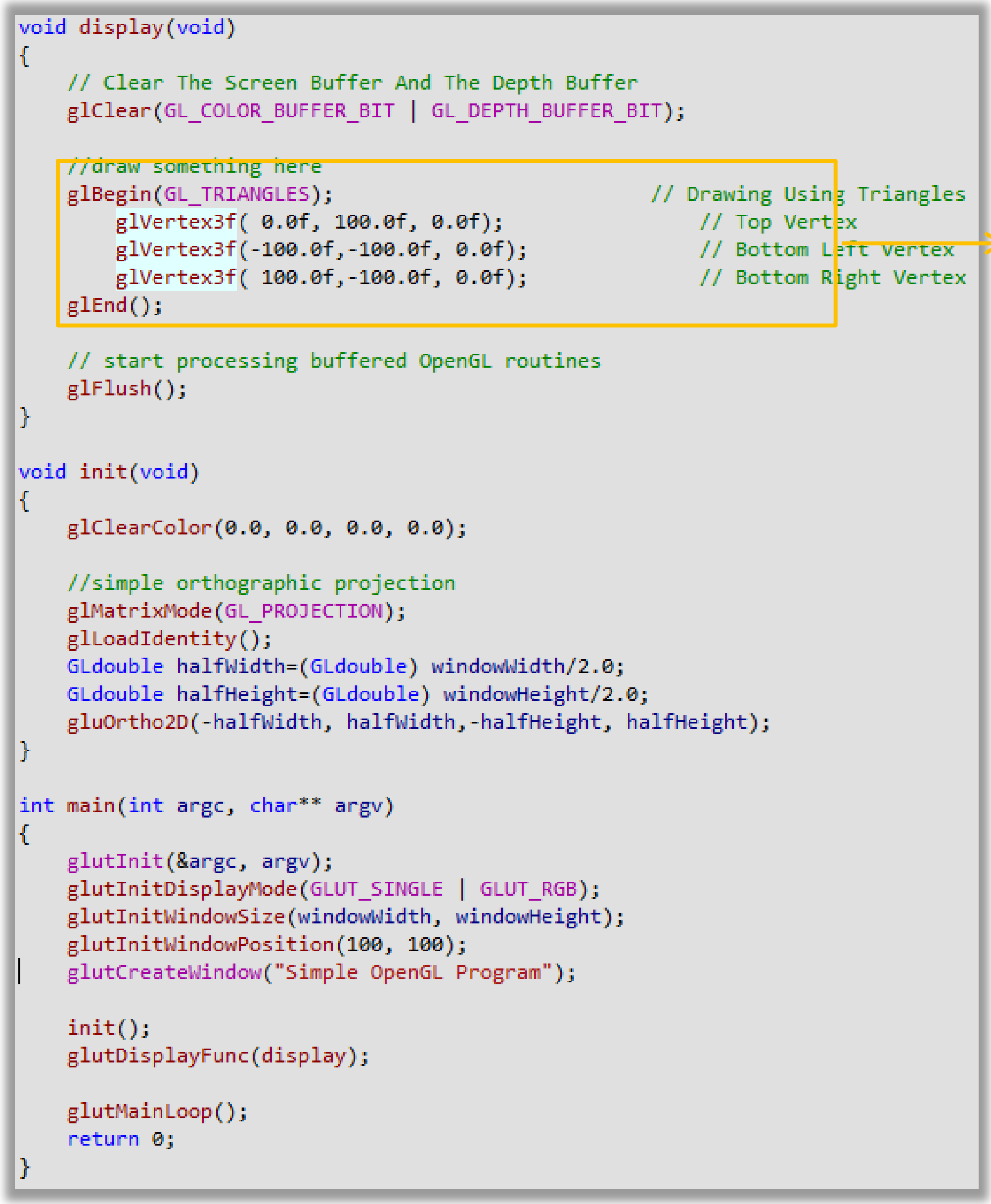
A Simple OpenGL Program



```
void display(void)
{
    // Clear The Screen Buffer And The Depth Buffer
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    //draw something here
    glBegin(GL_TRIANGLES);                        // Drawing Using Triangles
        glVertex3f( 0.0f, 100.0f, 0.0f);            // Top Vertex
        glVertex3f(-100.0f,-100.0f, 0.0f);          // Bottom Left Vertex
        glVertex3f( 100.0f,-100.0f, 0.0f);          // Bottom Right Vertex
    glEnd();

    // start processing buffered OpenGL routines
    glFlush();
}

void init(void)
{
    glClearColor(0.0, 0.0, 0.0, 0.0);

    //simple orthographic projection
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    GLdouble halfWidth=(GLdouble) windowWidth/2.0;
    GLdouble halfHeight=(GLdouble) windowHeight/2.0;
    gluOrtho2D(-halfWidth, halfWidth,-halfHeight, halfHeight);
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(windowWidth, windowHeight);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Simple OpenGL Program");

    init();
    glutDisplayFunc(display);

    glutMainLoop();
    return 0;
}
```

Clear all the buffers.

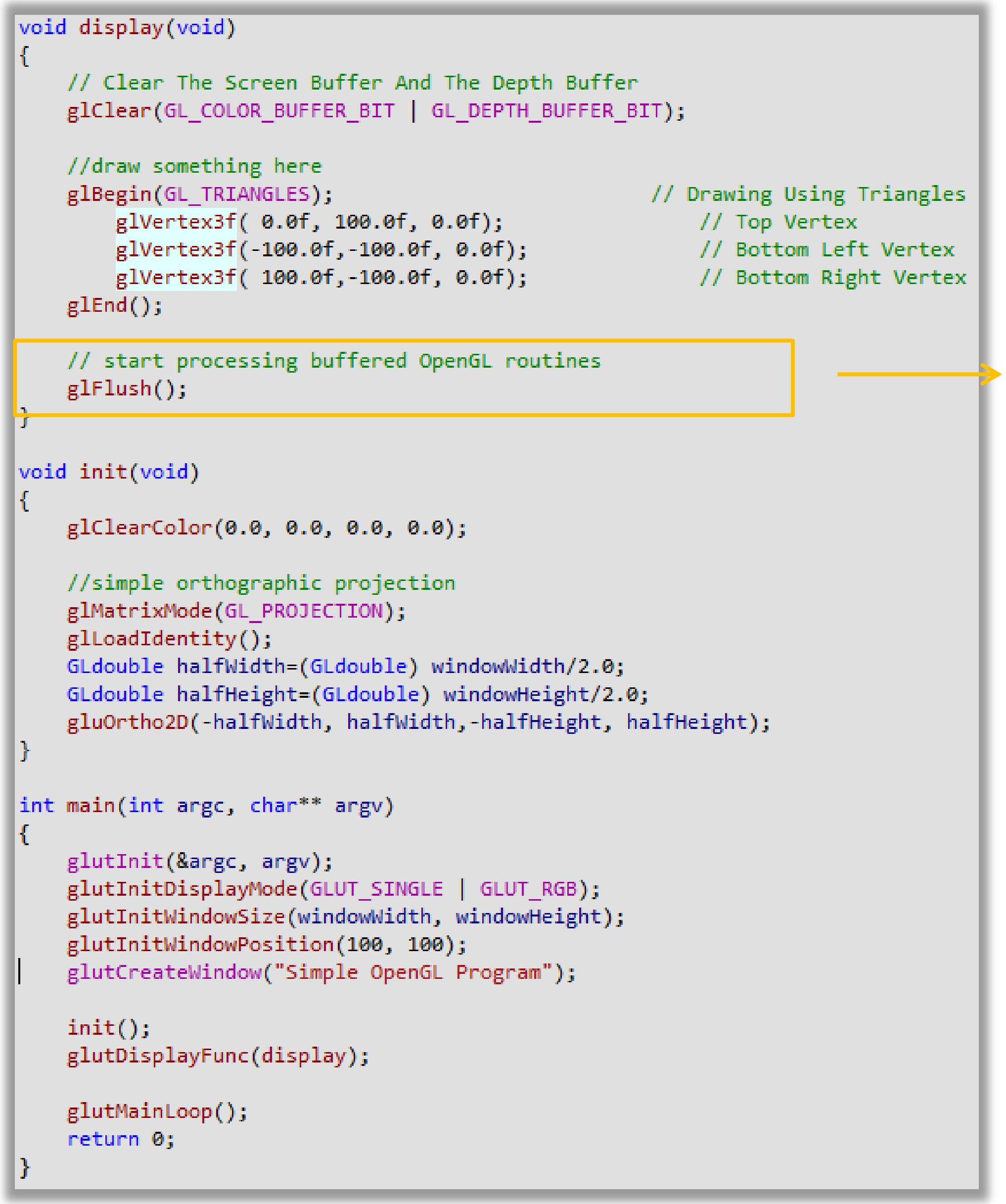
A Simple OpenGL Program



```
void display(void)
{
    // Clear The Screen Buffer And The Depth Buffer
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    //draw something here
    glBegin(GL_TRIANGLES);                        // Drawing Using Triangles
        glVertex3f( 0.0f, 100.0f, 0.0f);              // Top Vertex
        glVertex3f(-100.0f,-100.0f, 0.0f);            // Bottom Left Vertex
        glVertex3f( 100.0f,-100.0f, 0.0f);            // Bottom Right Vertex
    glEnd();

    // start processing buffered OpenGL routines
    glFlush();
}

void init(void)
{
    glClearColor(0.0, 0.0, 0.0, 0.0);

    //simple orthographic projection
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    GLdouble halfWidth=(GLdouble) windowWidth/2.0;
    GLdouble halfHeight=(GLdouble) windowHeight/2.0;
    gluOrtho2D(-halfWidth, halfWidth,-halfHeight, halfHeight);
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(windowWidth, windowHeight);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Simple OpenGL Program");

    init();
    glutDisplayFunc(display);

    glutMainLoop();
    return 0;
}
```

Draw a triangle

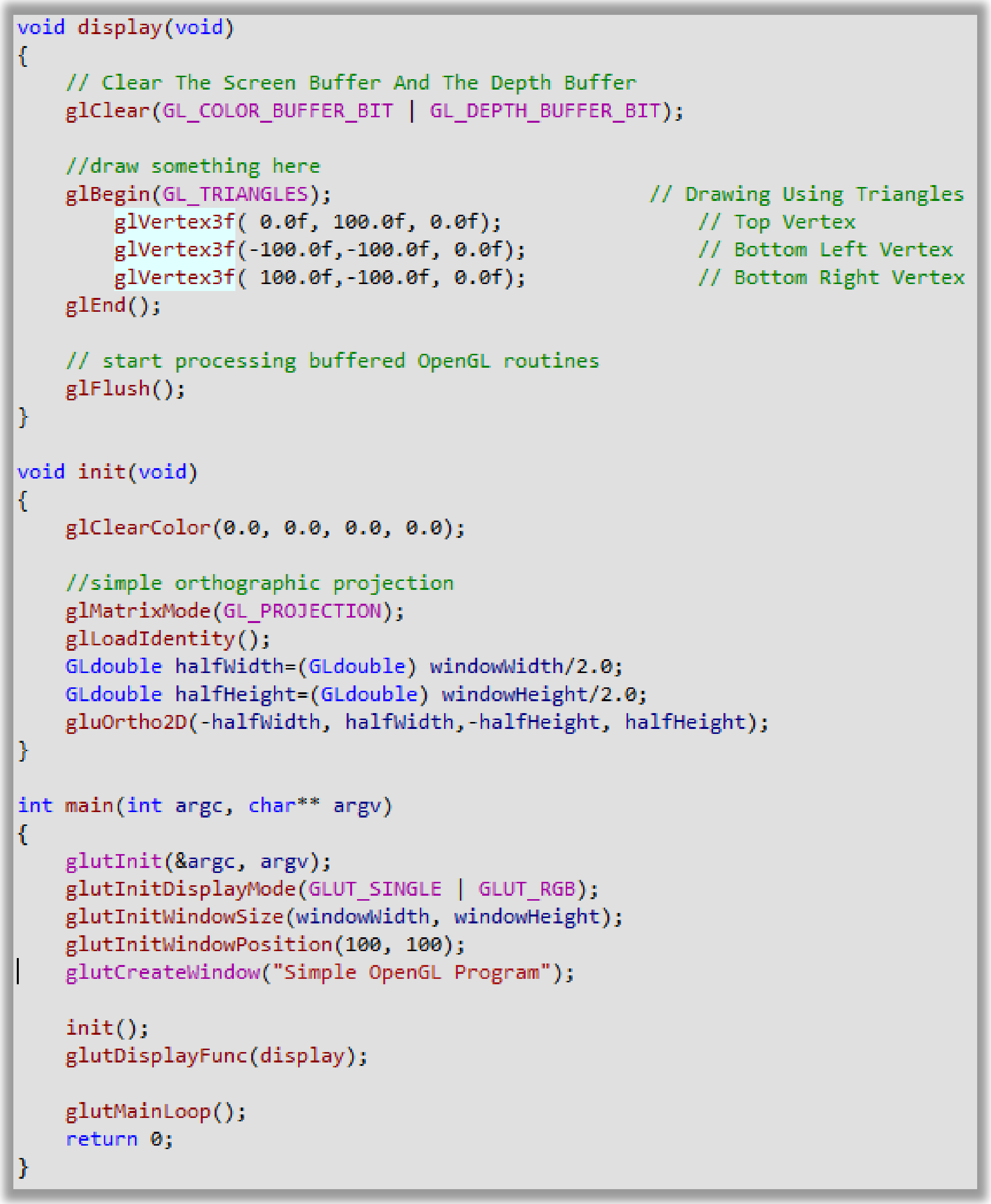
A Simple OpenGL Program



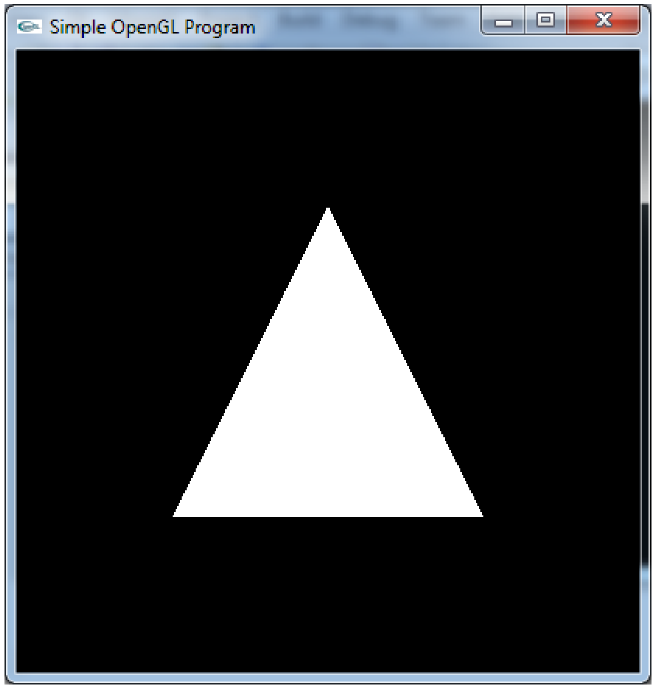
```
void display(void)
{
    // Clear The Screen Buffer And The Depth Buffer
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    //draw something here
    glBegin(GL_TRIANGLES);                          // Drawing Using Triangles
        glVertex3f( 0.0f, 100.0f, 0.0f);                // Top Vertex
        glVertex3f(-100.0f,-100.0f, 0.0f);              // Bottom Left Vertex
        glVertex3f( 100.0f,-100.0f, 0.0f);              // Bottom Right Vertex
    glEnd();

    // start processing buffered OpenGL routines
    glFlush();
}

void init(void)
{
    glClearColor(0.0, 0.0, 0.0, 0.0);

    //simple orthographic projection
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    GLdouble halfWidth=(GLdouble) windowWidth/2.0;
    GLdouble halfHeight=(GLdouble) windowHeight/2.0;
    gluOrtho2D(-halfWidth, halfWidth,-halfHeight, halfHeight);
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(windowWidth, windowHeight);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Simple OpenGL Program");

    init();
    glutDisplayFunc(display);

    glutMainLoop();
    return 0;
}
```

Process the buffers and display onto the screen

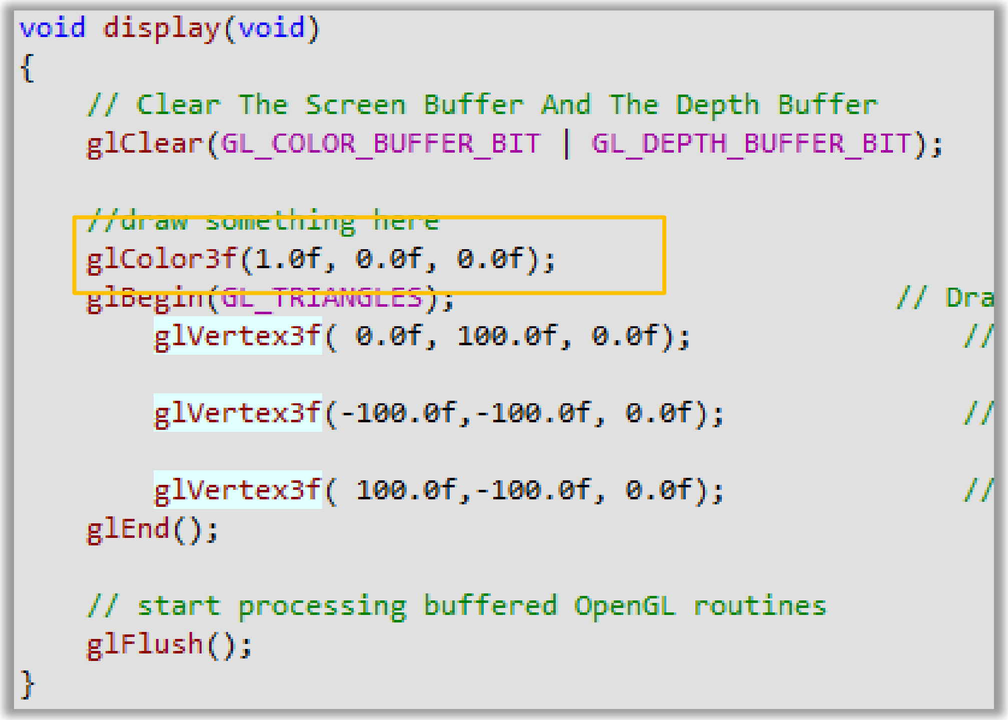
A Simple OpenGL Program



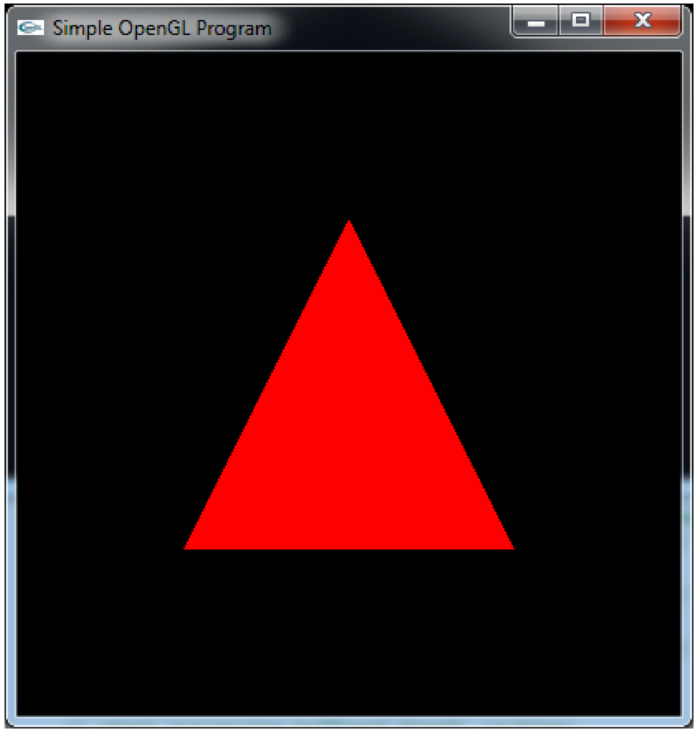
```
void display(void)
{
    // Clear The Screen Buffer And The Depth Buffer
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    //draw something here
    glBegin(GL_TRIANGLES);                          // Drawing Using Triangles
        glVertex3f( 0.0f, 100.0f, 0.0f);                // Top Vertex
        glVertex3f(-100.0f,-100.0f, 0.0f);              // Bottom Left Vertex
        glVertex3f( 100.0f,-100.0f, 0.0f);              // Bottom Right Vertex
    glEnd();

    // start processing buffered OpenGL routines
    glFlush();
}

void init(void)
{
    glClearColor(0.0, 0.0, 0.0, 0.0);

    //simple orthographic projection
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    GLdouble halfWidth=(GLdouble) windowWidth/2.0;
    GLdouble halfHeight=(GLdouble) windowHeight/2.0;
    gluOrtho2D(-halfWidth, halfWidth,-halfHeight, halfHeight);
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(windowWidth, windowHeight);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Simple OpenGL Program");

    init();
    glutDisplayFunc(display);

    glutMainLoop();
    return 0;
}
```



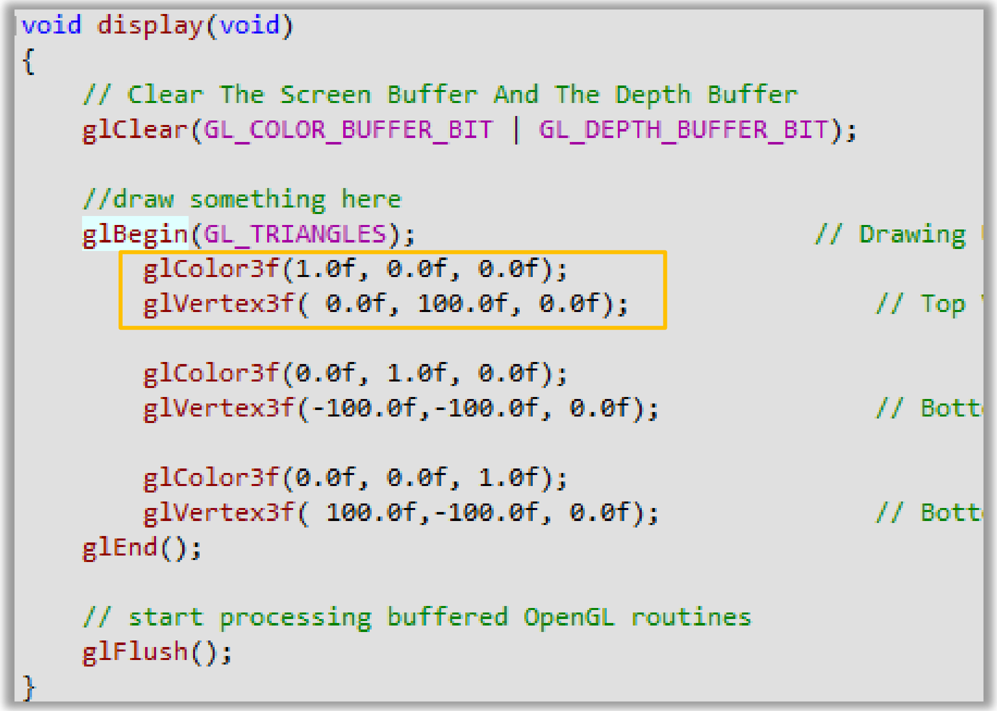
A Simple OpenGL Program



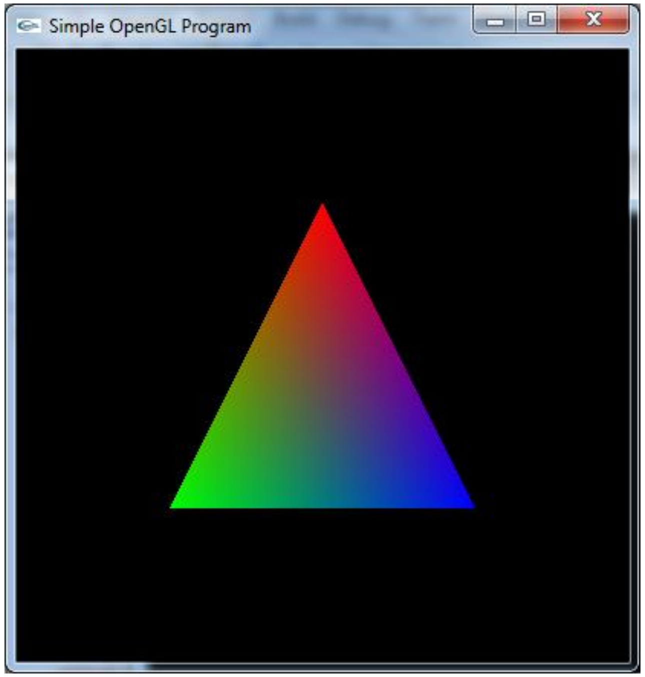
```
void display(void)
{
    // Clear The Screen Buffer And The Depth Buffer
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    //draw something here
    glColor3f(1.0f, 0.0f, 0.0f);
    glBegin(GL_TRIANGLES);                          // Dra
        glVertex3f( 0.0f, 100.0f, 0.0f);                //

        glVertex3f(-100.0f,-100.0f, 0.0f);              //

        glVertex3f( 100.0f,-100.0f, 0.0f);              //
    glEnd();

    // start processing buffered OpenGL routines
    glFlush();
}
```



A Simple OpenGL Program

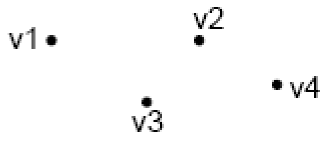


```
void display(void)
{
    // Clear The Screen Buffer And The Depth Buffer
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    //draw something here
    glBegin(GL_TRIANGLES);                          // Drawing
        glColor3f(1.0f, 0.0f, 0.0f);
        glVertex3f( 0.0f, 100.0f, 0.0f);                // Top

        glColor3f(0.0f, 1.0f, 0.0f);
        glVertex3f(-100.0f,-100.0f, 0.0f);              // Bott

        glColor3f(0.0f, 0.0f, 1.0f);
        glVertex3f( 100.0f,-100.0f, 0.0f);              // Bott
    glEnd();

    // start processing buffered OpenGL routines
    glFlush();
}
```

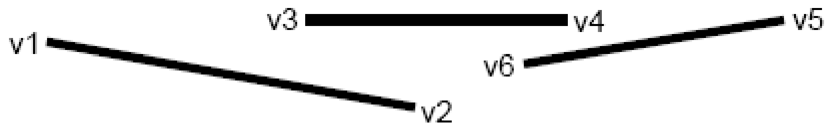


glBegin(…)

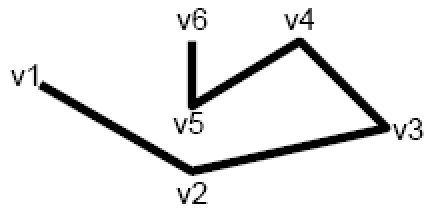
- GL_POINTS



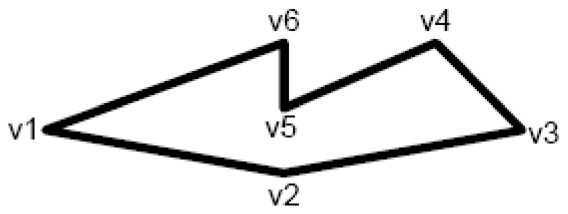
- GL_LINES



- GL_LINE_STRIP

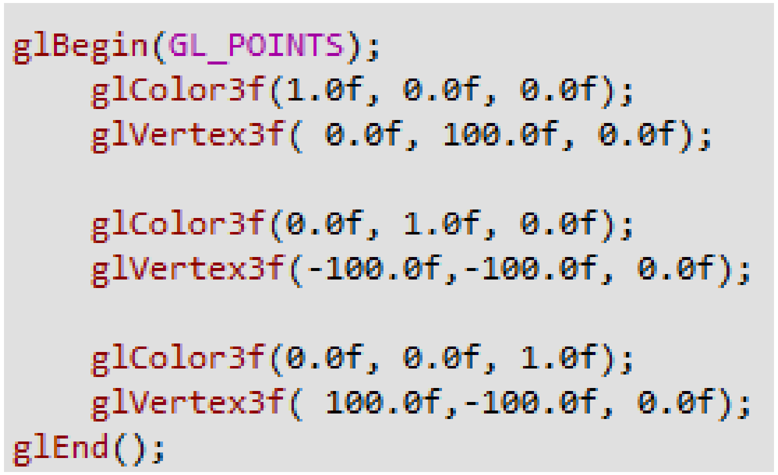


- GL_LINE_LOOP

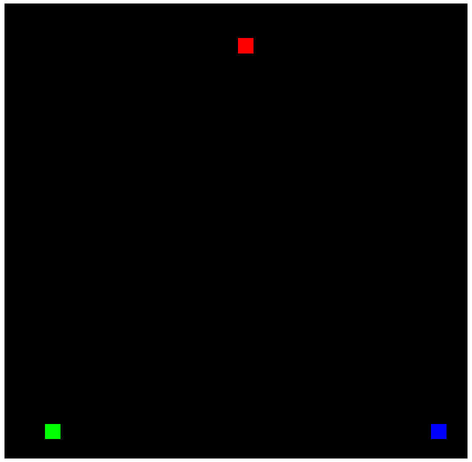


glBegin(…)

- GL_POINTS

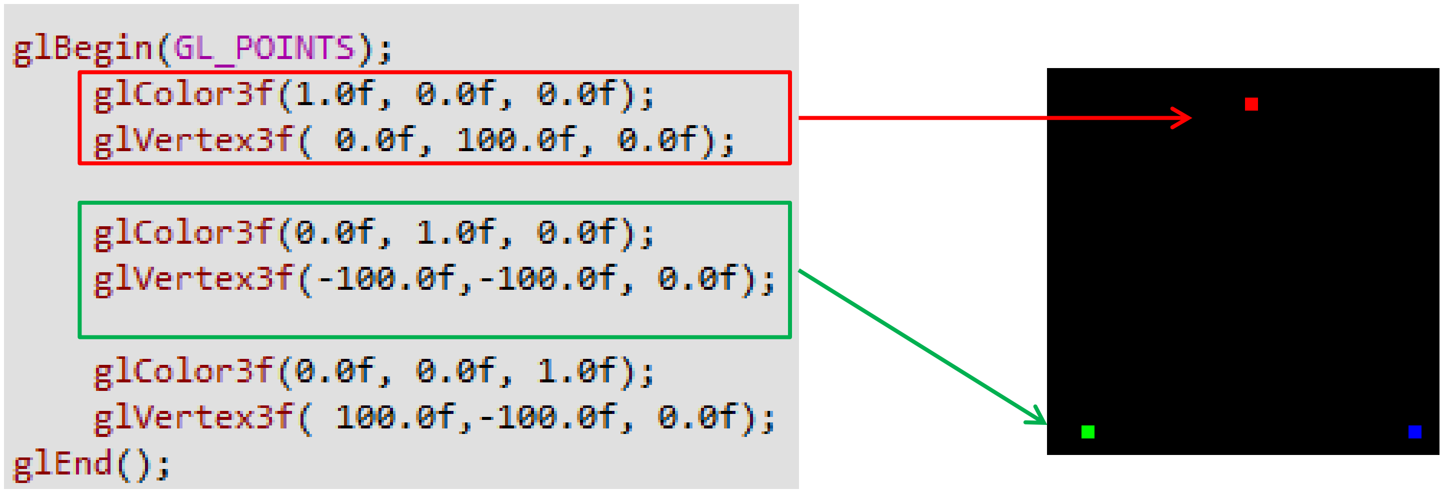


```
glBegin(GL_POINTS);
    glColor3f(1.0f, 0.0f, 0.0f);
    glVertex3f( 0.0f, 100.0f, 0.0f);

    glColor3f(0.0f, 1.0f, 0.0f);
    glVertex3f(-100.0f,-100.0f, 0.0f);

    glColor3f(0.0f, 0.0f, 1.0f);
    glVertex3f( 100.0f,-100.0f, 0.0f);
glEnd();
```



glBegin(…)

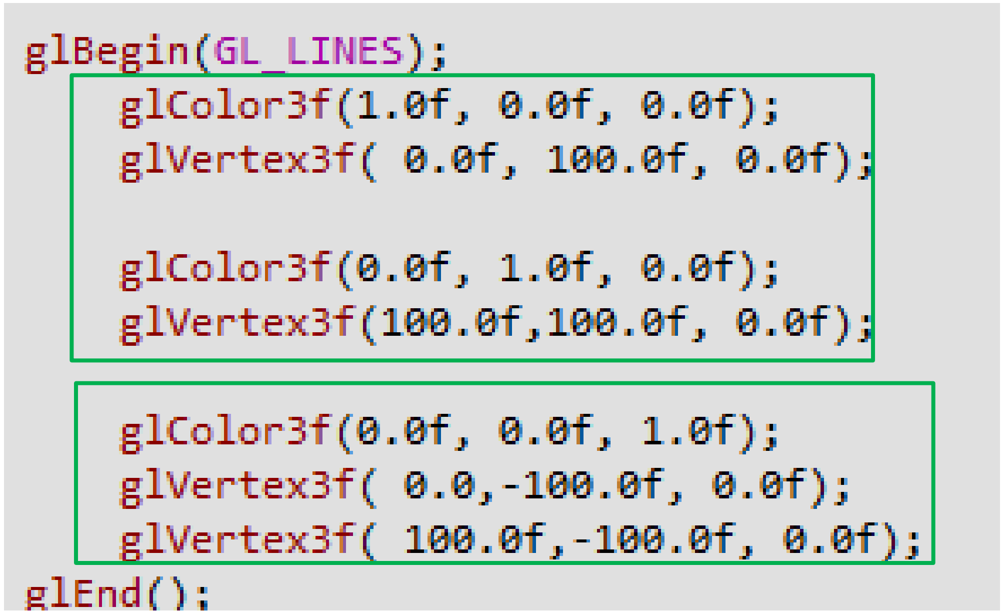
- GL_POINTS



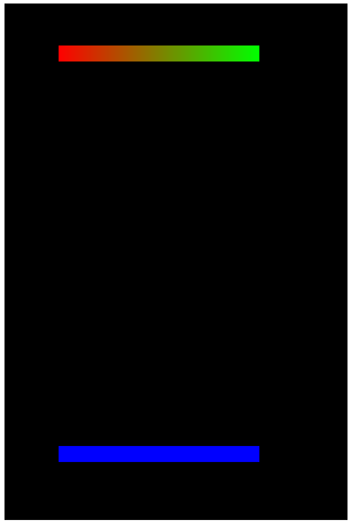
```
glBegin(GL_POINTS);
    glColor3f(1.0f, 0.0f, 0.0f);
    glVertex3f( 0.0f, 100.0f, 0.0f);

    glColor3f(0.0f, 1.0f, 0.0f);
    glVertex3f(-100.0f,-100.0f, 0.0f);

    glColor3f(0.0f, 0.0f, 1.0f);
    glVertex3f( 100.0f,-100.0f, 0.0f);
glEnd();
```

glBegin(…)

- GL_LINES

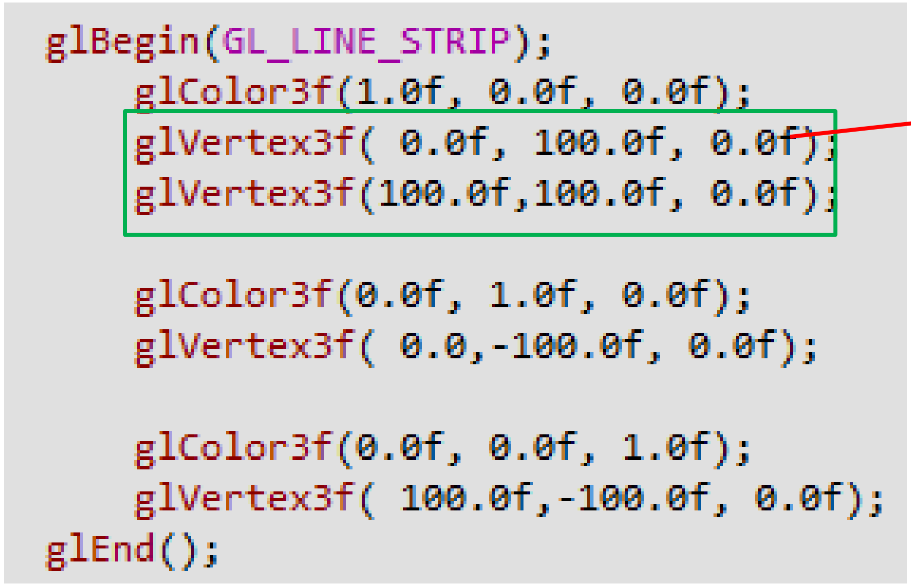


```
glBegin(GL_LINES);
    glColor3f(1.0f, 0.0f, 0.0f);
    glVertex3f( 0.0f, 100.0f, 0.0f);

    glColor3f(0.0f, 1.0f, 0.0f);
    glVertex3f(100.0f,100.0f, 0.0f);

    glColor3f(0.0f, 0.0f, 1.0f);
    glVertex3f( 0.0,-100.0f, 0.0f);
    glVertex3f( 100.0f,-100.0f, 0.0f);
glEnd();
```

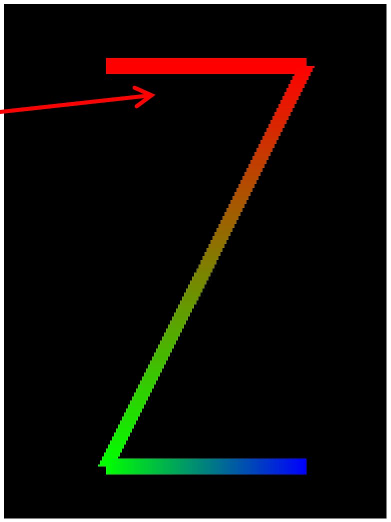


glBegin(…)

- GL_LINE_STRIP

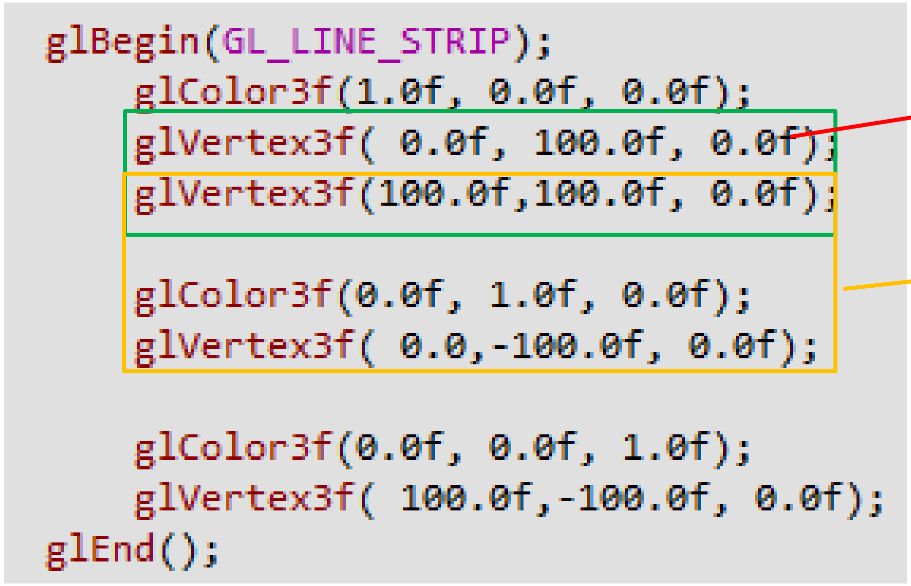


```
glBegin(GL_LINE_STRIP);
    glColor3f(1.0f, 0.0f, 0.0f);
    glVertex3f( 0.0f, 100.0f, 0.0f);
    glVertex3f(100.0f,100.0f, 0.0f);

    glColor3f(0.0f, 1.0f, 0.0f);
    glVertex3f( 0.0,-100.0f, 0.0f);

    glColor3f(0.0f, 0.0f, 1.0f);
    glVertex3f( 100.0f,-100.0f, 0.0f);
glEnd();
```

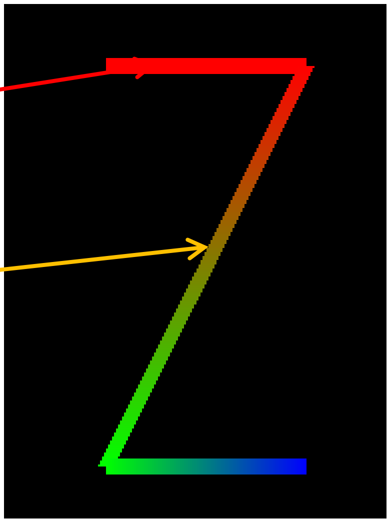


glBegin(…)

- GL_LINE_STRIP

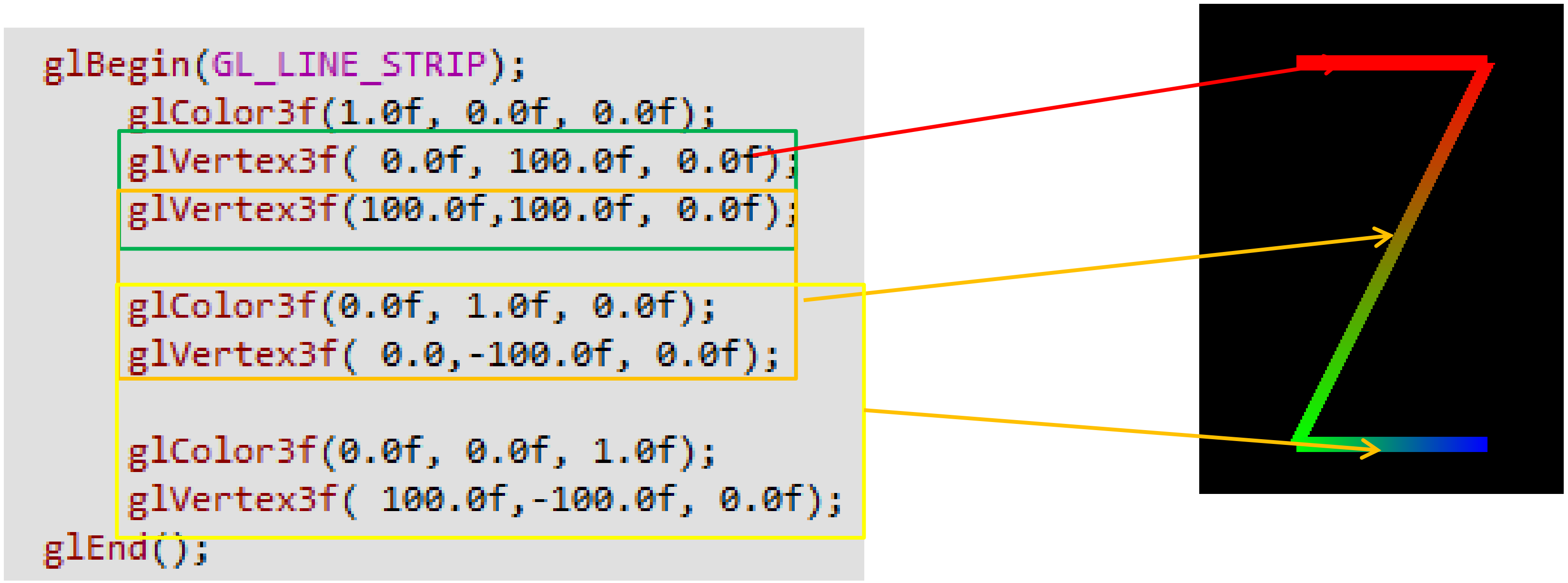


```
glBegin(GL_LINE_STRIP);
    glColor3f(1.0f, 0.0f, 0.0f);
    glVertex3f( 0.0f, 100.0f, 0.0f);
    glVertex3f(100.0f,100.0f, 0.0f);

    glColor3f(0.0f, 1.0f, 0.0f);
    glVertex3f( 0.0,-100.0f, 0.0f);

    glColor3f(0.0f, 0.0f, 1.0f);
    glVertex3f( 100.0f,-100.0f, 0.0f);
glEnd();
```



glBegin(…)

- GL_LINE_STRIP

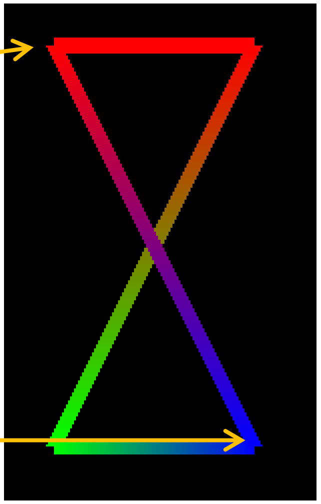


```
glBegin(GL_LINE_STRIP);
    glColor3f(1.0f, 0.0f, 0.0f);
    glVertex3f( 0.0f, 100.0f, 0.0f);
    glVertex3f(100.0f,100.0f, 0.0f);

    glColor3f(0.0f, 1.0f, 0.0f);
    glVertex3f( 0.0,-100.0f, 0.0f);

    glColor3f(0.0f, 0.0f, 1.0f);
    glVertex3f( 100.0f,-100.0f, 0.0f);
glEnd();
```

glBegin(…)

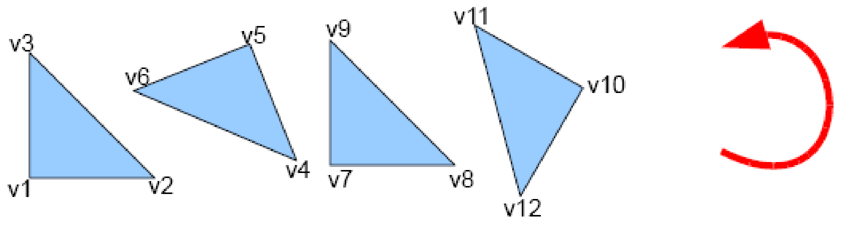
- GL_LINE_LOOP

```
glBegin(GL_LINE_LOOP);
    glColor3f(1.0f, 0.0f, 0.0f);
    glVertex3f( 0.0f, 100.0f, 0.0f);
    glVertex3f(100.0f,100.0f, 0.0f);

    glColor3f(0.0f, 1.0f, 0.0f);
    glVertex3f( 0.0,-100.0f, 0.0f);

    glColor3f(0.0f, 0.0f, 1.0f);
    glVertex3f( 100.0f,-100.0f, 0.0f);
glEnd();
```

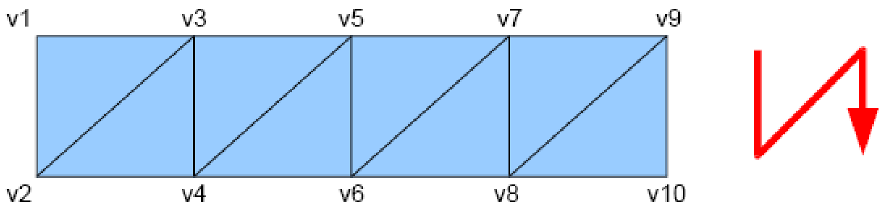


glBegin(…)

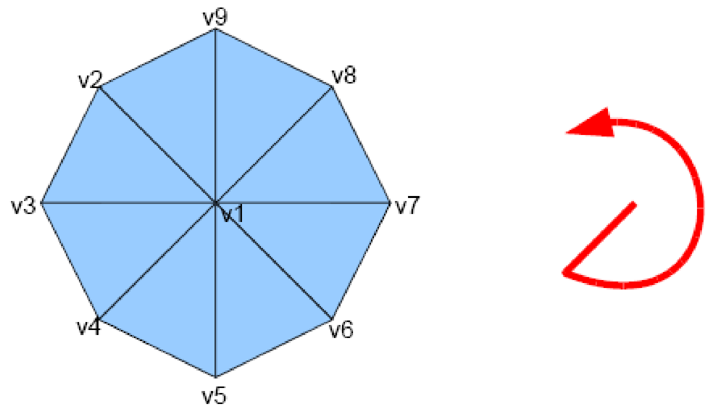
- GL_TRIANGLES



- GL_TRIANGLE_STRIP

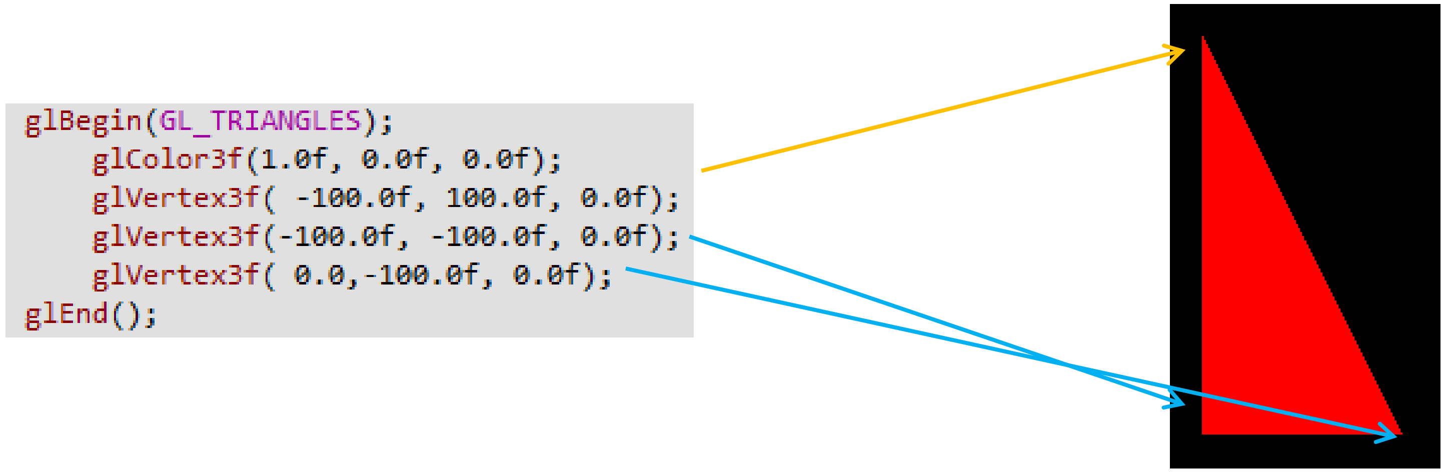


- GL_TRIANGLE_FAN



glBegin(…)

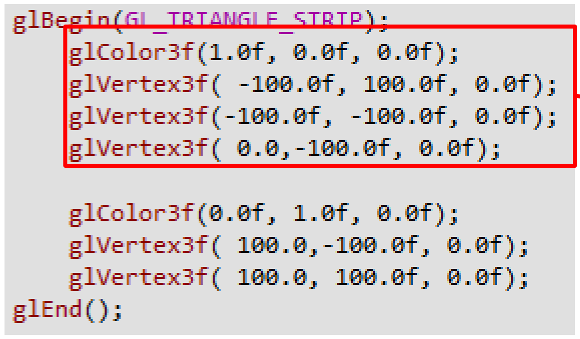
- GL_TRIANGLES



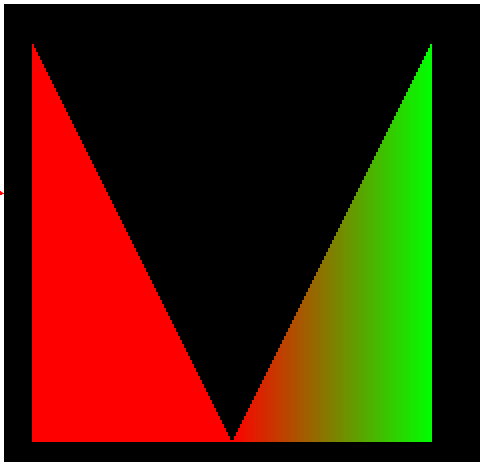
```
glBegin(GL_TRIANGLES);
    glColor3f(1.0f, 0.0f, 0.0f);
    glVertex3f( -100.0f, 100.0f, 0.0f);
    glVertex3f(-100.0f, -100.0f, 0.0f);
    glVertex3f( 0.0,-100.0f, 0.0f);
glEnd();
```

glBegin(…)

- GL_TRIANGLE_STRIP

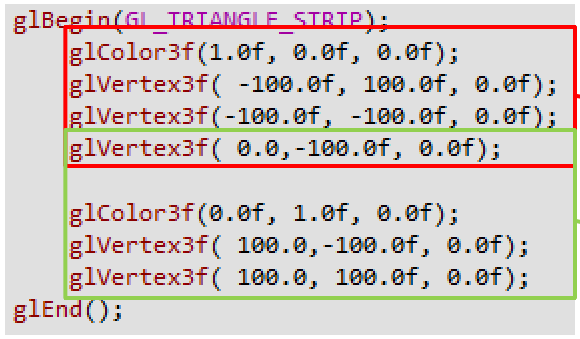


```
glBegin(GL_TRIANGLE_STRIP);
    glColor3f(1.0f, 0.0f, 0.0f);
    glVertex3f( -100.0f, 100.0f, 0.0f);
    glVertex3f(-100.0f, -100.0f, 0.0f);
    glVertex3f( 0.0,-100.0f, 0.0f);

    glColor3f(0.0f, 1.0f, 0.0f);
    glVertex3f( 100.0,-100.0f, 0.0f);
    glVertex3f( 100.0, 100.0f, 0.0f);
glEnd();
```

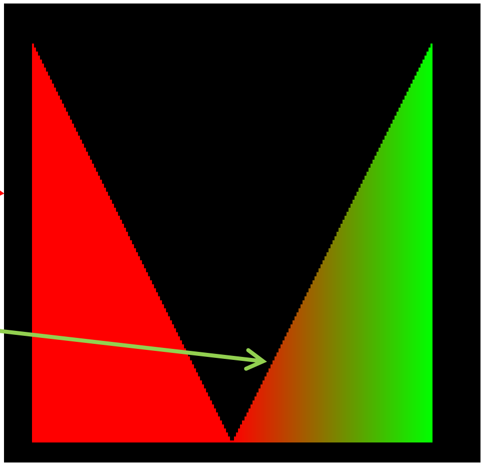


glBegin(…)

- GL_TRIANGLE_STRIP



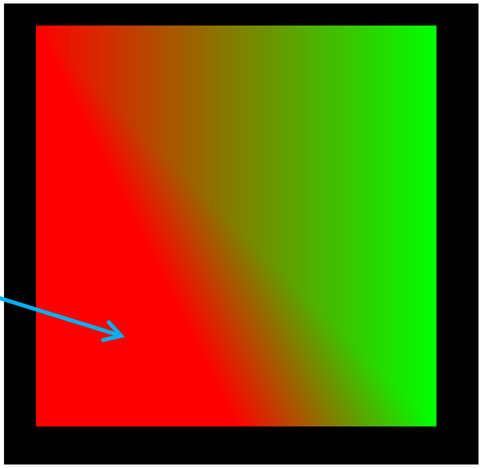
```
glBegin(GL_TRIANGLE_STRIP);
    glColor3f(1.0f, 0.0f, 0.0f);
    glVertex3f( -100.0f, 100.0f, 0.0f);
    glVertex3f(-100.0f, -100.0f, 0.0f);
    glVertex3f( 0.0,-100.0f, 0.0f);

    glColor3f(0.0f, 1.0f, 0.0f);
    glVertex3f( 100.0,-100.0f, 0.0f);
    glVertex3f( 100.0, 100.0f, 0.0f);
glEnd();
```



glBegin(…)

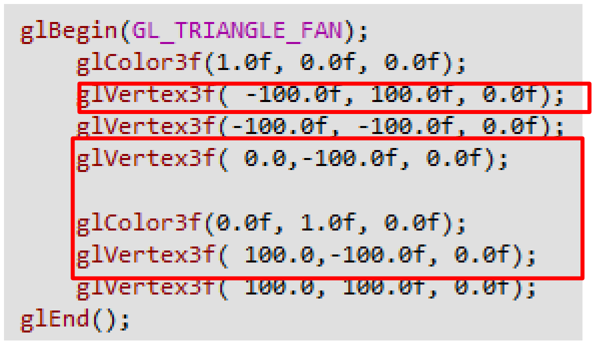
- GL_TRIANGLE_FAN

```
glBegin(GL_TRIANGLE_FAN);
    glColor3f(1.0f, 0.0f, 0.0f);
    glVertex3f( -100.0f, 100.0f, 0.0f);
    glVertex3f(-100.0f, -100.0f, 0.0f);
    glVertex3f( 0.0,-100.0f, 0.0f);

    glColor3f(0.0f, 1.0f, 0.0f);
    glVertex3f( 100.0,-100.0f, 0.0f);
    glVertex3f( 100.0, 100.0f, 0.0f);
glEnd();
```

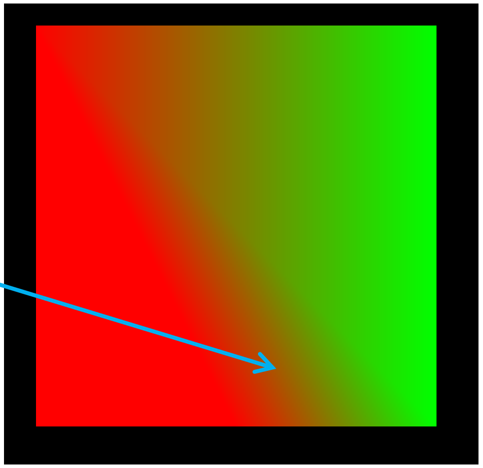


glBegin(…)

- GL_TRIANGLE_FAN

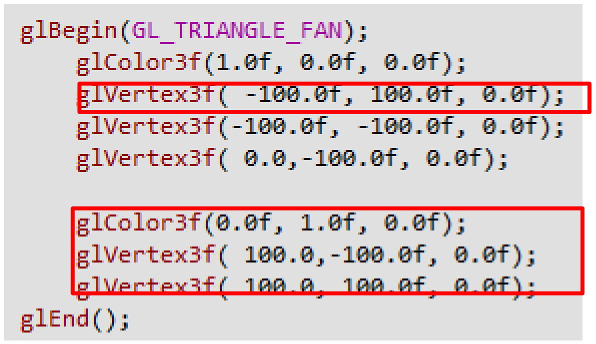


```
glBegin(GL_TRIANGLE_FAN);
    glColor3f(1.0f, 0.0f, 0.0f);
    glVertex3f( -100.0f, 100.0f, 0.0f);
    glVertex3f(-100.0f, -100.0f, 0.0f);
    glVertex3f( 0.0,-100.0f, 0.0f);

    glColor3f(0.0f, 1.0f, 0.0f);
    glVertex3f( 100.0,-100.0f, 0.0f);
    glVertex3f( 100.0, 100.0f, 0.0f);
glEnd();
```

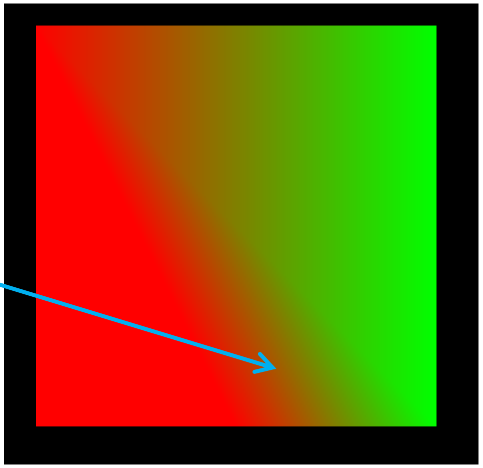


glBegin(…)

- GL_TRIANGLE_FAN

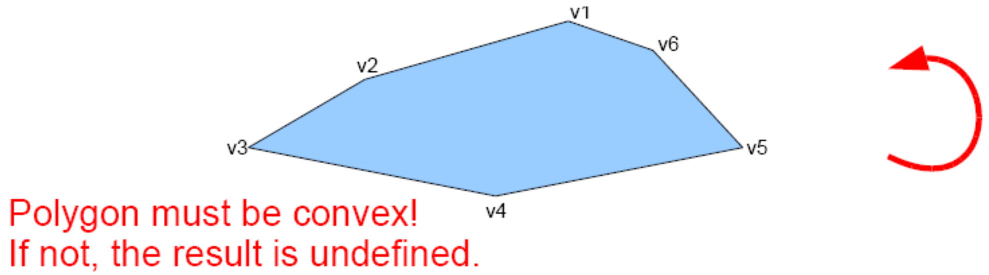


```
glBegin(GL_TRIANGLE_FAN);
    glColor3f(1.0f, 0.0f, 0.0f);
    glVertex3f( -100.0f, 100.0f, 0.0f);
    glVertex3f(-100.0f, -100.0f, 0.0f);
    glVertex3f( 0.0,-100.0f, 0.0f);

    glColor3f(0.0f, 1.0f, 0.0f);
    glVertex3f( 100.0,-100.0f, 0.0f);
    glVertex3f( 100.0, 100.0f, 0.0f);
glEnd();
```

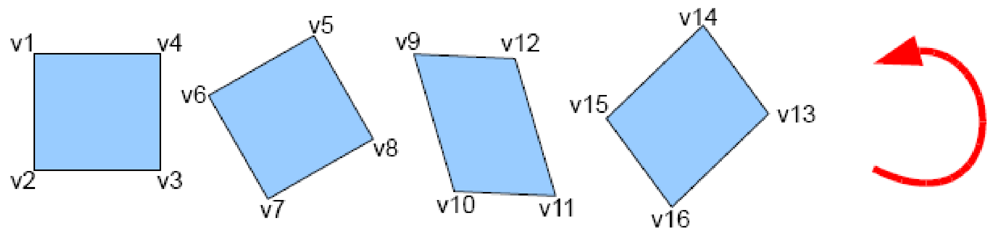


glBegin(…)

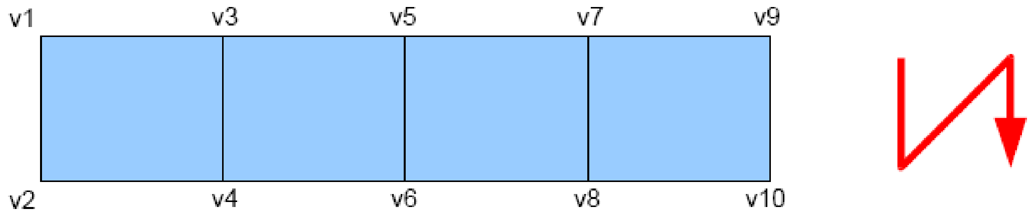
- GL_POLYGON



- GL_QUADS



- GL_QUAD_STRIP



OpenGL Basic Command

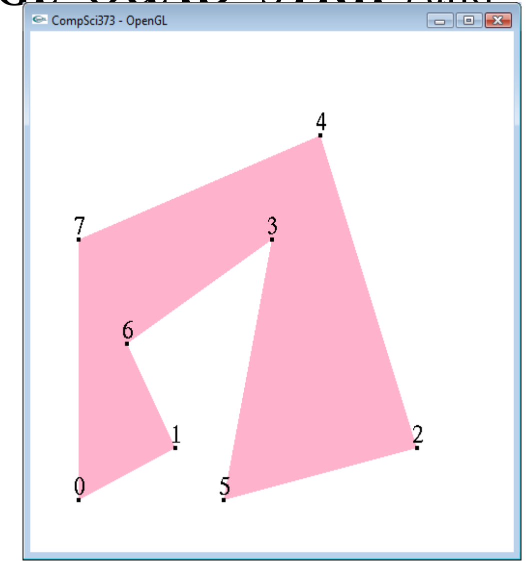
Given are the vertices

const int numVertices=8;

const float vertices[numVertices][2] =
{{50,50},{150,100},{400,100},{250,300},
{300,400},{200,50},{100, 200},{50, 300}};

Which calling sequence of these vertices (using glVertex2fv) results in the shape on the right if we use the OpenGL commands glBegin(**GL_QUAD_STRIP**) and glEnd()?

A. 0, 1, 2, 3, 4, 5, 6, 7
B. 3, 5, 2, 4, 7, 0, 1, 6
C. 2, 5, 4, 3, 7, 6, 0, 1
D. 0, 7, 1, 6, 3, 4, 5, 2
E. None of the others correct



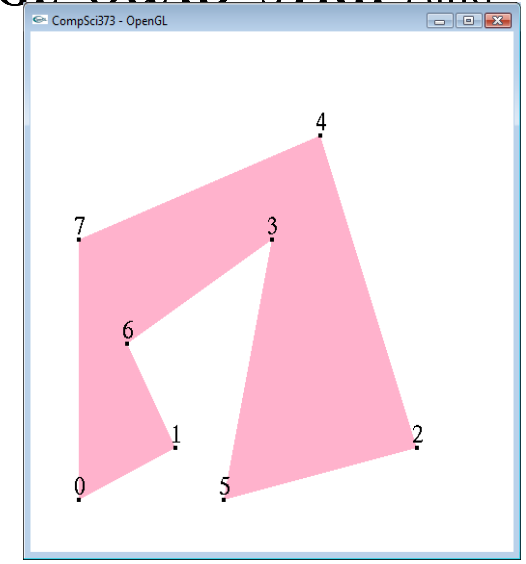
OpenGL Basic Command

Given are the vertices

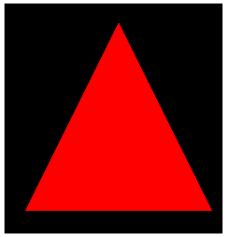
```
const int numVertices=8;
const float vertices[numVertices][2] =
        {{50,50},{150,100},{400,100},{250,300},
         {300,400},{200,50},{100, 200},{50, 300}};
```

Which calling sequence of these vertices (using glVertex2fv) results in the shape on the right if we use the OpenGL commands glBegin(**GL_QUAD_STRIP**) and glEnd()?

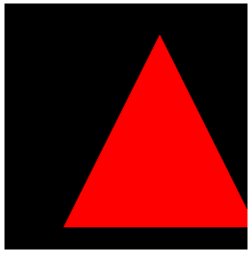
A. 0, 1, 2, 3, 4, 5, 6, 7
B. 3, 5, 2, 4, 7, 0, 1, 6
C. 2, 5, 4, 3, 7, 6, 0, 1
D. 0, 7, 1, 6, 3, 4, 5, 2
E. None of the others correct

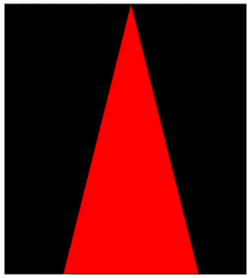


Transformations



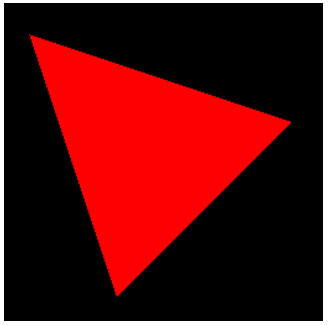
- **Translation**
  - glTranslatef(xUnits, yUnits, zUnits);
  - Ex: glTranslatef(100.0f, 0.0f, 0.0f);





- **Scaling**
  - glScalef(xUnits, yUnits, zUnits);
  - Ex: glScalef(1, 2, 1);

- **Rotation**
  - glRotatef(degree, xAxis, yAxis, zAxis);
  - Ex: glRotatef(45, 0.0f, 0.0f, 1.0f);



Transformations

OpenGL's transformations are done with *right multiply* of matrices, which is the opposite order of the operation applied:

```
glRotatef(...);
glTranslatef(...);

= R * T
⇒First Translate THEN Rotate
```

Parametric Curves and Surfaces

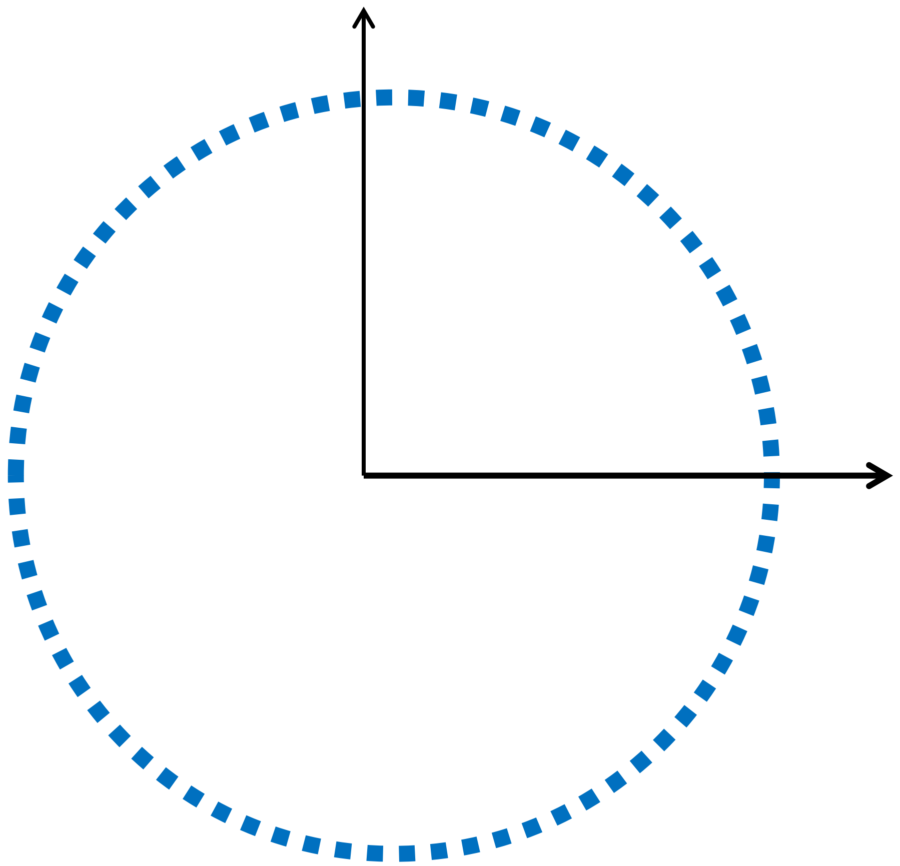
Describe the x, y, z coordinates by other parameters:

2D curve: $$p(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix}$$

where x(t) and y(t) are simple functions t goes from 0 to 1

Parametric Curves and Surfaces

Ex: Circle

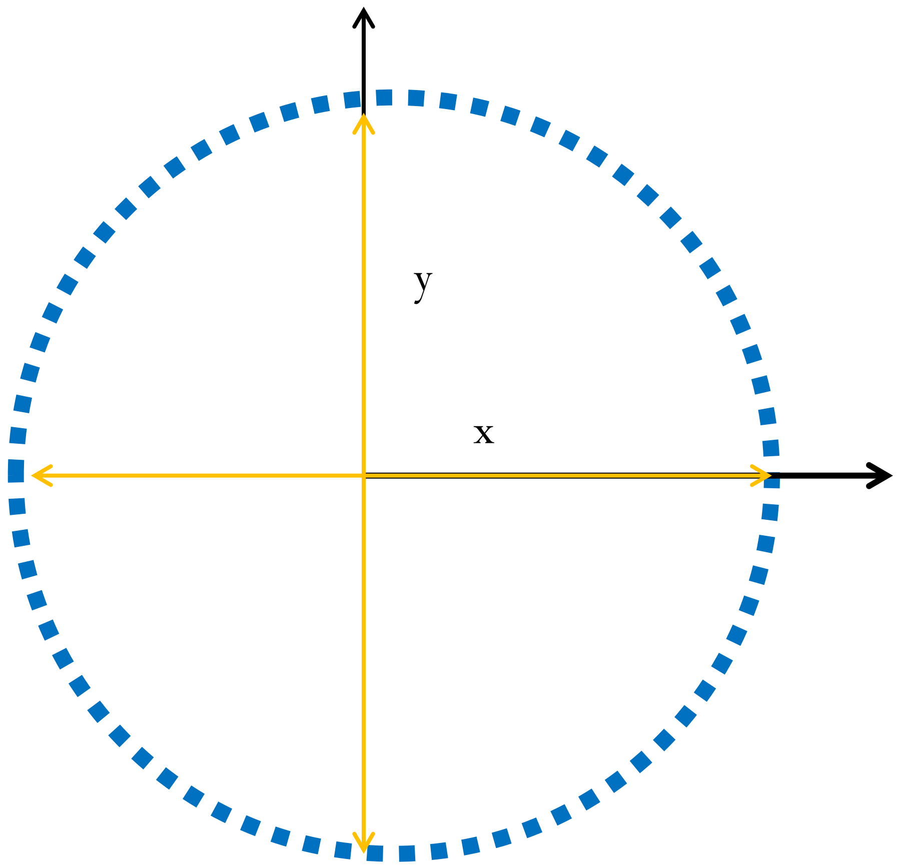


$$p(t) = \begin{pmatrix} f_x(t) \\ f_y(t) \end{pmatrix}$$

$$0 \le t \le 1$$

Parametric Curves and Surfaces

Ex: Circle

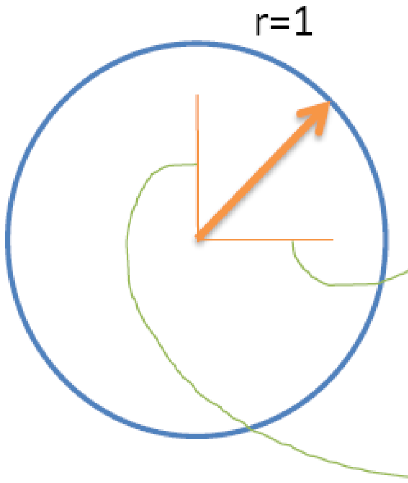


$$p(t) = \begin{pmatrix} f_x(t) \\ f_y(t) \end{pmatrix}$$

$$0 \le t \le 1$$

Parametric Curves and Surfaces

Ex: Circle



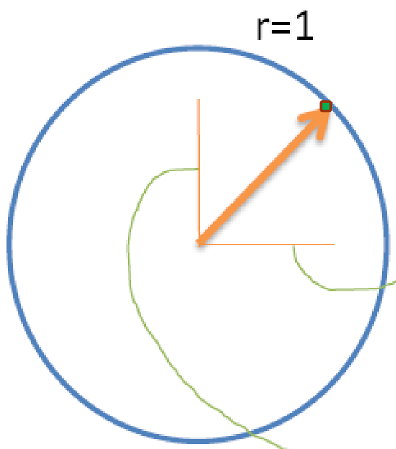
$$p(t) = \begin{pmatrix} \cos(2\pi t) \\ \sin(2\pi t) \end{pmatrix}$$

We can compute the tangent and normal at any point

- Tangent: $$p'(t) = \begin{pmatrix} \left(\frac{\partial x}{\partial t}\right) \\ \left(\frac{\partial y}{\partial t}\right) \end{pmatrix} = \begin{pmatrix} -2\pi \sin(2\pi t) \\ 2\pi \cos(2\pi t) \end{pmatrix}$$
- Normal is perpendicular to tangent

Parametric Curves and Surfaces

Ex: Circle



$$p(t) = \begin{pmatrix} \cos(2\pi t) \\ \sin(2\pi t) \end{pmatrix}$$

We can compute the tangent and normal at any point

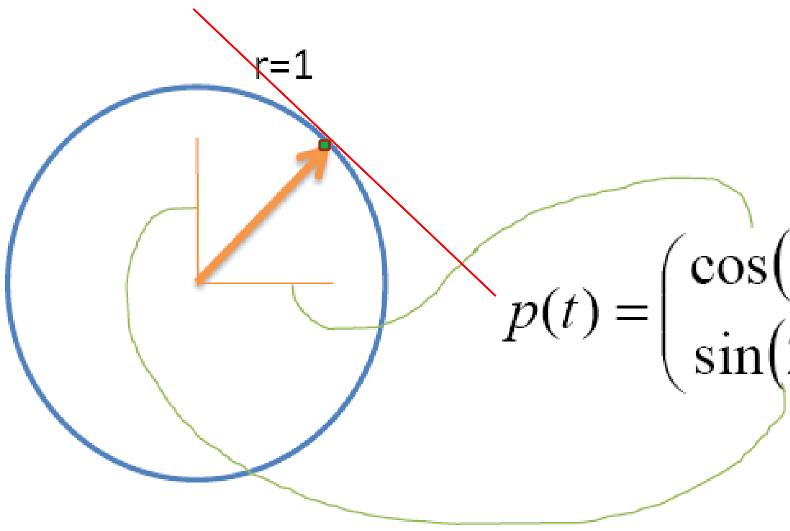
- Tangent:

$$p'(t) = \begin{pmatrix} \left(\dfrac{\partial x}{\partial t}\right) \\ \left(\dfrac{\partial y}{\partial t}\right) \end{pmatrix} = \begin{pmatrix} -2\pi \sin(2\pi t) \\ 2\pi \cos(2\pi t) \end{pmatrix}$$

- Normal is perpendicular to tangent

Parametric Curves and Surfaces

Ex: Circle



$$p(t) = \begin{pmatrix} \cos(2\pi t) \\ \sin(2\pi t) \end{pmatrix}$$

We can compute the tangent and normal at any point

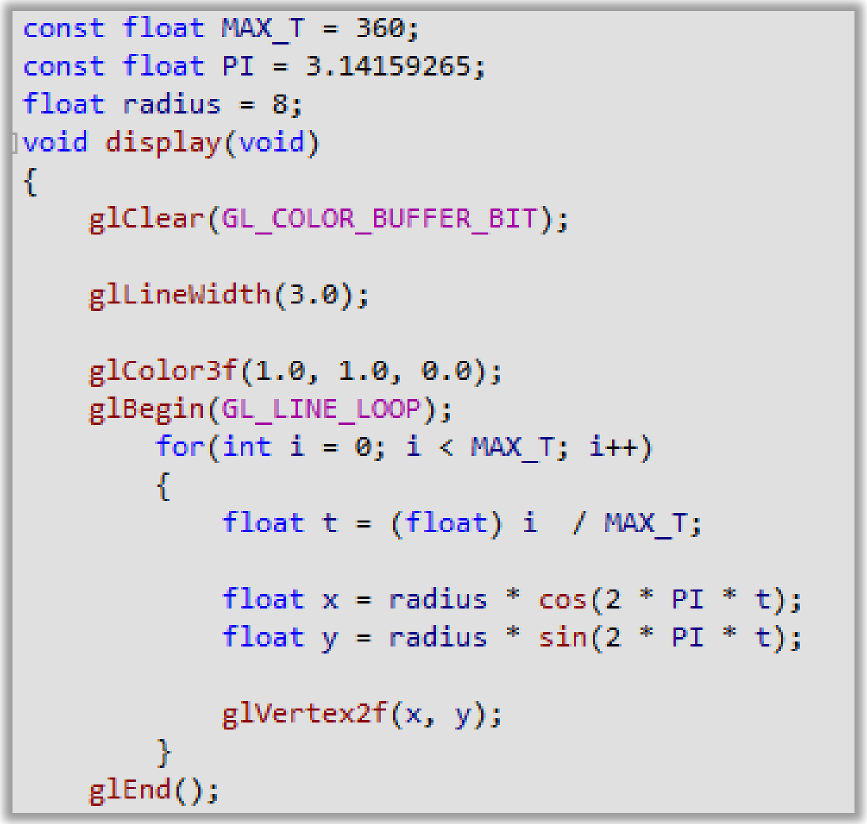
- Tangent:

$$p'(t) = \begin{pmatrix} \left(\frac{\partial x}{\partial t}\right) \\ \left(\frac{\partial y}{\partial t}\right) \end{pmatrix} = \begin{pmatrix} -2\pi \sin(2\pi t) \\ 2\pi \cos(2\pi t) \end{pmatrix}$$

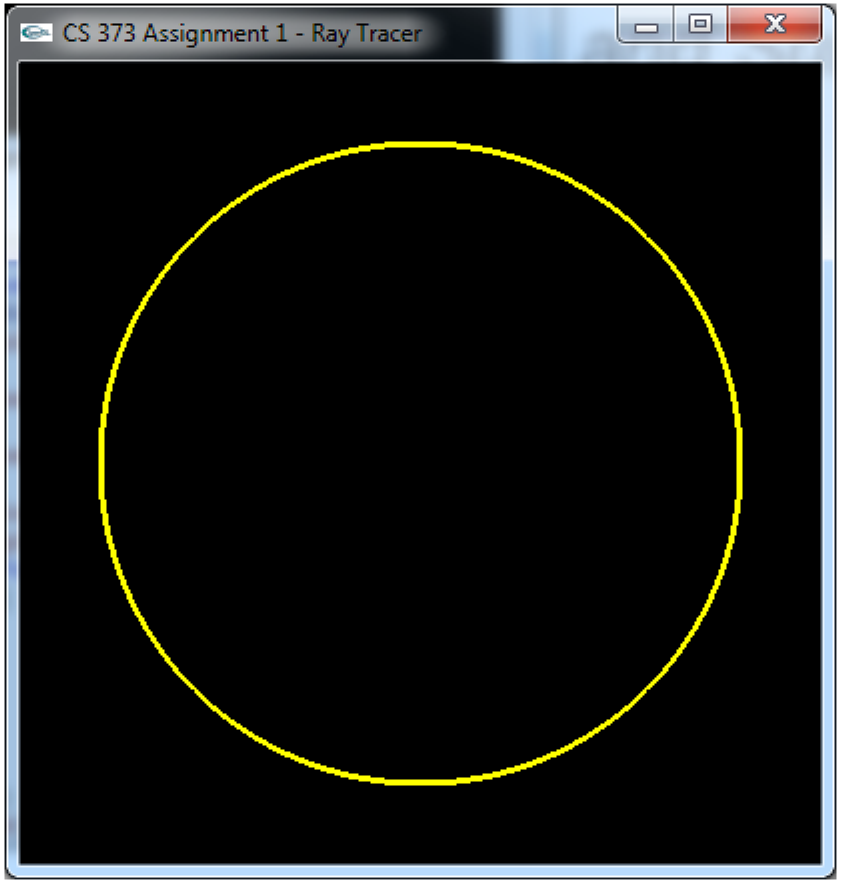
- Normal is perpendicular to tangent

Parametric Curves and Surfaces

Ex: Circle

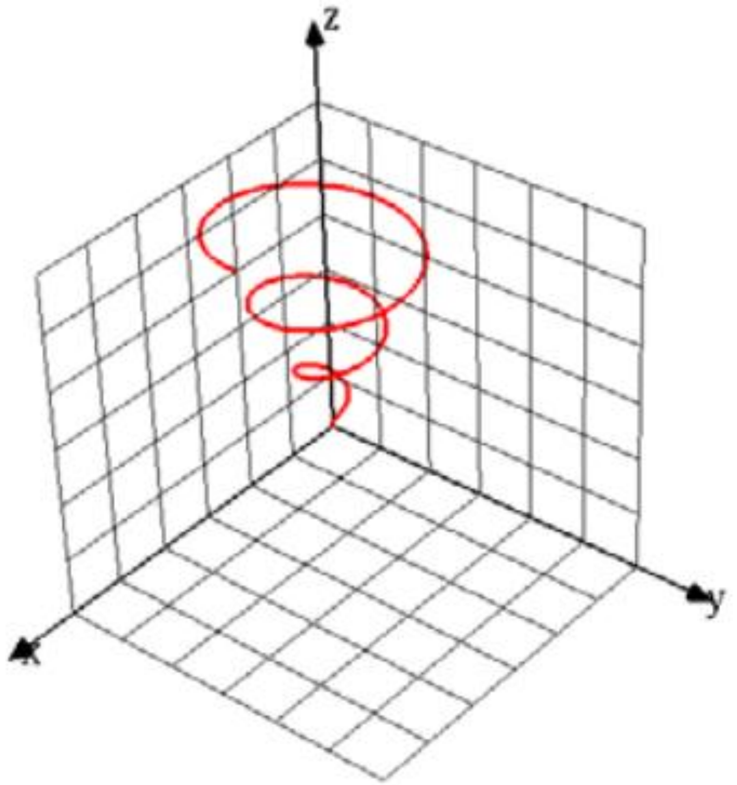


```
const float MAX_T = 360;
const float PI = 3.14159265;
float radius = 8;
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glLineWidth(3.0);

    glColor3f(1.0, 1.0, 0.0);
    glBegin(GL_LINE_LOOP);
        for(int i = 0; i < MAX_T; i++)
        {
            float t = (float) i  / MAX_T;

            float x = radius * cos(2 * PI * t);
            float y = radius * sin(2 * PI * t);

            glVertex2f(x, y);
        }
    glEnd();
```



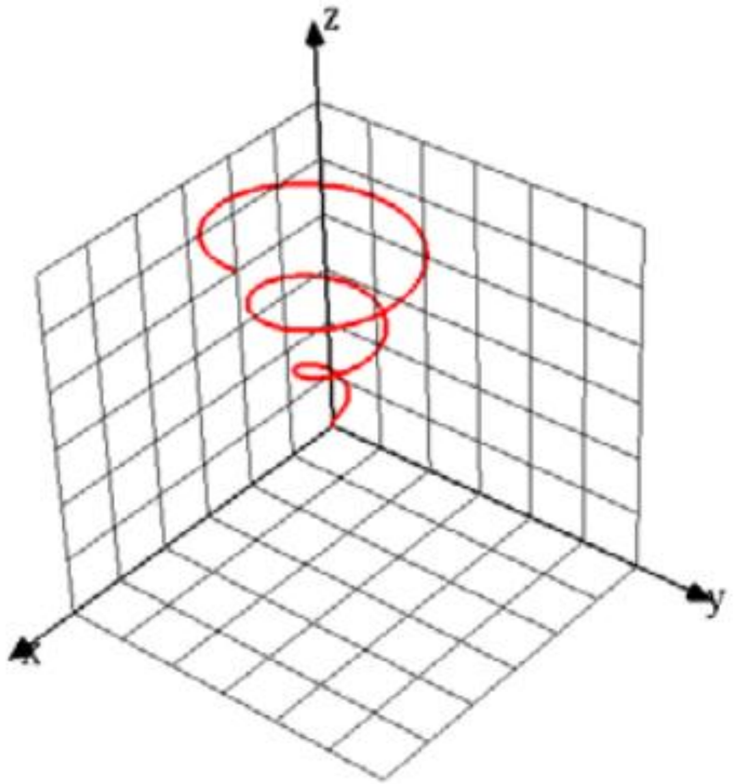
The picture on the right shows a parametric curve which is a spiral with 3 revolutions, a height of 4 units, and a radius rising from 0 to 2 units. The spiral starts at the origin and its centre axis is the z-axis.

What is the parametric equation of this curve?



The picture on the right shows a parametric curve which is a spiral with 3 revolutions, a height of 4 units, and a radius rising from 0 to 2 units. The spiral starts at the origin and its centre axis is the z-axis.

What is the parametric equation of this curve?



(a) $\begin{pmatrix} 2\cos(3\pi t) \\ 2\sin(3\pi t) \\ 4t \end{pmatrix}$

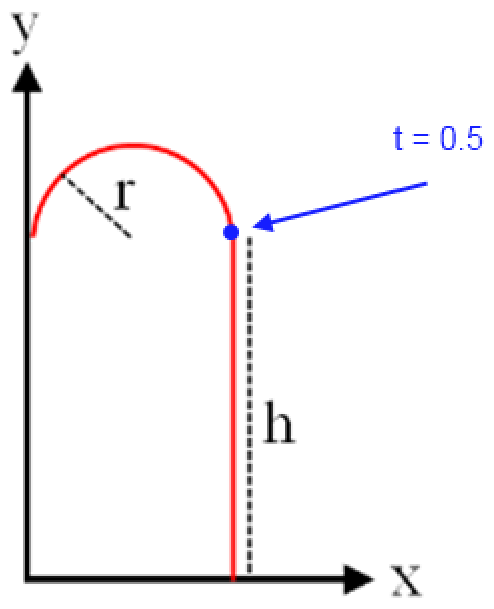
(b) $\begin{pmatrix} 3t\cos(6\pi t) \\ 3t\sin(6\pi t) \\ 4 \end{pmatrix}$

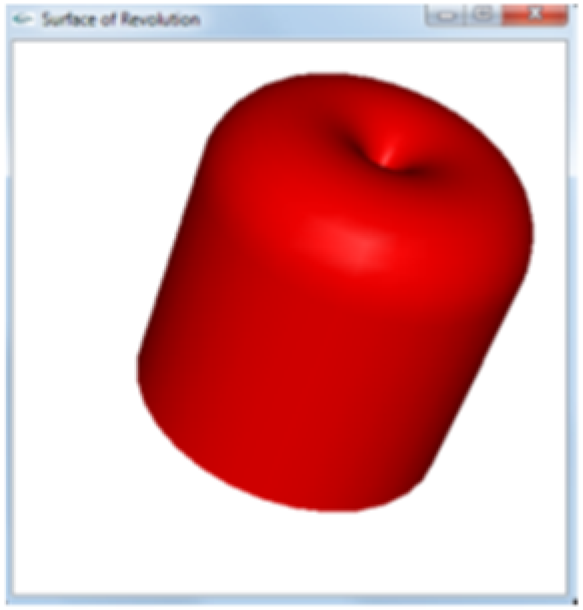
(c) $\begin{pmatrix} 2\cos(3\pi t) \\ 2\sin(3\pi t) \\ 4 \end{pmatrix}$

(d) $\begin{pmatrix} 2t\cos(6\pi t) \\ 2t\sin(6\pi t) \\ 4t \end{pmatrix}$

(e) None of the above

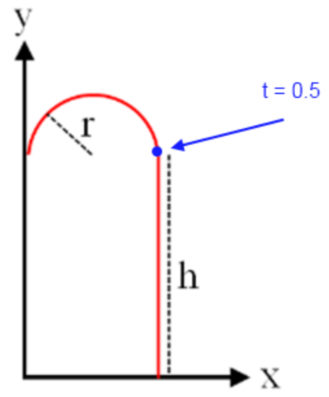
The surface of revolution in the image below on the right is created by revolving the profile curve **c**(t)=(x(t), y(t), 0) in the image below on the left around the y-axis.

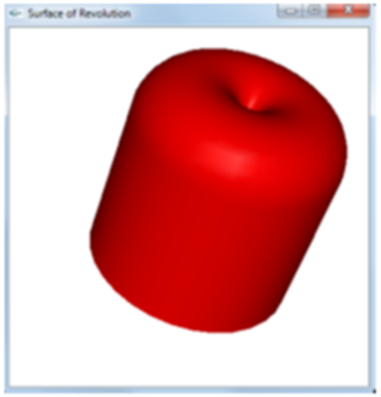




What is the equation of the x-coordinate x(t) of the profile curve above? Note that the profile curve starts on the x-axis and ends on the y-axis.

The surface of revolution in the image below on the right is created by revolving the profile curve **c**(t)=(x(t), y(t), 0) in the image below on the left around the y-axis.





(a) $x(t) = \begin{cases} 2ht & \text{if } t < 0.5 \\ r + r * \cos(2\pi(t-0.5)) & \text{if } t \geq 0.5 \end{cases}$

(b) $x(t) = \begin{cases} 2ht & \text{if } t < 0.5 \\ r + r * \sin(2\pi(t-0.5)) & \text{if } t \geq 0.5 \end{cases}$

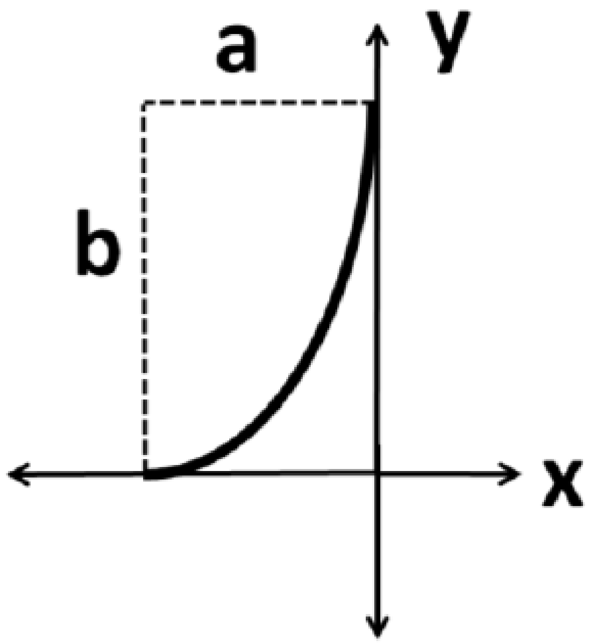
(c) $x(t) = \begin{cases} 2r & \text{if } t < 0.5 \\ r + r * \cos(2\pi(t-0.5)) & \text{if } t \geq 0.5 \end{cases}$

(d) $x(t) = \begin{cases} 2r & \text{if } t < 0.5 \\ r + r * \sin(2\pi(t-0.5)) & \text{if } t \geq 0.5 \end{cases}$

(e) None of the others

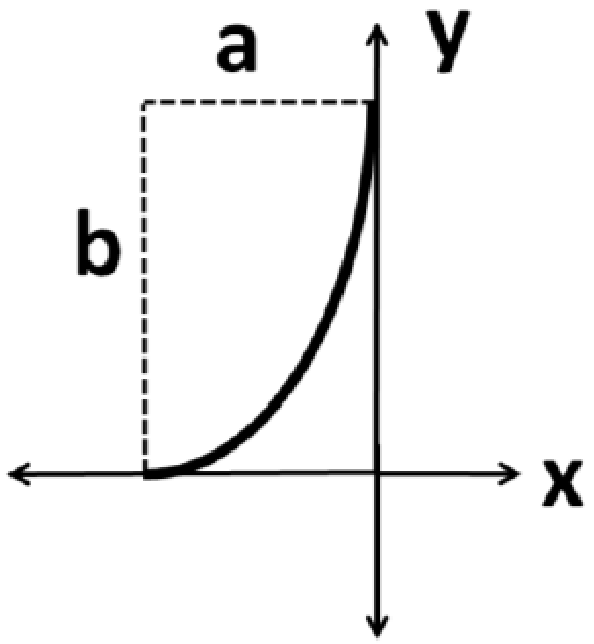
Question 46:

Which of the formulas below defines the parametric curve on the right?



Question 46:

Which of the formulas below defines the parametric curve on the right?



(a) $\begin{pmatrix} b\cos\left(\frac{1}{2}\pi + t\frac{1}{2}\pi\right) + b \\ a\,sin\left(\frac{1}{2}\pi + t\frac{1}{2}\pi\right) - a \end{pmatrix}, \quad t \in [0,1]$

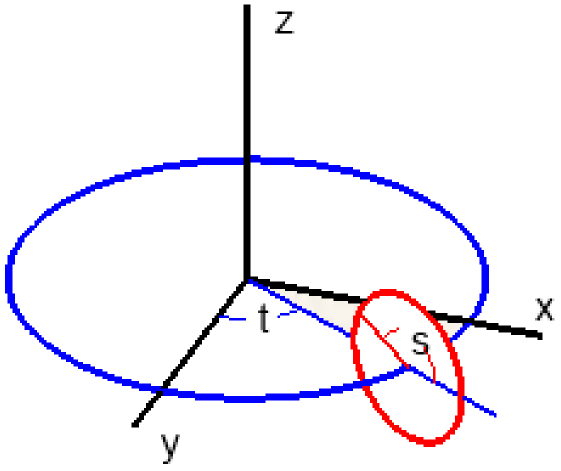
(b) $\begin{pmatrix} a\cos(2\pi t) - a \\ b\,sin(2\pi t) + b \end{pmatrix}, \quad t \in [0,1]$

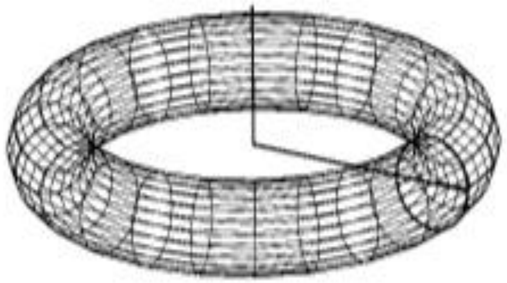
(c) $\begin{pmatrix} a\cos\left(\frac{3}{2}\pi + t\frac{1}{2}\pi\right) - a \\ b\,sin\left(\frac{3}{2}\pi + t\frac{1}{2}\pi\right) + b \end{pmatrix}, \quad t \in [0,1]$

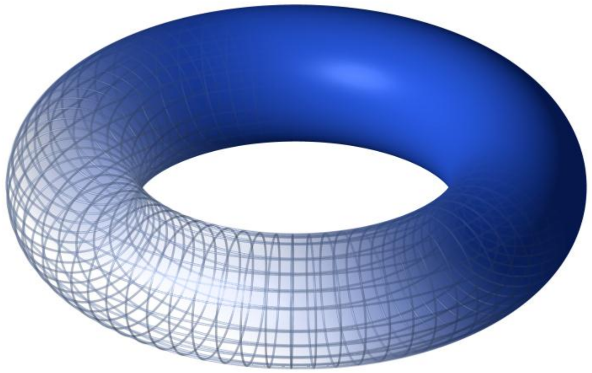
(d) $\begin{pmatrix} b\cos\left(\pi + t\frac{1}{2}\pi\right) + b \\ a\,sin\left(\pi + t\frac{1}{2}\pi\right) - a \end{pmatrix}, \quad t \in [0,1]$

Parametric Curves and Surfaces

Ex: Torus

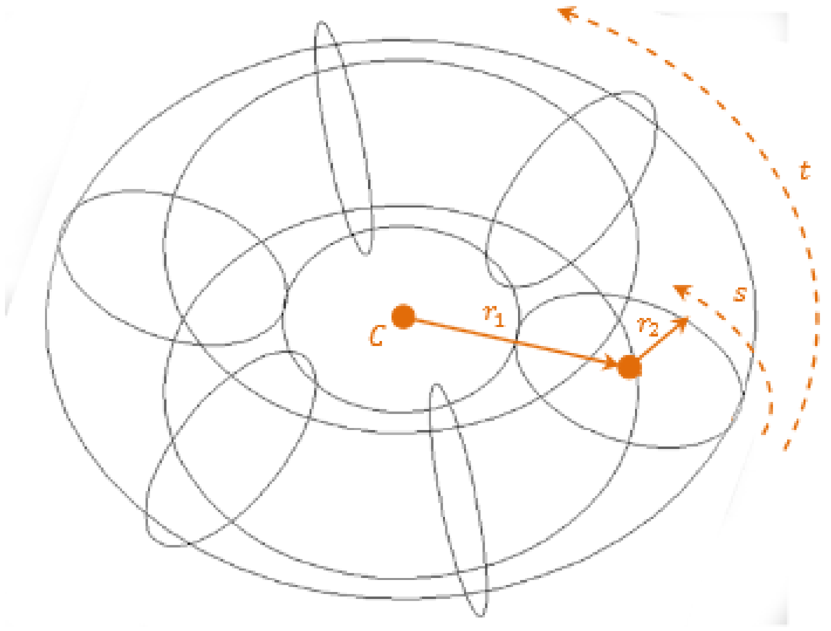






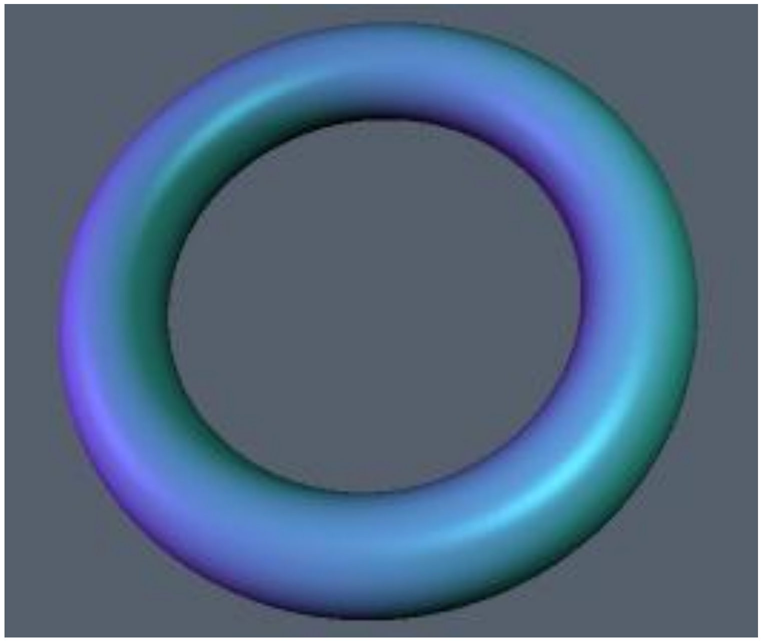
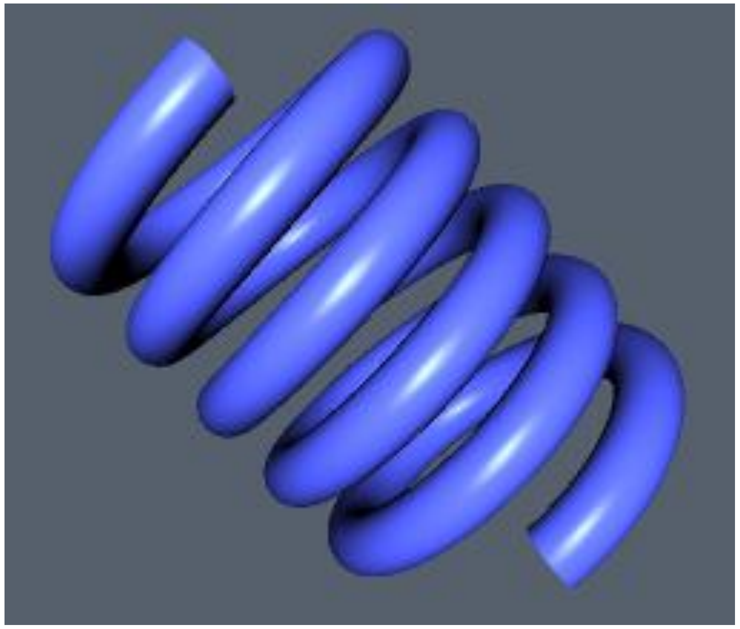
Parametric Curves and Surfaces

Ex: Torus



Parametric Curves and Surfaces

Ex: Spring

Surfaces and Curves

Given is the parametric surface $p(\boldsymbol{s},\boldsymbol{t}) = \begin{pmatrix} \boldsymbol{s^3} \\ \boldsymbol{t\sin(2\pi s)} \\ \boldsymbol{e^{-t} + s} \end{pmatrix}$. What is the normal (before normalization) at the point p(s,t)?

(a) $n(\boldsymbol{s},\boldsymbol{t}) = \begin{pmatrix} \boldsymbol{0} \\ \boldsymbol{\sin(2\pi s)} \\ \boldsymbol{-e^{-t}} \end{pmatrix} \times \begin{pmatrix} \boldsymbol{3s^2} \\ \boldsymbol{t\cos(2\pi s)2\pi} \\ \boldsymbol{1} \end{pmatrix}$

(b) $n(\boldsymbol{s},\boldsymbol{t}) = \begin{pmatrix} \boldsymbol{0} \\ \boldsymbol{\sin(2\pi s)} \\ \boldsymbol{-e^{-t}} \end{pmatrix} + \begin{pmatrix} \boldsymbol{3s^2} \\ \boldsymbol{t\cos(2\pi s)2\pi} \\ \boldsymbol{1} \end{pmatrix}$

(c) $n(\boldsymbol{s},\boldsymbol{t}) = \begin{pmatrix} \boldsymbol{0} \\ \boldsymbol{\sin(2\pi s)} \\ \boldsymbol{e^{-t}} \end{pmatrix} \times \begin{pmatrix} \boldsymbol{3s^2} \\ \boldsymbol{t\cos(2\pi s)} \\ \boldsymbol{1} \end{pmatrix}$

(d) $n(\boldsymbol{s},\boldsymbol{t}) = \begin{pmatrix} \boldsymbol{0} \\ \boldsymbol{\sin(2\pi s)} \\ \boldsymbol{e^{-t}} \end{pmatrix} + \begin{pmatrix} \boldsymbol{3s^2} \\ \boldsymbol{t\cos(2\pi s)} \\ \boldsymbol{1} \end{pmatrix}$

(e) None of the above

Surfaces and Curves

Given is the parametric surface $p(s,t) = \begin{pmatrix} s^3 \\ t\sin(2\pi s) \\ e^{-t}+s \end{pmatrix}$. What is the normal (before normalization) at the point p(s,t)?

(a) $n(s,t) = \begin{pmatrix} 0 \\ \sin(2\pi s) \\ -e^{-t} \end{pmatrix} \times \begin{pmatrix} 3s^2 \\ t\cos(2\pi s)2\pi \\ 1 \end{pmatrix}$

(b) $n(s,t) = \begin{pmatrix} 0 \\ \sin(2\pi s) \\ -e^{-t} \end{pmatrix} + \begin{pmatrix} 3s^2 \\ t\cos(2\pi s)2\pi \\ 1 \end{pmatrix}$

(c) $n(s,t) = \begin{pmatrix} 0 \\ \sin(2\pi s) \\ e^{-t} \end{pmatrix} \times \begin{pmatrix} 3s^2 \\ t\cos(2\pi s) \\ 1 \end{pmatrix}$

(d) $n(s,t) = \begin{pmatrix} 0 \\ \sin(2\pi s) \\ e^{-t} \end{pmatrix} + \begin{pmatrix} 3s^2 \\ t\cos(2\pi s) \\ 1 \end{pmatrix}$

(e) None of the above

Surfaces and Curves

Given is a parametric curve $\mathbf{p}_1(t)=(x(t), y(t), 0)$ in the xy-plane. An extruded surfac
is constructed by defining another curve $\mathbf{p}_2(t)=(x(t), y(t), z)$, where $z>0$, and connecting all points with the same t parameter.

What is the direction of the surface normal $\mathbf{n}$ at the point $\mathbf{p}=(x(t), y(t), z/2)$? I.e. which of the following vectors $\mathbf{n}$ is orthogonal to the surface at the point $\mathbf{p}$?

A. $\mathbf{n} = \begin{pmatrix} x(t) \\ y(t) \\ 1 \end{pmatrix}$

B. $\mathbf{n} = \begin{pmatrix} x'(t) \\ y'(t) \\ 0 \end{pmatrix}$

C. $\mathbf{n} = \begin{pmatrix} y'(t) \\ -x'(t) \\ 0 \end{pmatrix}$

D. $\mathbf{n} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$

E. None of the others correct

Surfaces and Curves

Given is a parametric curve $\mathbf{p}_1(t)=(x(t), y(t), 0)$ in the xy-plane. An extruded surface is constructed by defining another curve $\mathbf{p}_2(t)=(x(t), y(t), z)$, where $z > 0$, and connecting all points with the same t parameter.

What is the direction of the surface normal $\mathbf{n}$ at the point $\mathbf{p}=(x(t), y(t), z/2)$? I.e. which of the following vectors $\mathbf{n}$ is orthogonal to the surface at the point $\mathbf{p}$?

A. $\mathbf{n} = \begin{pmatrix} x(t) \\ y(t) \\ 1 \end{pmatrix}$

B. $\mathbf{n} = \begin{pmatrix} x'(t) \\ y'(t) \\ 0 \end{pmatrix}$

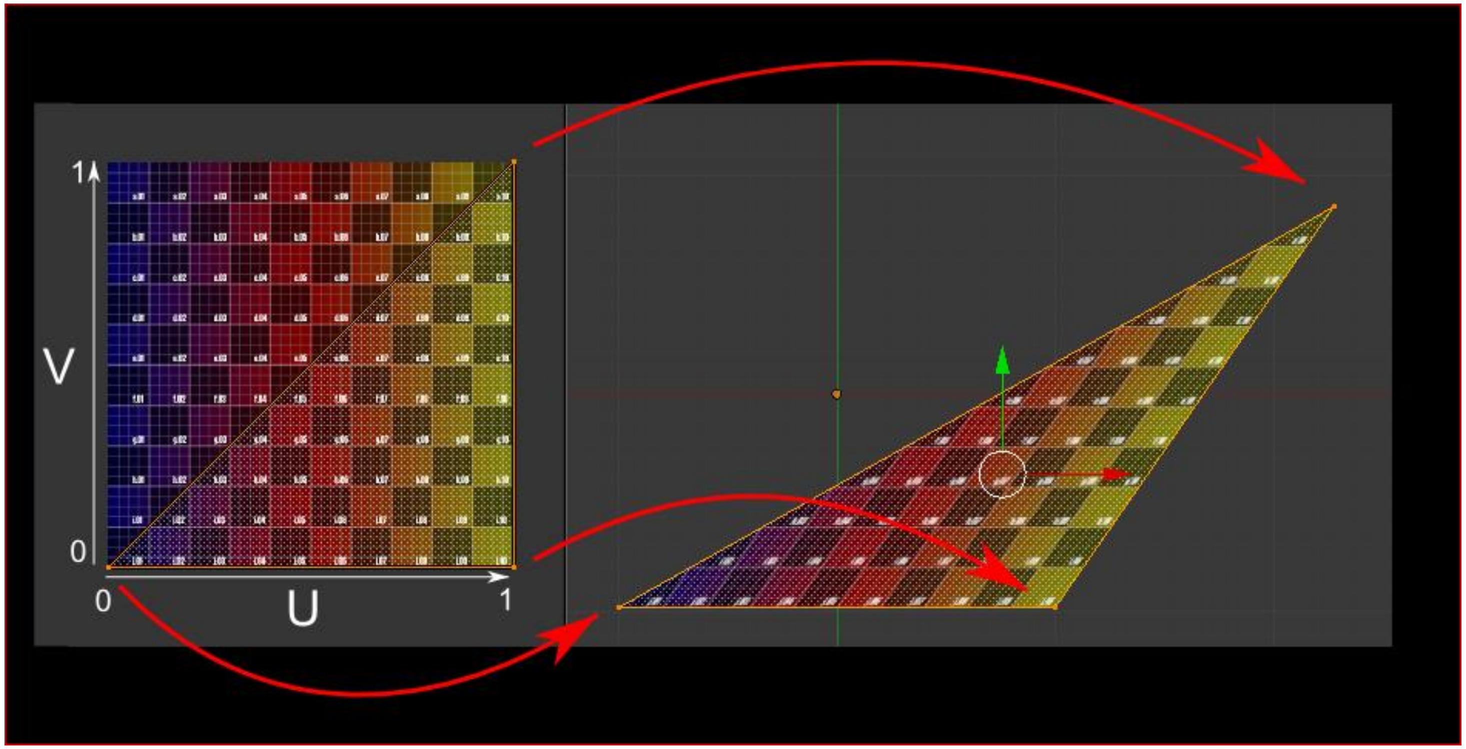
C. $\mathbf{n} = \begin{pmatrix} y'(t) \\ -x'(t) \\ 0 \end{pmatrix}$

D. $\mathbf{n} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$

E. None of the others correct

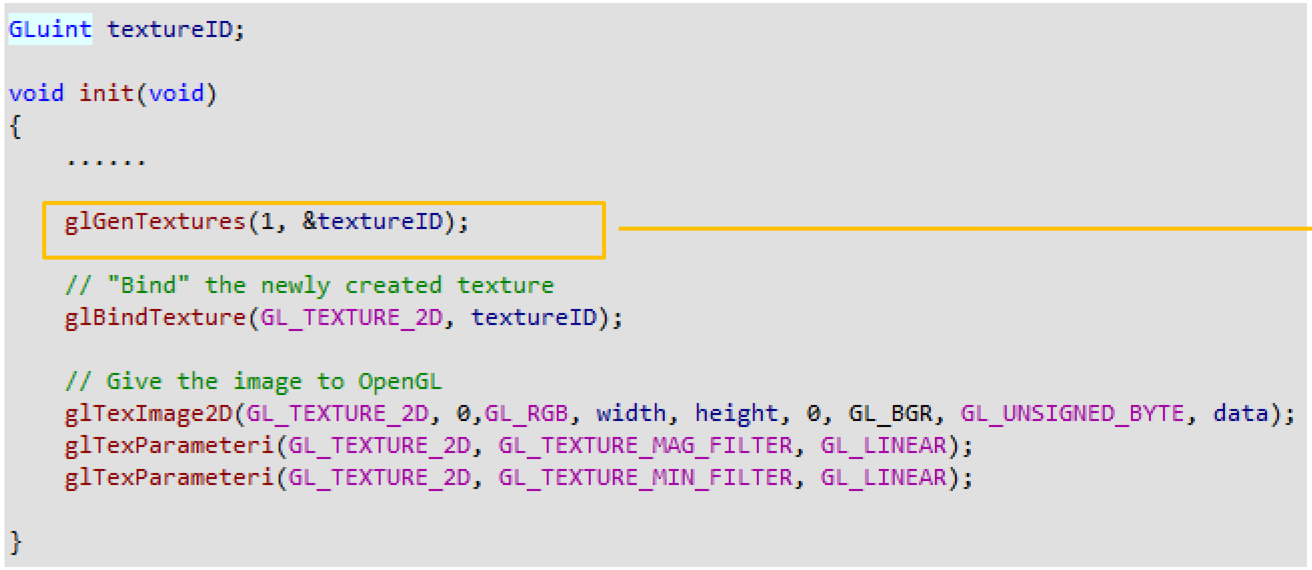
Texture Mapping

When texturing a mesh, you need a way to tell to OpenGL which part of the image has to be used for each triangle. This is done with UV coordinates



Texture Mapping

Must Setup Texture Before You can use it

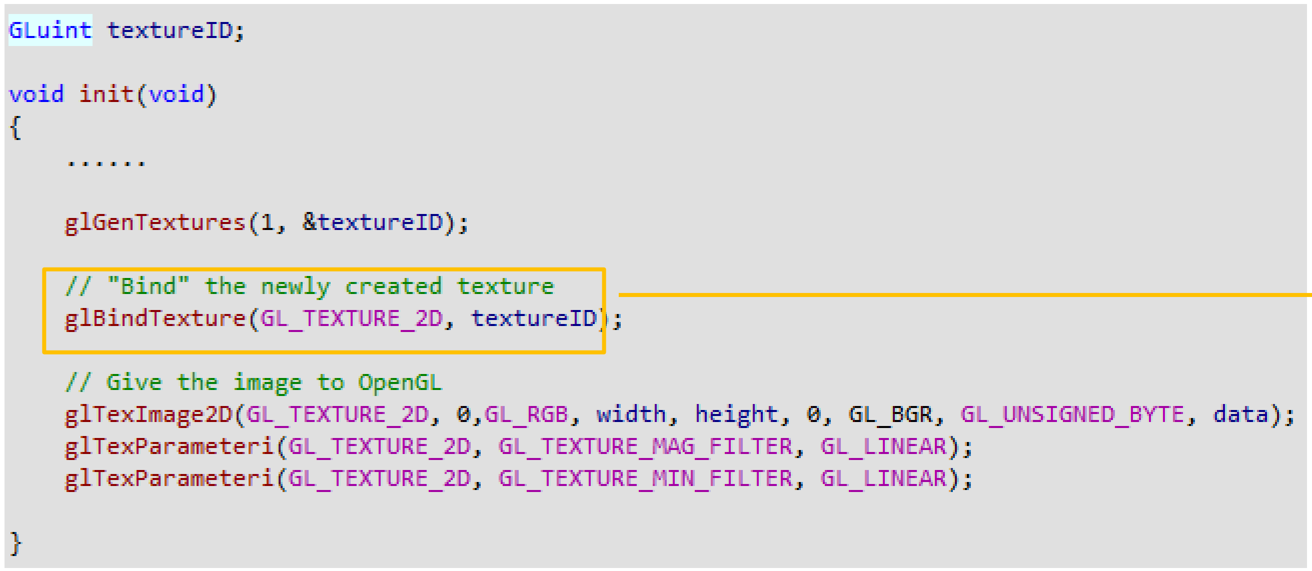


```
GLuint textureID;

void init(void)
{
    ......

    glGenTextures(1, &textureID);

    // "Bind" the newly created texture
    glBindTexture(GL_TEXTURE_2D, textureID);

    // Give the image to OpenGL
    glTexImage2D(GL_TEXTURE_2D, 0,GL_RGB, width, height, 0, GL_BGR, GL_UNSIGNED_BYTE, data);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

}
```

Tells OpenGL we want to generate one texture name

Texture Mapping

Must Setup Texture Before You can use it

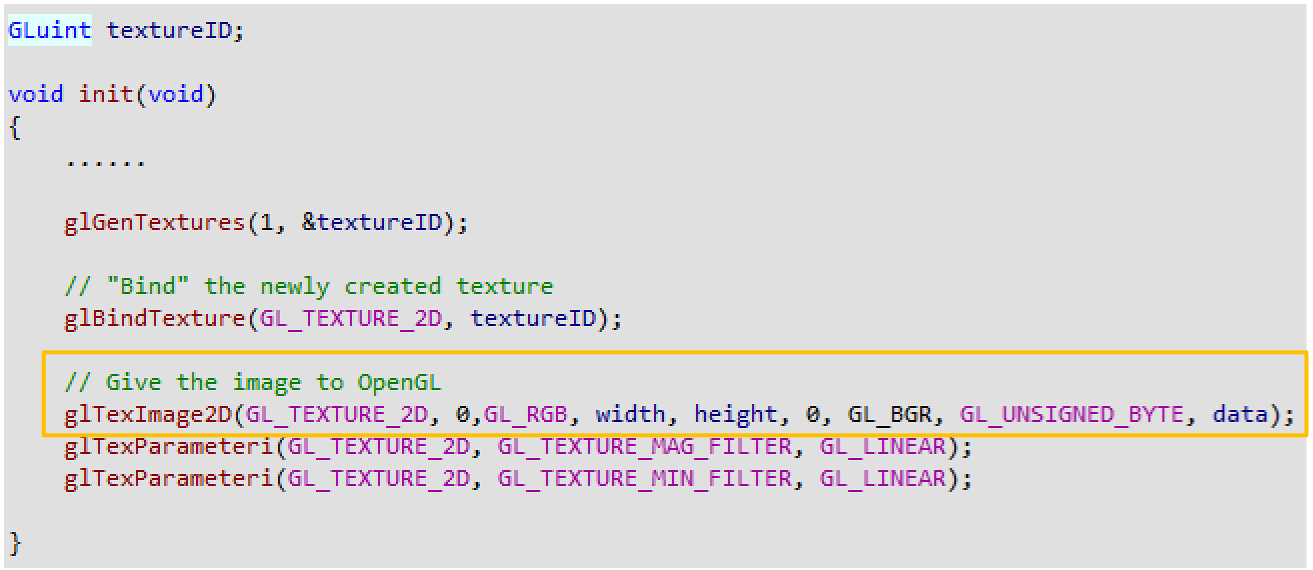


```
GLuint textureID;

void init(void)
{
    ......

    glGenTextures(1, &textureID);

    // "Bind" the newly created texture
    glBindTexture(GL_TEXTURE_2D, textureID);

    // Give the image to OpenGL
    glTexImage2D(GL_TEXTURE_2D, 0,GL_RGB, width, height, 0, GL_BGR, GL_UNSIGNED_BYTE, data);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

}
```

Tells OpenGL we are going to use texture with the given ID

Texture Mapping

Must Setup Texture Before You can use it

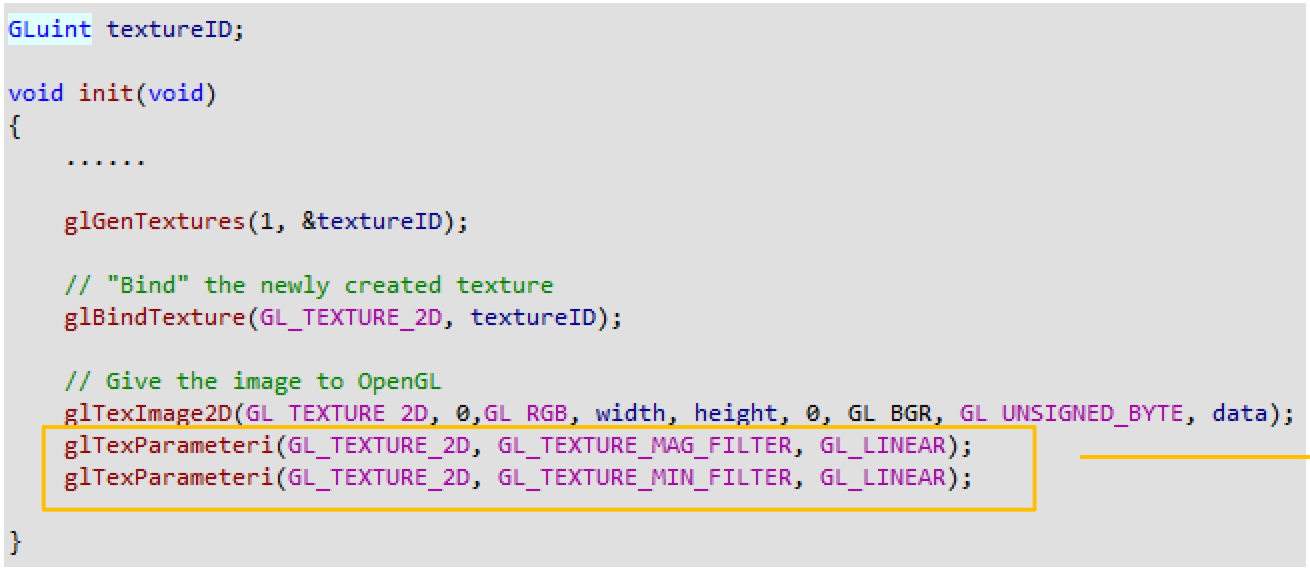


```
GLuint textureID;

void init(void)
{
    ......

    glGenTextures(1, &textureID);

    // "Bind" the newly created texture
    glBindTexture(GL_TEXTURE_2D, textureID);

    // Give the image to OpenGL
    glTexImage2D(GL_TEXTURE_2D, 0,GL_RGB, width, height, 0, GL_BGR, GL_UNSIGNED_BYTE, data);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

}
```

→ Create the actual texture

Texture Mapping

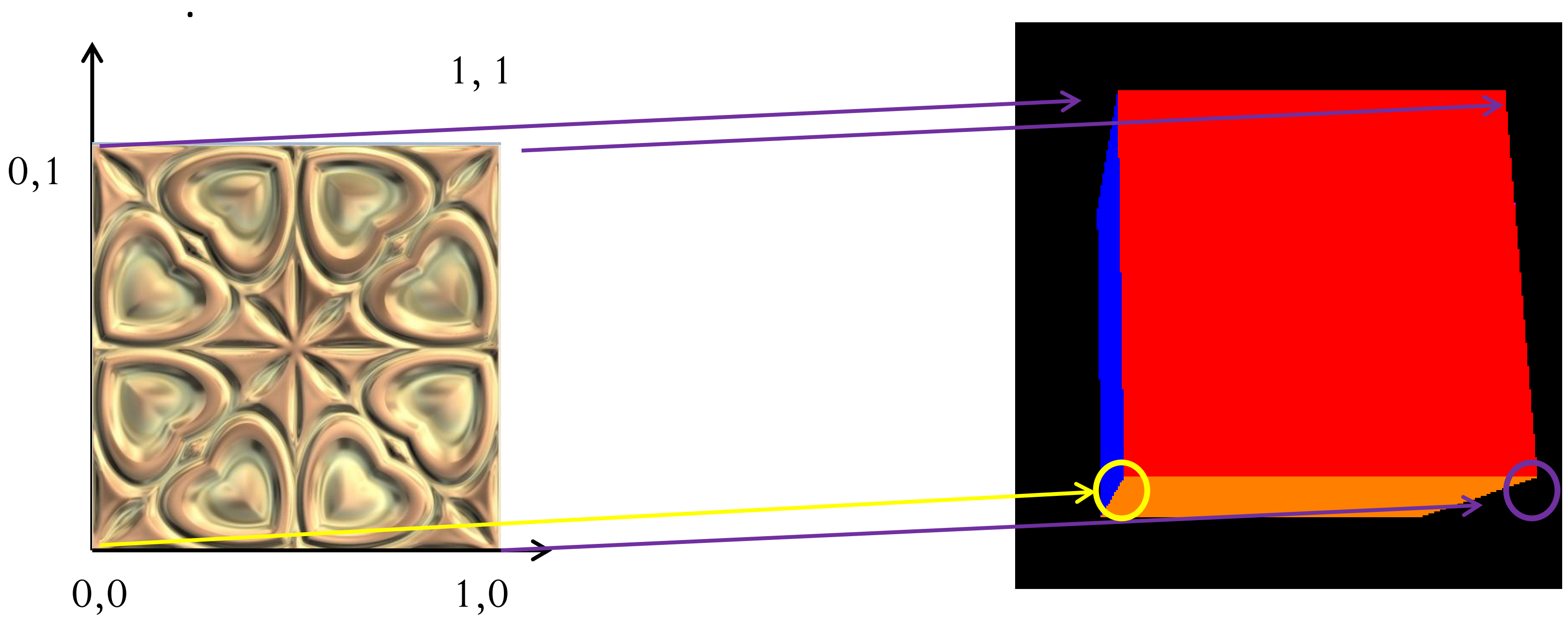
Must Setup Texture Before You can use it



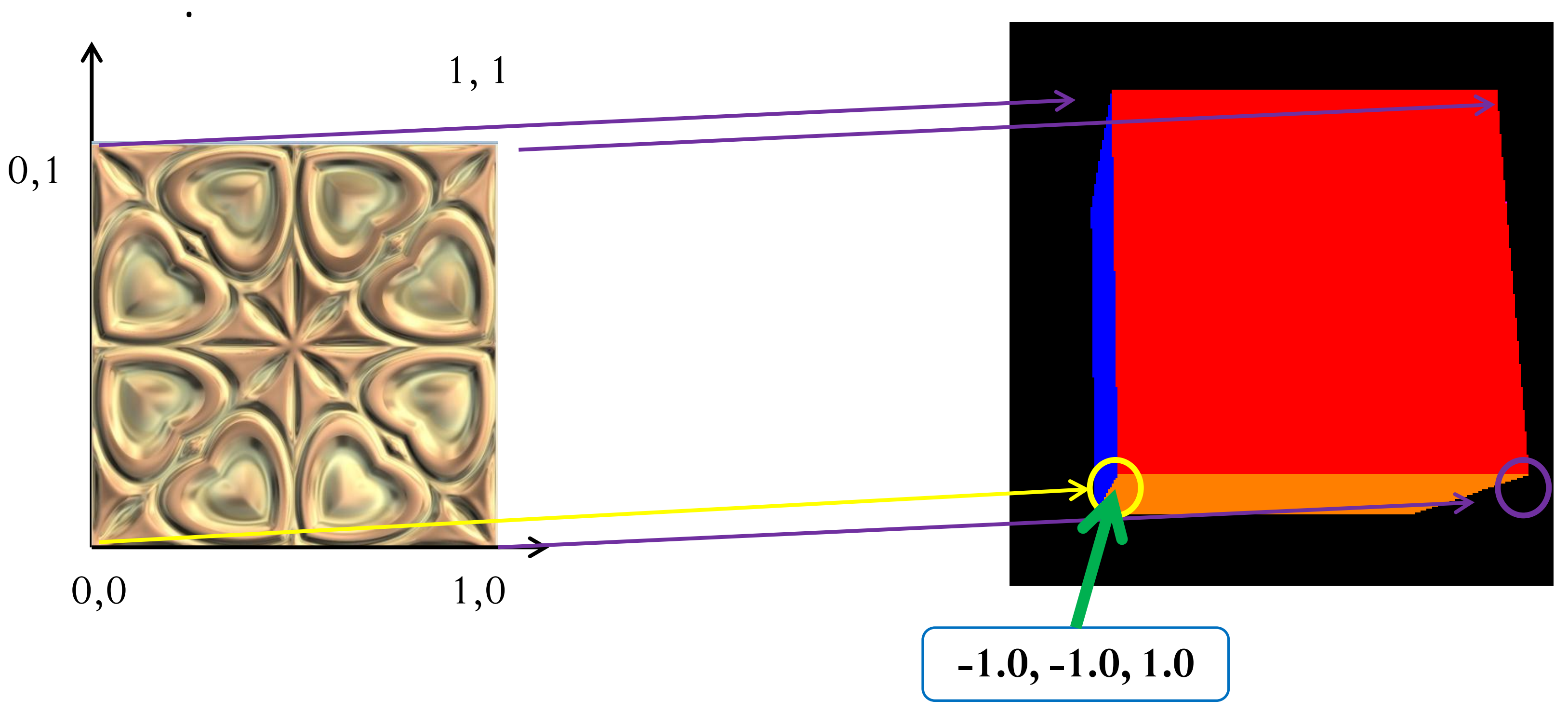
```
GLuint textureID;

void init(void)
{
    ......

    glGenTextures(1, &textureID);

    // "Bind" the newly created texture
    glBindTexture(GL_TEXTURE_2D, textureID);

    // Give the image to OpenGL
    glTexImage2D(GL_TEXTURE_2D, 0,GL_RGB, width, height, 0, GL_BGR, GL_UNSIGNED_BYTE, data);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

}
```

How we want OpenGL handle the texture when the object is bigger or smaller than the texture

Texture Mapping

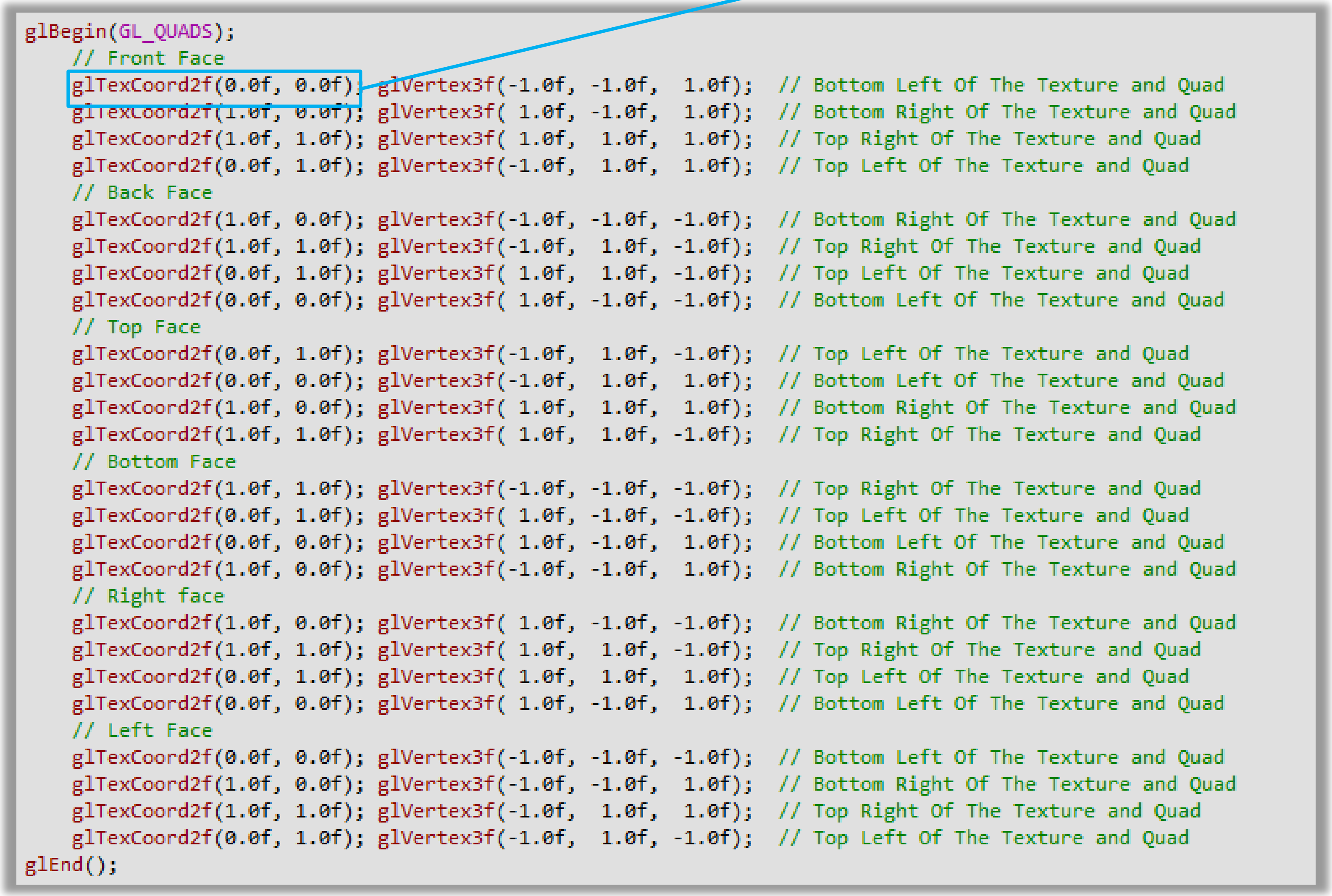


Texture Mapping



Texture Mapping

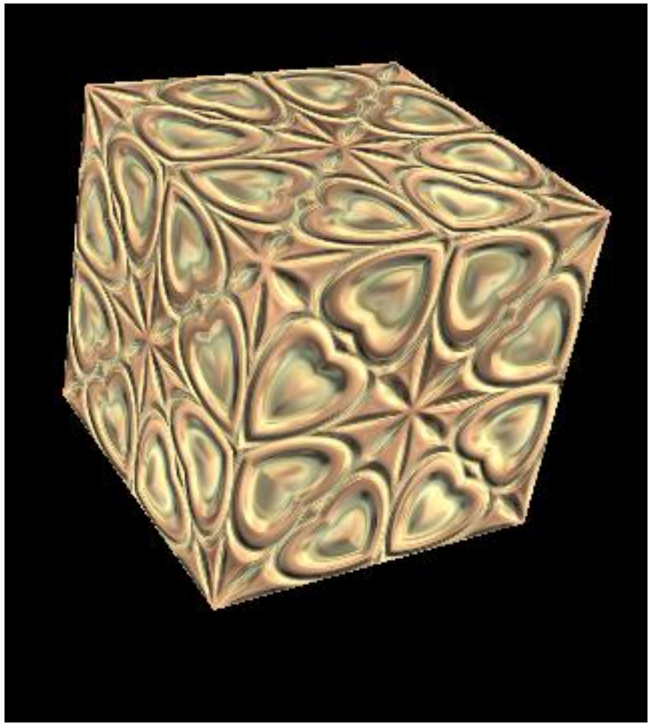
The coordinate of the texture to be mapped with this vertex



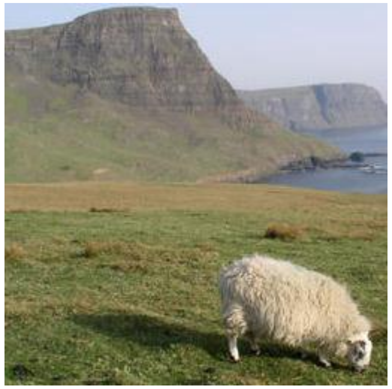
```
glBegin(GL_QUADS);
    // Front Face
    glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, -1.0f,  1.0f);  // Bottom Left Of The Texture and Quad
    glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f, -1.0f,  1.0f);  // Bottom Right Of The Texture and Quad
    glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f,  1.0f,  1.0f);  // Top Right Of The Texture and Quad
    glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f,  1.0f,  1.0f);  // Top Left Of The Texture and Quad
    // Back Face
    glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f, -1.0f);  // Bottom Right Of The Texture and Quad
    glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f,  1.0f, -1.0f);  // Top Right Of The Texture and Quad
    glTexCoord2f(0.0f, 1.0f); glVertex3f( 1.0f,  1.0f, -1.0f);  // Top Left Of The Texture and Quad
    glTexCoord2f(0.0f, 0.0f); glVertex3f( 1.0f, -1.0f, -1.0f);  // Bottom Left Of The Texture and Quad
    // Top Face
    glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f,  1.0f, -1.0f);  // Top Left Of The Texture and Quad
    glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f,  1.0f,  1.0f);  // Bottom Left Of The Texture and Quad
    glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f,  1.0f,  1.0f);  // Bottom Right Of The Texture and Quad
    glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f,  1.0f, -1.0f);  // Top Right Of The Texture and Quad
    // Bottom Face
    glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f, -1.0f, -1.0f);  // Top Right Of The Texture and Quad
    glTexCoord2f(0.0f, 1.0f); glVertex3f( 1.0f, -1.0f, -1.0f);  // Top Left Of The Texture and Quad
    glTexCoord2f(0.0f, 0.0f); glVertex3f( 1.0f, -1.0f,  1.0f);  // Bottom Left Of The Texture and Quad
    glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f,  1.0f);  // Bottom Right Of The Texture and Quad
    // Right face
    glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f, -1.0f, -1.0f);  // Bottom Right Of The Texture and Quad
    glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f,  1.0f, -1.0f);  // Top Right Of The Texture and Quad
    glTexCoord2f(0.0f, 1.0f); glVertex3f( 1.0f,  1.0f,  1.0f);  // Top Left Of The Texture and Quad
    glTexCoord2f(0.0f, 0.0f); glVertex3f( 1.0f, -1.0f,  1.0f);  // Bottom Left Of The Texture and Quad
    // Left Face
    glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, -1.0f, -1.0f);  // Bottom Left Of The Texture and Quad
    glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f,  1.0f);  // Bottom Right Of The Texture and Quad
    glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f,  1.0f,  1.0f);  // Top Right Of The Texture and Quad
    glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f,  1.0f, -1.0f);  // Top Left Of The Texture and Quad
glEnd();
```

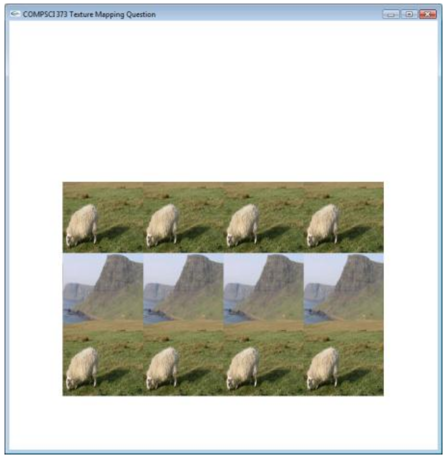
Texture Mapping

•



Texture



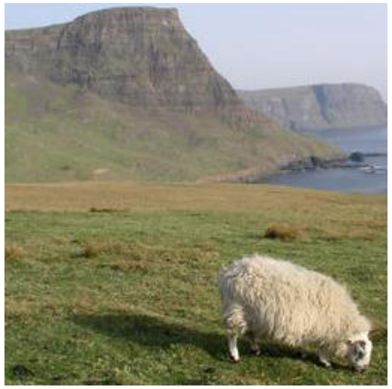


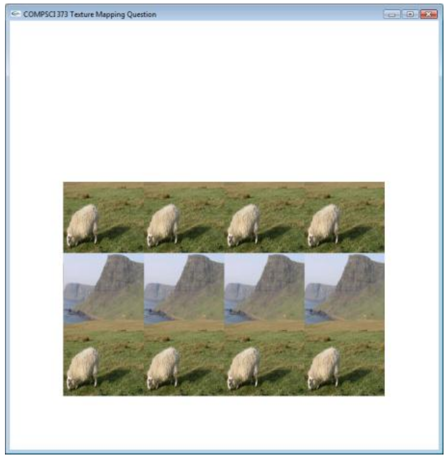
The image below on the left is used to texture map a rectangle as illustrated in the image below on the right. Assume the **bottom-right** vertex of the rectangle has the texture coordinates (0, 0).

What are the texture coordinates of the other three vertices?

(a) Bottom-left vertex: (1, 0); Top-left vertex: (1, 1.5); Top-right vertex: (0, 1.5)
(b) Bottom-left vertex: (0, 4); Top-left vertex: (1.5, 4); Top-right vertex: (1.5, 0)
(c) Bottom-left vertex: (0, 1); Top-left vertex: (1.5, 1); Top-right vertex: (1.5, 0)
(d) Bottom-left vertex: (1, 0); Top-left vertex: (1, 1); Top-right vertex: (0, 1)
(e) None of the others

Texture





The image below on the left is used to texture map a rectangle as illustrated in the image below on the right. Assume the **bottom-right** vertex of the rectangle has the texture coordinates (0, 0).

What are the texture coordinates of the other three vertices?

(a) Bottom-left vertex: (1, 0); Top-left vertex: (1, 1.5); Top-right vertex: (0, 1.5)
(b) Bottom-left vertex: (0, 4); Top-left vertex: (1.5, 4); Top-right vertex: (1.5, 0)
(c) Bottom-left vertex: (0, 1); Top-left vertex: (1.5, 1); Top-right vertex: (1.5, 0)
(d) Bottom-left vertex: (1, 0); Top-left vertex: (1, 1); Top-right vertex: (0, 1)

None of the others - The correct answer should be Bottom-left (4.0), bottom-right (0,0), top-left (4, 1.5), top-right (0, 1.5)