3D Graphics Programming Tools

OpenGL 2D Drawing

Dr. Xianhui Cherry Che

x.che@qmul.ac.uk

Learning Objectives

- Understand the purpose and the basic use of OpenGL
- Learn to use OpenGL to draw basic shapes
- Learn common 2D drawing with OpenGL

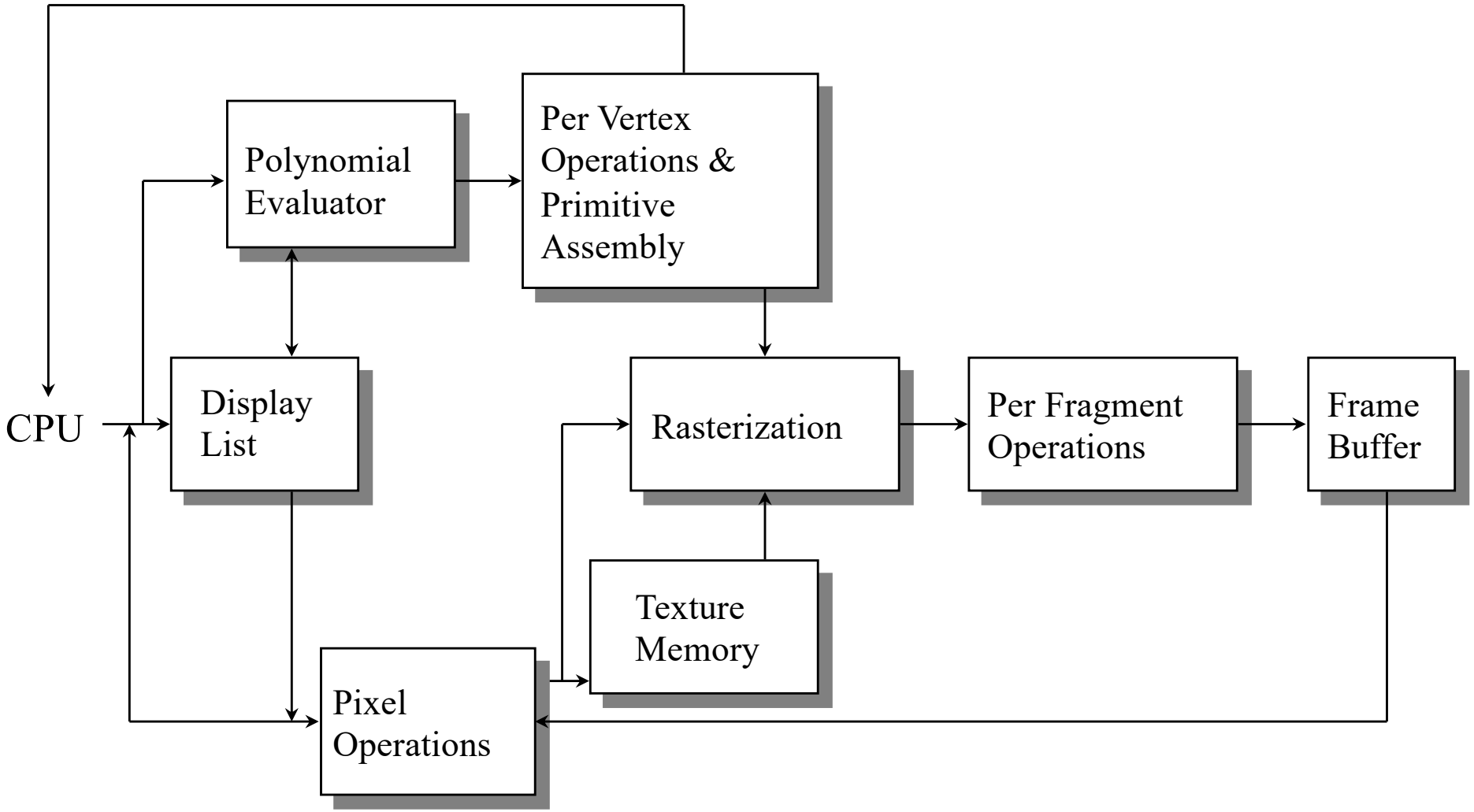
Topics

- Introduction to OpenGL
- Your First OpenGL Program
- Vertices and Coordinates
- OpenGL Primitives

What Is OpenGL?

- OpenGL standards for Open Graphics Library.
- Graphics rendering API
  - high-quality 2D and 3D graphics composed of geometric and image primitives
  - window system independent
  - operating system independent

OpenGL Architecture



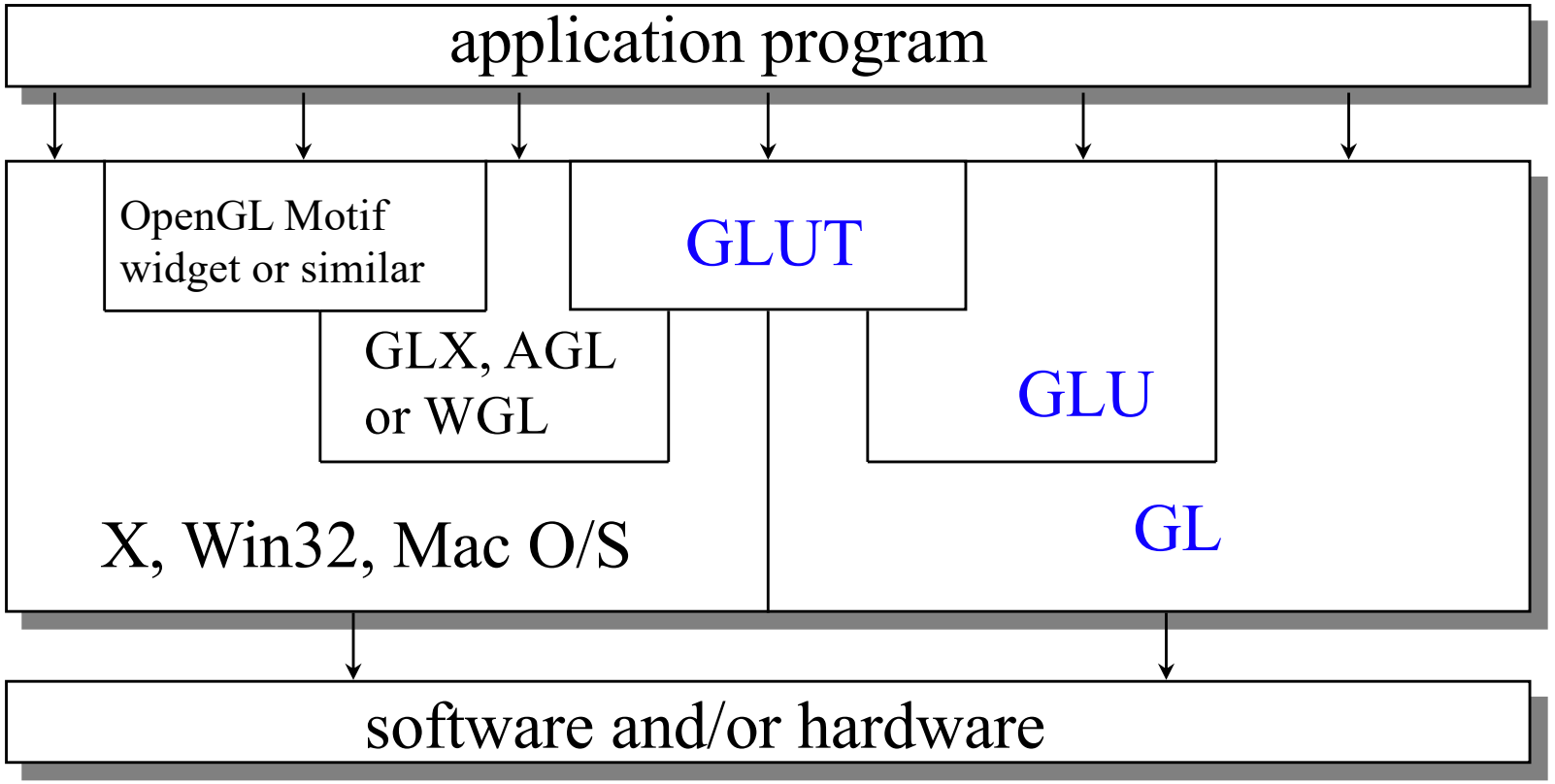
OpenGL as a Renderer

- Geometric primitives
  - points, lines and polygons
- Image Primitives
  - images and bitmaps
  - separate pipeline for images and geometry
    - linked through texture mapping
- Rendering depends on state
  - colours, materials, light sources, etc.

Related APIs

- AGL, GLX, WGL
  - glue between OpenGL and windowing systems
- GLU (OpenGL Utility Library)
  - **part of OpenGL**
  - NURBS, tessellators, quadric shapes, etc.
- GLUT (OpenGL Utility Toolkit)
  - portable windowing API
  - **not officially part of OpenGL**

OpenGL and Related APIs



Preliminaries

- Headers Files
  - #include <GL/gl.h>
  - #include <GL/glu.h>
  - **#include <GL/glut.h>**
  - **#include <GLUT/glut.h> (In Xcode)**
- Libraries
- Enumerated Types
  - OpenGL defines numerous data types for compatibility
    - **GLfloat, GLint, GLenum**, etc.
    - For more info on OpenGL data types, read: https://www.khronos.org/opengl/wiki/OpenGL_Type

Difference between GL, GLU, and GLUT

- GL is a core library of OpenGL, which contains most basic 2D/3D functions
- GLU is a utility library of OpenGL, which seems to assist GL.
- GLUT is the basic window interface which is cross-platform. It is independent from GL and GLU.

Official Documentations:

- **GL and GLU:** https://www.khronos.org/registry/OpenGL-Refpages/gl2.1/
- **GLUT:** *Background Reading – GLUT Documentation or* https://www.opengl.org/resources/libraries/glut/spec3/spec3.html

GLUT Basics

- Application Structure
  - Configure and open window
  - Initialize OpenGL state
  - Register input callback functions
    - render
    - resize
    - input: keyboard, mouse, etc.
  - Enter event processing loop

Topics

- Introduction to OpenGL
- Your First OpenGL Program
- Vertices and Coordinates
- OpenGL Primitives

Your First OpenGL Program

GL functions

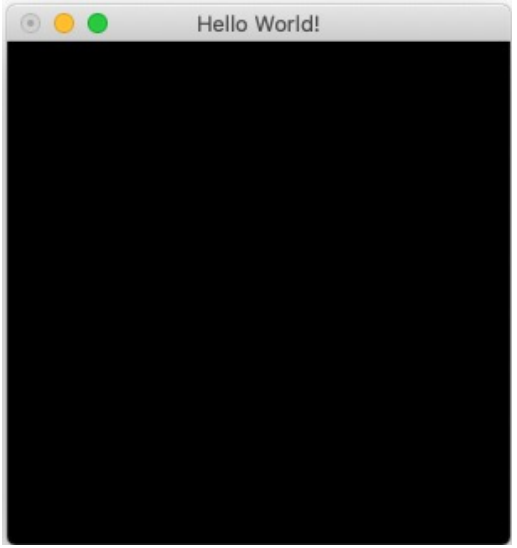
GLUT functions

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

void display() {
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glFlush();
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutCreateWindow("Hello World!");
    glutDisplayFunc(display);
    glutMainLoop();
}
```

Output:



Your First OpenGL Program

To make it even better

```
#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif

void display() {
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glFlush();
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutCreateWindow("Hello World!");
    glutDisplayFunc(display);
    glutMainLoop();
}
```

GLUT Functions

- `glutInit` allows the application to get command line arguments and initializes system
- `glutCreateWindow` creates a window with a title
- `glutDisplayFunc` declares the display callback
- `glutMainLoop` enters an infinite event loop

For more about GLUT Functions, read the official document:

Background Reading – GLUT Documentation

GL Functions

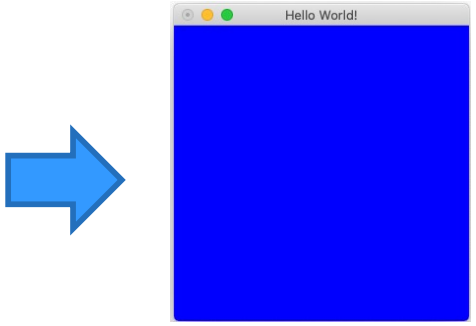
- `glClearColor` specifies clear values for the colour buffers.
- `glClear` clears buffers to preset values
- `glFlush` force execution of GL commands in finite time

For more about GL and GLU Functions, read the official reference:
https://www.khronos.org/registry/OpenGL-Refpages/gl2.1/

glClearColor

- `glClearColor` specifies clear values for the colour buffers.
- It takes four parameters: *red, green, blue, alpha*
- Specify the red, green, blue, and alpha values used when the colour buffers are cleared. The initial values are all 0.
- Values specified by this function are clamped to the range 0 – 1 .

```
glClearColor(0.0, 0.0, 1.0, 0.0);
```



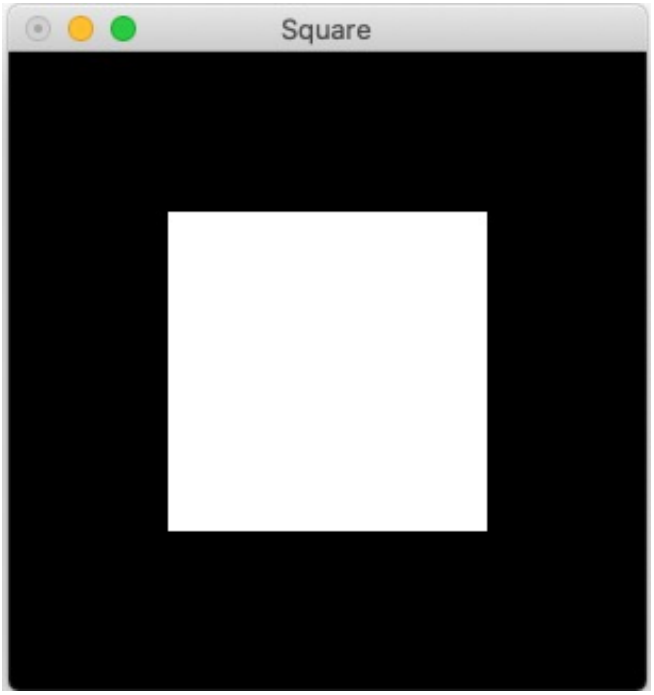
Topics

- Introduction to OpenGL
- Your First OpenGL Program
- Vertices and Coordinates
- OpenGL Primitives

Your Second Program – Draw a Square

```
#include <GL/glut.h>
void display(){
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_POLYGON);
    glVertex2f(-0.5, -0.5);
    glVertex2f(-0.5, 0.5);
    glVertex2f(0.5, 0.5);
    glVertex2f(0.5, -0.5);
    glEnd();
    glFlush();
}
int main(int argc, char** argv){
    glutInit(&argc, argv);
    glutCreateWindow("Square");
    glutDisplayFunc(display);
    glutMainLoop();
}
```

Output:



Draw a Square

Decide what to draw ←

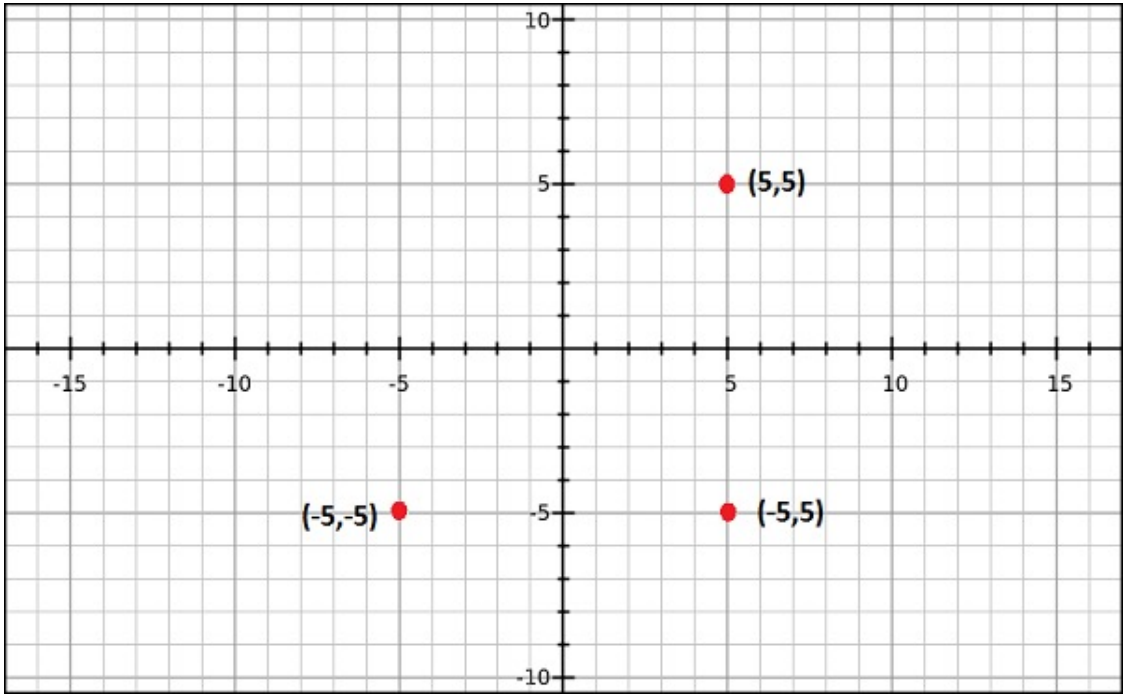
Specify vertices

End drawing ←

```
glBegin(GL_POLYGON);
glVertex2f(-0.5, -0.5);
glVertex2f(-0.5, 0.5);
glVertex2f(0.5, 0.5);
glVertex2f(0.5, -0.5);
glEnd();
```

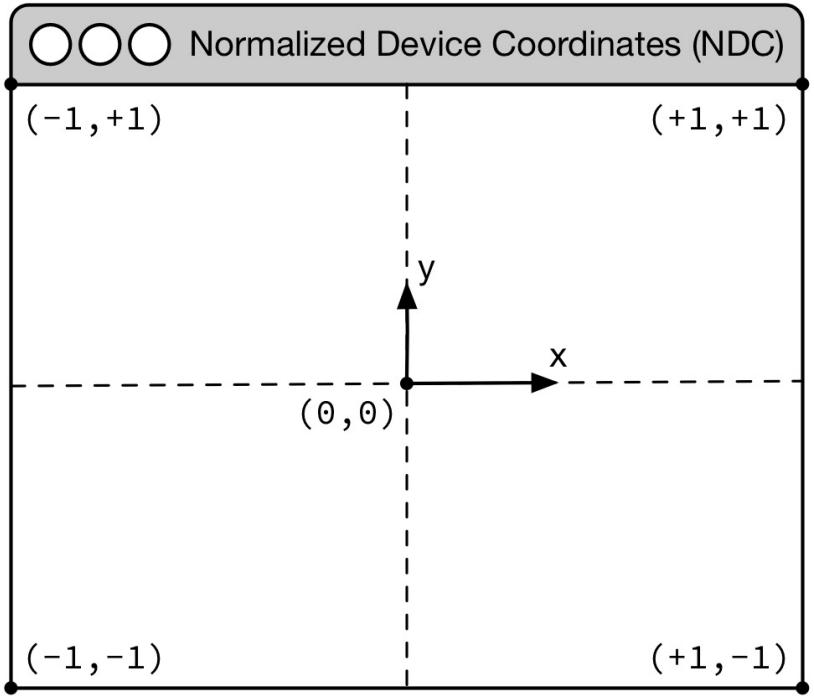
Vertices

- A **vertex** is a point which defines the conjunction of the edges of an object.
- In 2D context, it is represented by two values each representing x and y axes respectively.



Normalized Device Coordinates

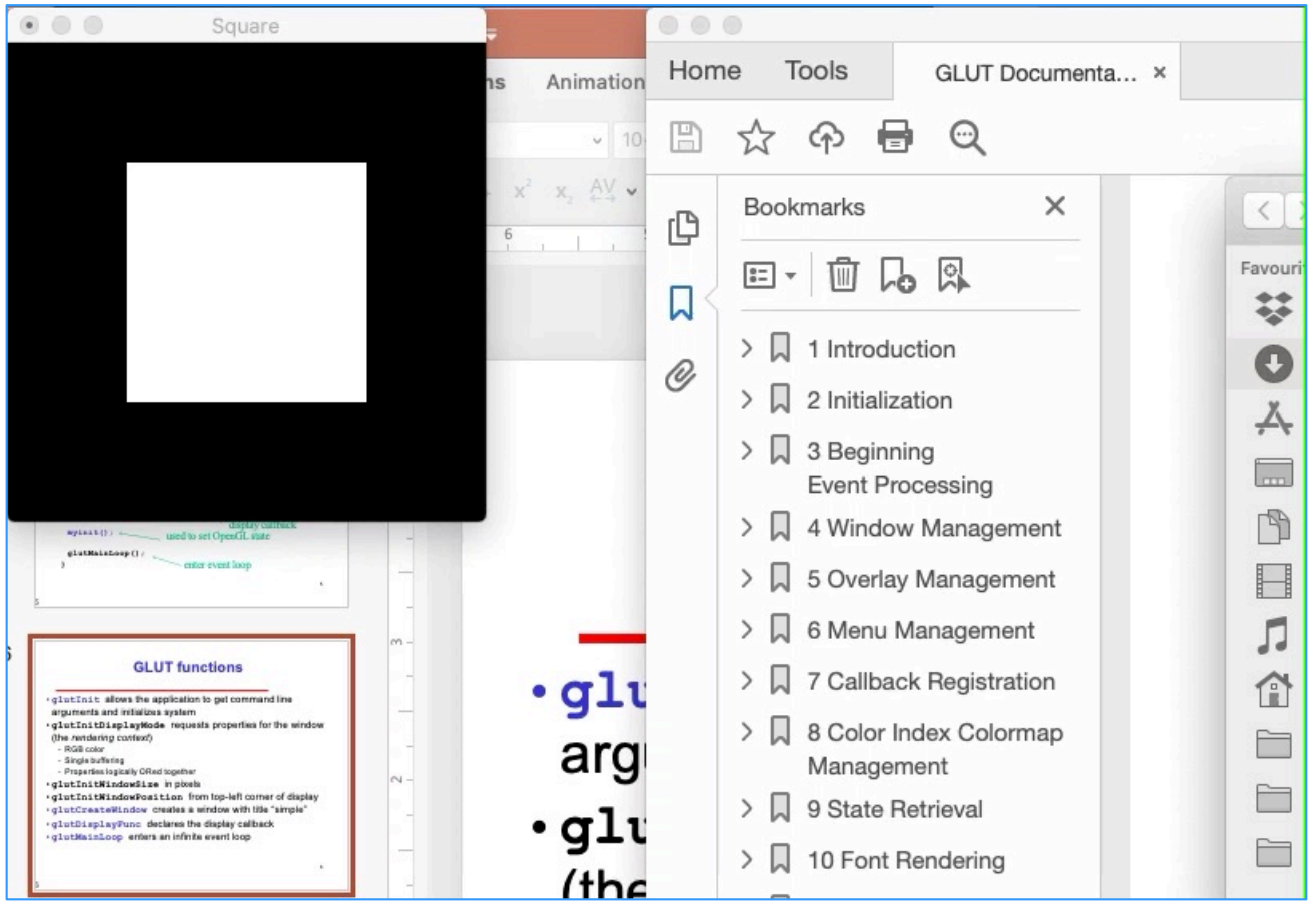
- Normalized Device Coordinates (NDC) is a display coordinate system that is **screen-independent**; it encompasses a 2D or 3D positions where the x, y, and/or z components range from -1 to 1.



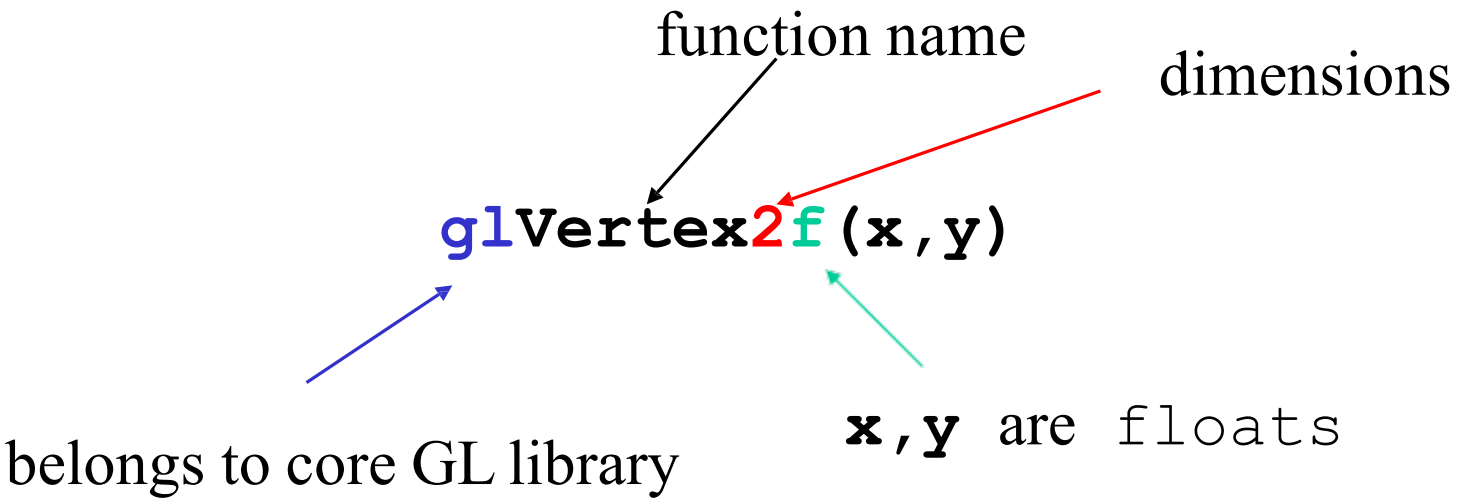
Window Size over NDC

- The polygon is stretched and positioned proportionally.

Demo



glVertex




```
glVertex2fv(p)
```

p is a pointer to an array

glVertex

```
glBegin(GL_POLYGON);
    glVertex2f(-0.5, -0.5);
    glVertex2f(-0.5, 0.5);
    glVertex2f(0.5, 0.5);
    glVertex2f(0.5, -0.5);
glEnd();
```

How to replace **glVertex2f** with **glVertex2fv**?

glVertex

```
GLfloat v[4][2] = {{-0.5, -0.5},
                   {-0.5,  0.5},
                   { 0.5,  0.5},
                    { 0.5, -0.5}};

    glBegin(GL_POLYGON);
        glVertex2fv(v[0]);
        glVertex2fv(v[1]);
        glVertex2fv(v[2]);
        glVertex2fv(v[3]);
    glEnd();
```

glVertex

Variations of this function in 2D include:

```
void glVertex2s( GLshort x,
                 GLshort y);

void glVertex2i( GLint x,
                 GLint y);

void glVertex2f( GLfloat x,
                 GLfloat y);

void glVertex2d( GLdouble x,
                 GLdouble y);
```

Draw a Square – Improved Structure

Function of drawing the square

Initialisation

The main program

```
#include <GLUT/glut.h>
void drawSquare(){
    glBegin(GL_POLYGON);
    glVertex2f(-0.5, -0.5);
    glVertex2f(-0.5, 0.5);
    glVertex2f(0.5, 0.5);
    glVertex2f(0.5, -0.5);
    glEnd();
    glFlush();
}
void init () {
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 1.0, 1.0);
    }
int main(int argc, char** argv){
    glutInit(&argc, argv);
    glutCreateWindow("Square");
    glutDisplayFunc(drawSquare);
    init();
    glutMainLoop();
}
```

Configure the Window

```
int main(int argc, char** argv){
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(300,300);
    glutInitWindowPosition(-1,-1);
    glutCreateWindow("Square");
    glutDisplayFunc(drawSquare);
    init();
    glutMainLoop();
}
```

GLUT Functions

- `glutInitDisplayMode` requests properties for the window (the *rendering context*)
  - -RGB colour
  - -Single buffering
  - -Properties logically ORed together (bitwise)
- `glutInitWindowSize` in pixels
- `glutInitWindowPosition` from top-left corner of display

Initial Window Size and Position

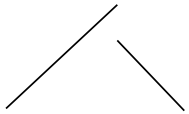
- The initial value of the initial window position GLUT state is -1 and -1. If either the X or Y component to the initial window position is negative, the actual window position is left to the window system to determine.
- The initial value of the initial window size GLUT state is 300 by 300.
- The intent of the initial window position and size values is to provide a suggestion to the window system for a window's initial size and position. The window system is not obligated to use this information. Therefore, GLUT programs should not assume the window was created at the specified size or position. A GLUT program should use the window's reshape callback to determine the true size of the window.

Topics

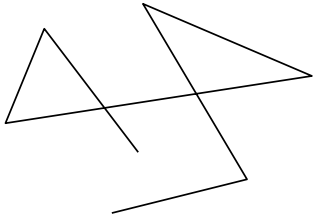
- Introduction to OpenGL
- Your First OpenGL Program
- Vertices and Coordinates
- OpenGL Primitives

OpenGL Primitives

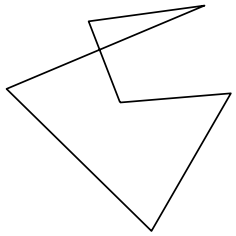
GL_POINTS



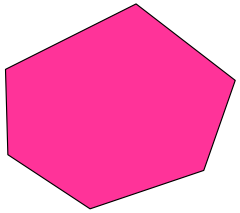
GL_LINES



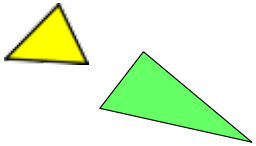
GL_LINE_STRIP



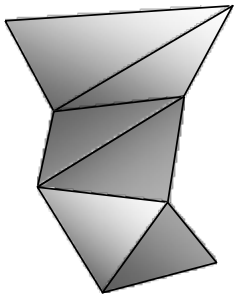
GL_LINE_LOOP



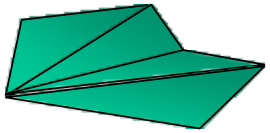
GL_POLYGON



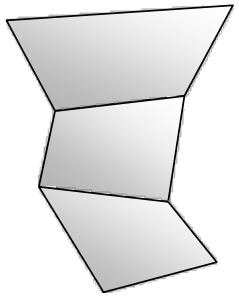
GL_TRIANGLES



GL_TRIANGLE_STRIP



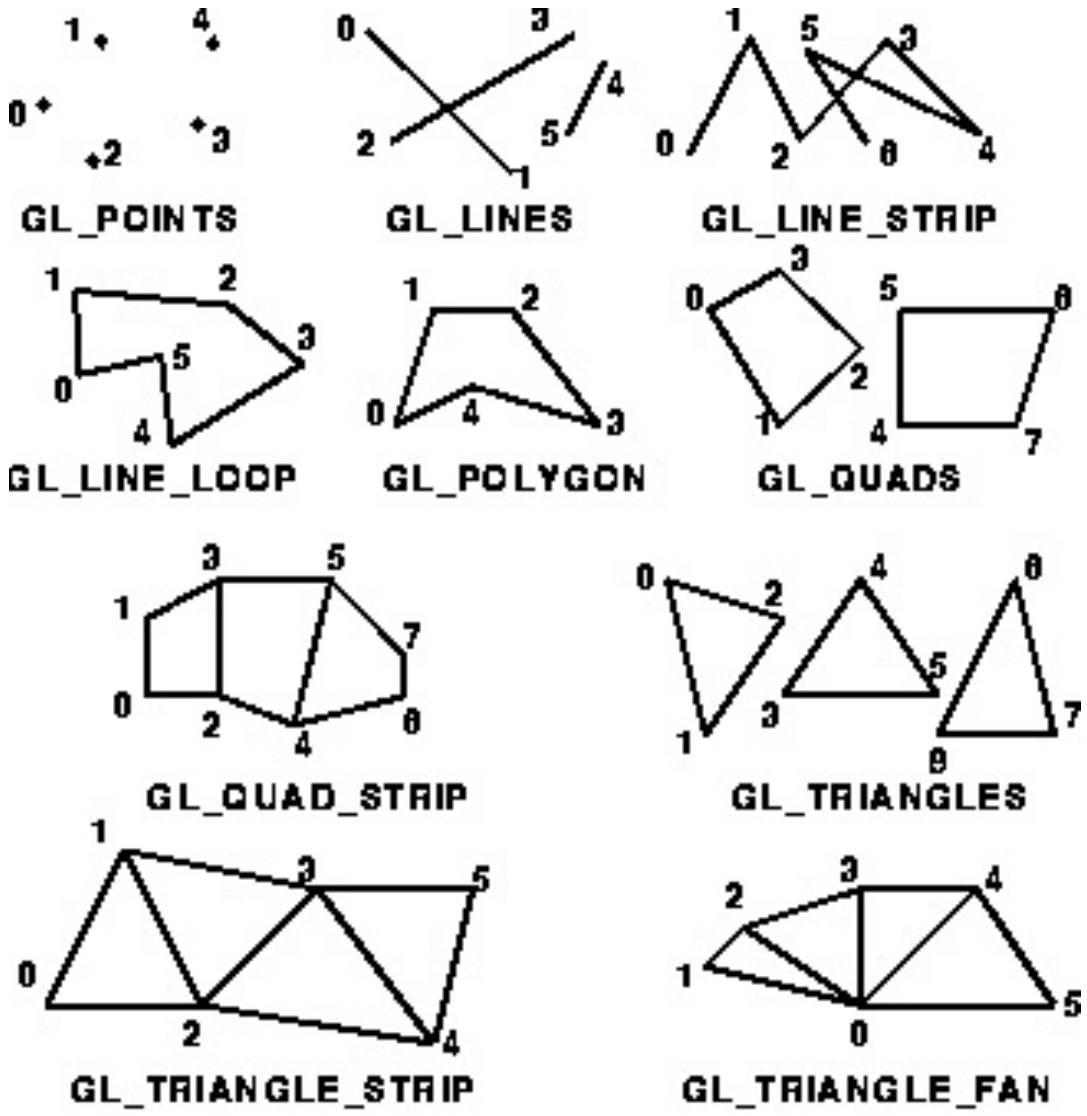
GL_TRIANGLE_FAN



GL_QUAD_STRIP

Vertices in OpenGL Primitives

- The order of executing (or drawing):

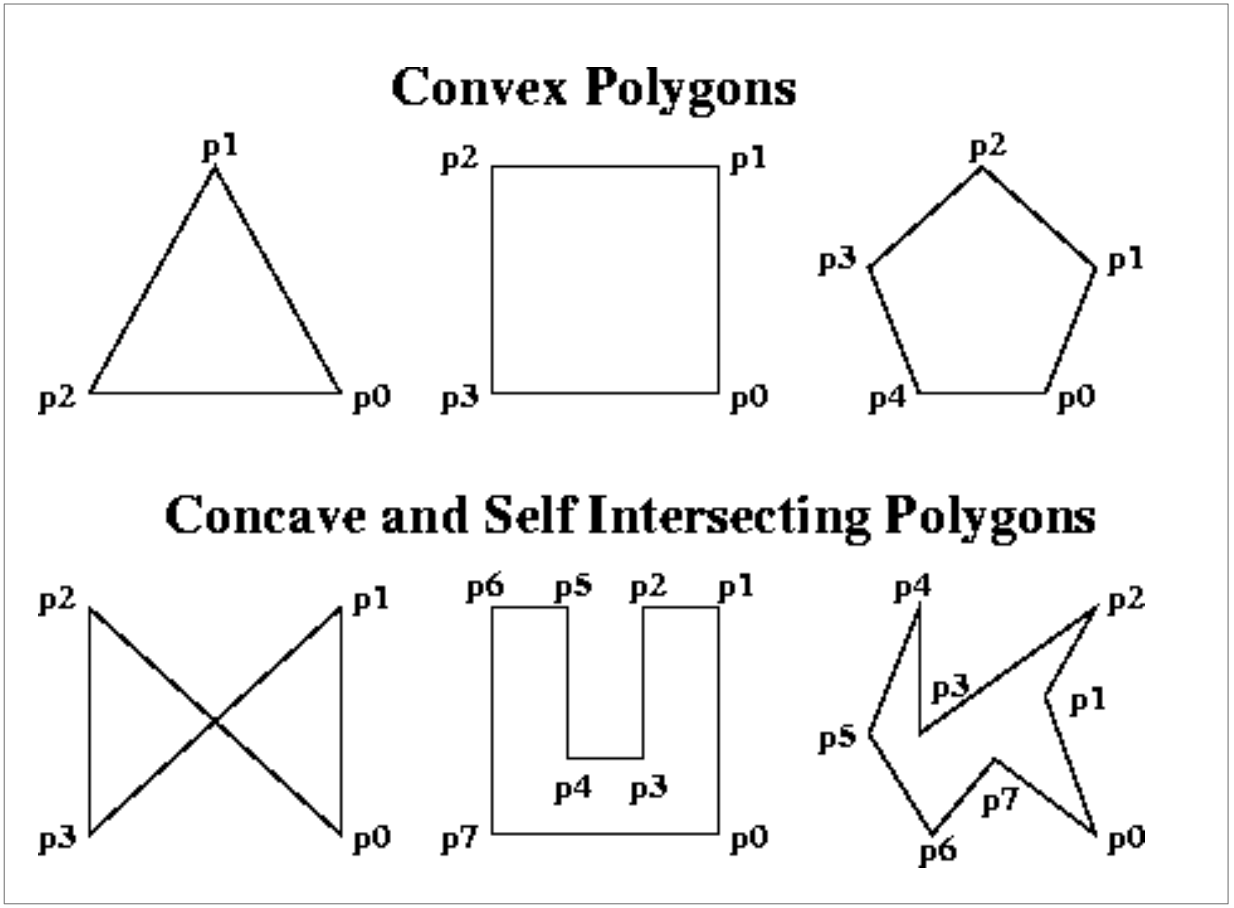


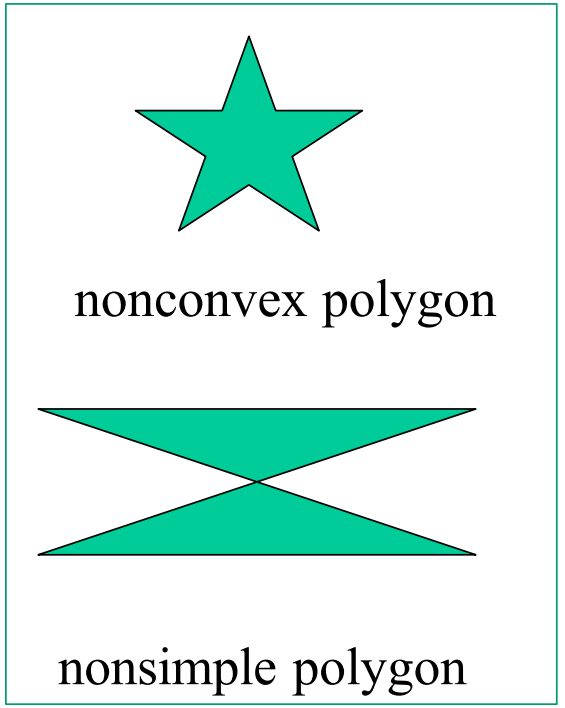
Polygon Issues

- OpenGL will only display polygons correctly that are
  - Simple: edges cannot cross
  - Convex: All points on a line segment between two points in a polygon are also in the polygon
  - Flat: all vertices are in the same plane
- User program can check if above true
  - OpenGL will produce output if these conditions are violated but it may not be what is desired
- Triangles satisfy all conditions

25

Polygon Issues





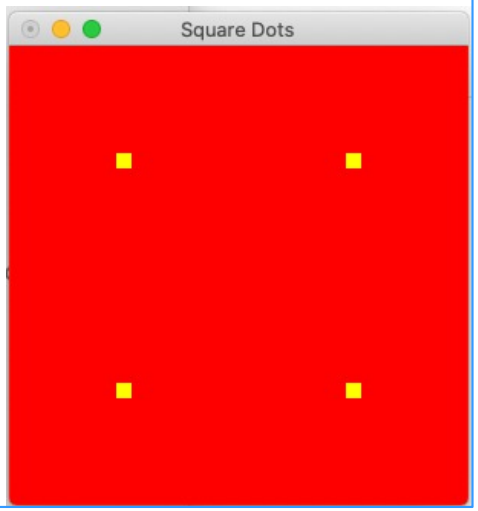
26

GL_POINTS

- Example:

```
#include <GLUT/glut.h>
void display(){
    GLfloat v[4][2] = {{-0.5, -0.5},{-0.5, 0.5},{ 0.5, 0.5}, {0.5, -0.5}};
    glClearColor(1.0, 0.0, 0.0, 0.0);
    glColor3f (1.0, 1.0, 0.0);
    glPointSize(10);
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_POINTS);
        glVertex2fv(v[0]);
        glVertex2fv(v[1]);
        glVertex2fv(v[2]);
        glVertex2fv(v[3]);
    glEnd();
    glFlush();
}
int main(int argc, char** argv){
    glutInit(&argc, argv);
    glutCreateWindow("Square Dots");
    glutDisplayFunc(display);
    glutMainLoop();
}
```

Output:



GL Functions

- `glColor3f` set the current colour with 3 floats (normalised).
- `glPointSize` specifies the diameter of rasterised points

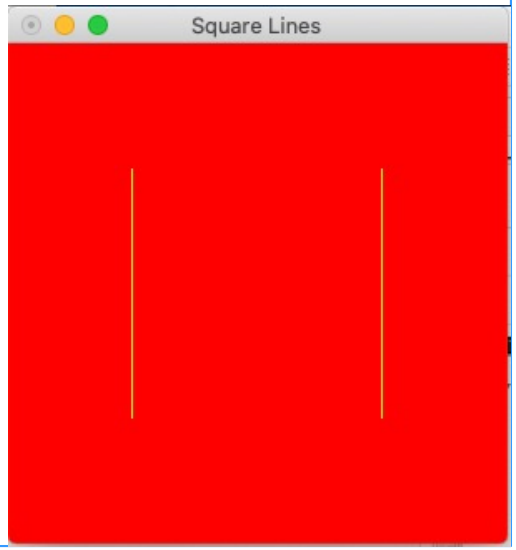
For more about GL and GLU Functions, read the official reference:
https://www.khronos.org/registry/OpenGL-Refpages/gl2.1/

GL_LINES

- Example:

```
#include <GLUT/glut.h>
void display(){
    GLfloat v[4][2] = {{-0.5, -0.5},{-0.5, 0.5},{ 0.5, 0.5}, {0.5, -0.5}};
    glClearColor(1.0, 0.0, 0.0, 0.0);
    glColor3f (1.0, 1.0, 0.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_LINES);
        glVertex2fv(v[0]);
        glVertex2fv(v[1]);
        glVertex2fv(v[2]);
        glVertex2fv(v[3]);
    glEnd();
    glFlush();
}
int main(int argc, char** argv){
    glutInit(&argc, argv);
    glutCreateWindow("Square Dots");
    glutDisplayFunc(display);
    glutMainLoop();
}
```

Output:

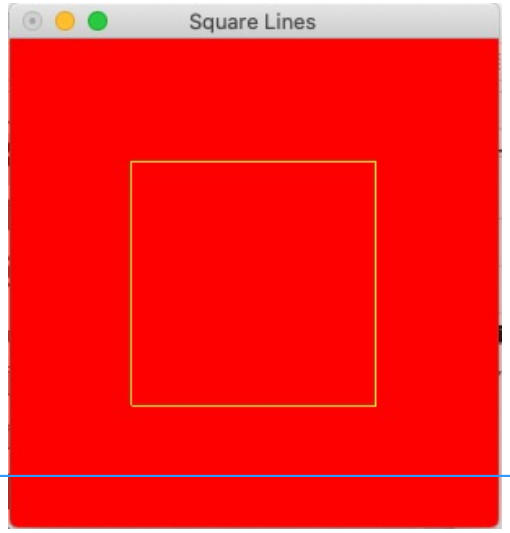


GL_LINES

- To rectify:

```
#include <GLUT/glut.h>
void display(){
    GLfloat v[8][2] = {{-0.5, -0.5},{-0.5, 0.5},{-0.5, 0.5},{ 0.5, 0.5},{ 0.5, 0.5},
{0.5, -0.5}, {0.5, -0.5}, {-0.5, -0.5}};
    glClearColor(1.0, 0.0, 0.0, 0.0);
    glColor3f (1.0, 1.0, 0.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_LINES);
        glVertex2fv(v[0]);
        glVertex2fv(v[1]);
        glVertex2fv(v[2]);
        glVertex2fv(v[3]);
        glVertex2fv(v[4]);
        glVertex2fv(v[5]);
        glVertex2fv(v[6]);
        glVertex2fv(v[7]);
    glEnd();
    glFlush();
}
```

Output:



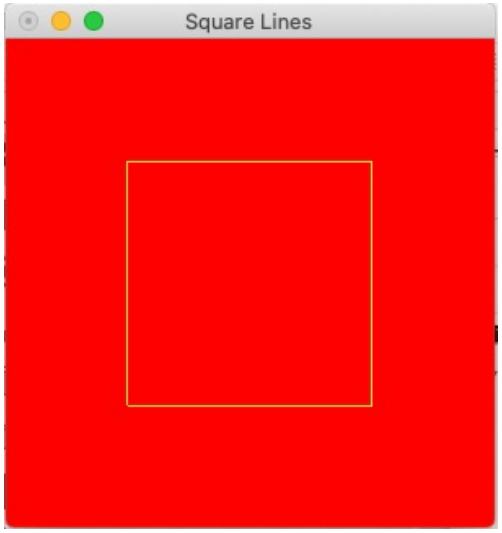
GL_LINE_STRIP

- A more efficient way:

Five vertices are needed as the first one shows up in the end again.

```
#include <GLUT/glut.h>
void display(){
    GLfloat v[5][2] = {{-0.5, -0.5},{-0.5, 0.5},{ 0.5, 0.5}, {0.5, -0.5}, {-0.5, -
0.5}};
    glClearColor(1.0, 0.0, 0.0, 0.0);
    glColor3f (1.0, 1.0, 0.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_LINE_STRIP);
        glVertex2fv(v[0]);
        glVertex2fv(v[1]);
        glVertex2fv(v[2]);
        glVertex2fv(v[3]);
        glVertex2fv(v[4]);
    glEnd();
    glFlush();
}
int main(int argc, char** argv){
    glutInit(&argc, argv);
    glutCreateWindow("Square Lines");
    glutDisplayFunc(display);
    glutMainLoop();
}
```

Output:



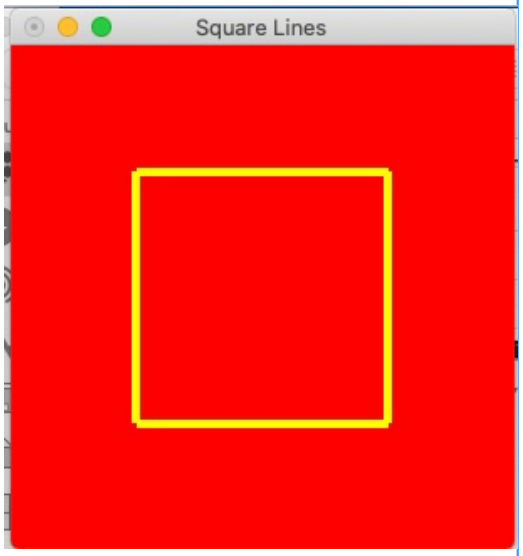
GL_LINE_LOOP

- An even better way!

Only the original four vertices are needed.

```
#include <GLUT/glut.h>
void display(){
    GLfloat v[4][2] = {{-0.5, -0.5},{-0.5, 0.5},{ 0.5, 0.5}, {0.5, -0.5}};
    glClearColor(1.0, 0.0, 0.0, 0.0);
    glColor3f (1.0, 1.0, 0.0);
    glLineWidth (5);
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_LINE_LOOP);
        glVertex2fv(v[0]);
        glVertex2fv(v[1]);
        glVertex2fv(v[2]);
        glVertex2fv(v[3]);
    glEnd();
    glFlush();
}
int main(int argc, char** argv){
    glutInit(&argc, argv);
    glutCreateWindow("Square Lines");
    glutDisplayFunc(display);
    glutMainLoop();
}
```

Output:



GL Functions

- `glLineWidth` specifies the width of rasterised lines

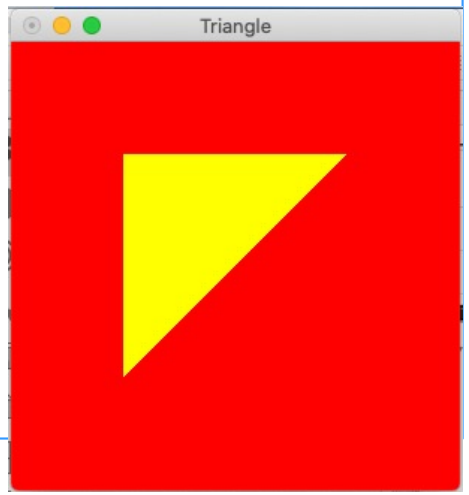
For more about GL and GLU Functions, read the official reference:
https://www.khronos.org/registry/OpenGL-Refpages/gl2.1/

GL_TRIANGLES

- Example:

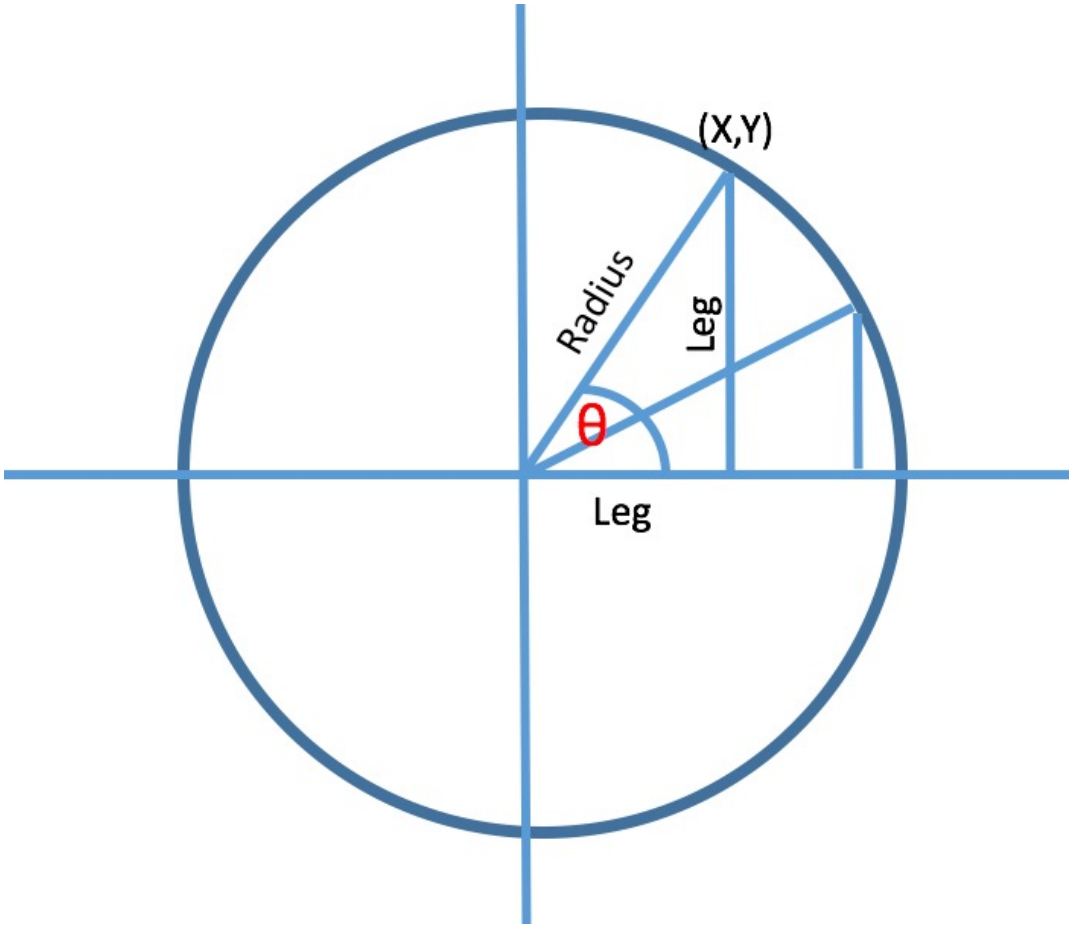
```
#include <GLUT/glut.h>
void display(){
    GLfloat v[3][2] = {{-0.5, -0.5},{-0.5, 0.5},{ 0.5, 0.5}};
    glClearColor(1.0, 0.0, 0.0, 0.0);
    glColor3f (1.0, 1.0, 0.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_TRIANGLES);
        glVertex2fv(v[0]);
        glVertex2fv(v[1]);
        glVertex2fv(v[2]);
    glEnd();
    glFlush();
}
int main(int argc, char** argv){
    glutInit(&argc, argv);
    glutCreateWindow("Square Lines");
    glutDisplayFunc(display);
    glutMainLoop();
}
```

Output:



Drawing a Circle

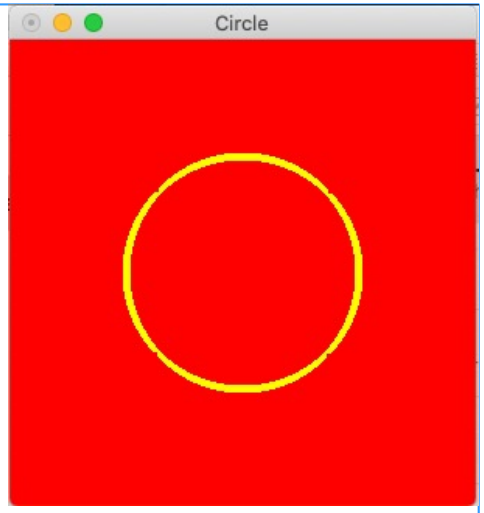
- Circle is not an existing OpenGL primitive, but you can use **`GL_LINE_LOOP`** or **`GL_TRIANGLE_FAN`** to create a smooth circle.



Drawing a Circle

- Using **GL_LINE_LOOP:**

Output:




```
#include <GLUT/glut.h>
#include <math.h>
void display(){
    float r = 0.5;
    int num_segments = 100;
    glClearColor(1.0, 0.0, 0.0, 0.0);
    glColor3f (1.0, 1.0, 0.0);
    glLineWidth (5);
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_LINE_LOOP);
    for(int i = 0; i < num_segments; i++) {
        float theta = 2.0f * 3.1415926f * float(i) / float(num_segments);
        float x = r * cosf(theta)
        float y = r * sinf(theta);
        glVertex2f(x, y);    }
    glEnd();
    glFlush(); }
int main(int argc, char** argv){
    glutInit(&argc, argv);
    glutCreateWindow("Circle");
    glutDisplayFunc(display);
    glutMainLoop();
}
```

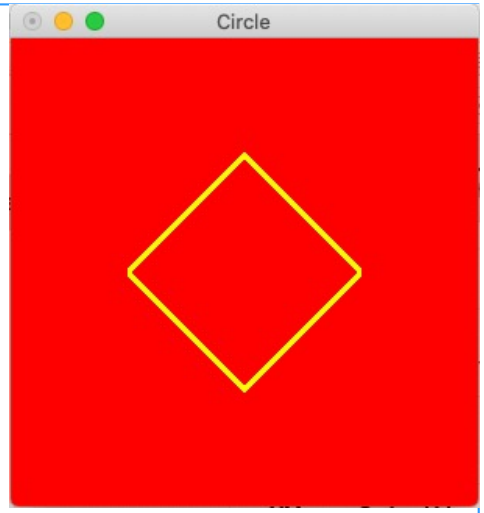
Calculate the current angle;
Retrieve x and y position;

Drawing a Circle

- Using **GL_LINE_LOOP:**

A circle with extremely low resolution☺

Output:

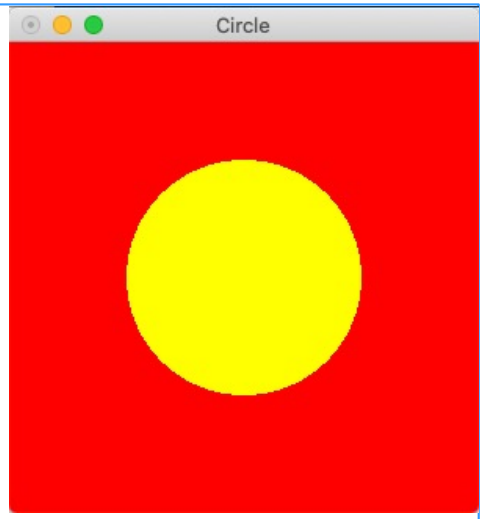



```
#include <GLUT/glut.h>
#include <math.h>
void display(){
    float r = 0.5;
    int num_segments = 4;
    glClearColor(1.0, 0.0, 0.0, 0.0);
    glColor3f (1.0, 1.0, 0.0);
    glLineWidth (5);
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_LINE_LOOP);
    for(int i = 0; i < num_segments; i++) {
        float theta = 2.0f * 3.1415926f * float(i) / float(num_segments);
        float x = r * cosf(theta)
        float y = r * sinf(theta);
        glVertex2f(x, y);    }
    glEnd();
    glFlush(); }
int main(int argc, char** argv){
    glutInit(&argc, argv);
    glutCreateWindow("Circle");
    glutDisplayFunc(display);
    glutMainLoop();
}
```

Drawing a Circle

- Using **GL_TRIANGLE_FAN:**

Output:




```
#include <GLUT/glut.h>
#include <math.h>
void display(){
    float r = 0.5;
    int num_segments = 100;
    glClearColor(1.0, 0.0, 0.0, 0.0);
    glColor3f (1.0, 1.0, 0.0);
    glLineWidth (5);
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_TRIANGLE_FAN);
    for(int i = 0; i < num_segments; i++) {
        float theta = 2.0f * 3.1415926f * float(i) / float(num_segments);
        float x = r * cosf(theta)
        float y = r * sinf(theta);
        glVertex2f(x, y);    }
    glEnd();
    glFlush(); }
int main(int argc, char** argv){
    glutInit(&argc, argv);
    glutCreateWindow("Circle");
    glutDisplayFunc(display);
    glutMainLoop();
}
```

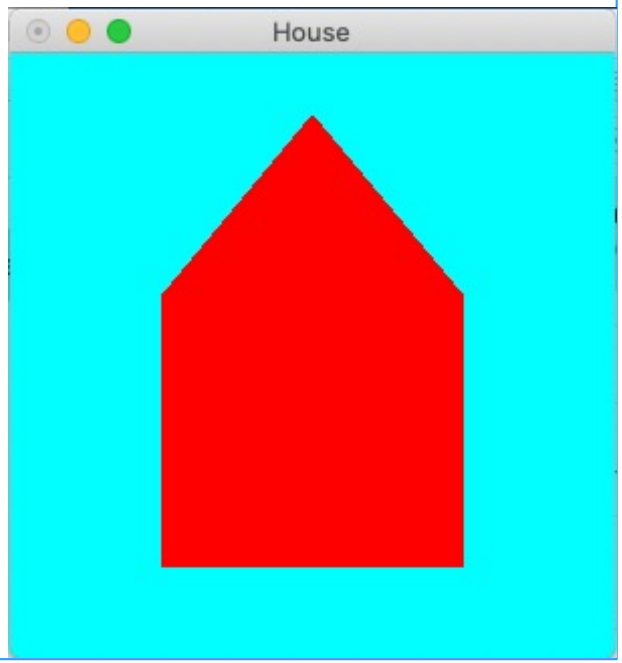
OpenGL State

- OpenGL operates as a state machine.
- It means that once the value of a property is set, the value persists until a new value is set.
- For example, if `glColor` command is used to set the current drawing color to black, black will be used to draw ALL objects until `glColor` command is used again to change the drawing color.
- Everything shall remain until you explicitly change it!

OpenGL State – Example

```
#include <GLUT/glut.h>
void display(){
    GLfloat t[3][2] = {{0.0,0.8}, {0.5,0.2}, {-0.5,0.2}};
    GLfloat s[4][2] = {{0.5,0.2}, {0.5,-0.7}, {-0.5,-0.7}, {-0.5,0.2}};
    glClearColor(0.0, 1.0, 1.0, 0.0);
    glColor3f (1.0, 0.0, 0.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_TRIANGLES);
        glVertex2fv(t[0]);
        glVertex2fv(t[1]);
        glVertex2fv(t[2]);
    glEnd();
    glBegin(GL_POLYGON);
            glVertex2fv(s[0]);
            glVertex2fv(s[1]);
            glVertex2fv(s[2]);
            glVertex2fv(s[3]);
    glEnd();
    glFlush();
}
int main(int argc, char** argv){
    glutInit(&argc, argv);
    glutCreateWindow("Square Lines");
    glutDisplayFunc(display);
    glutMainLoop();}
```

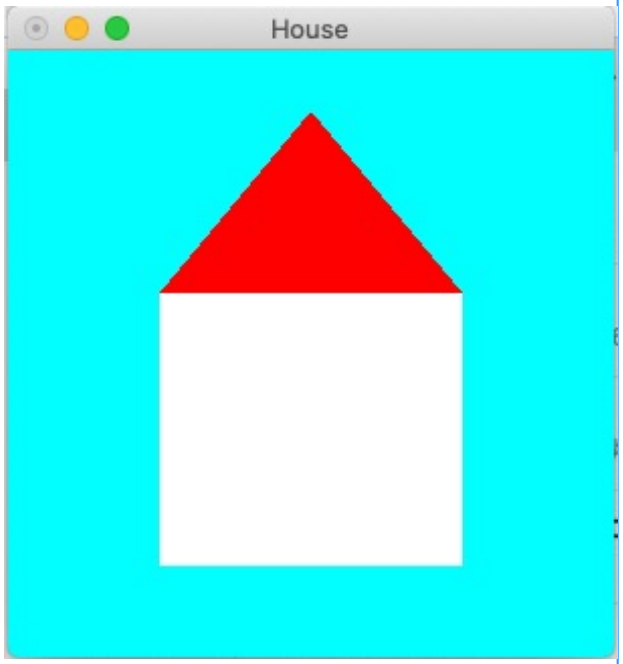
Output:



OpenGL State – Example

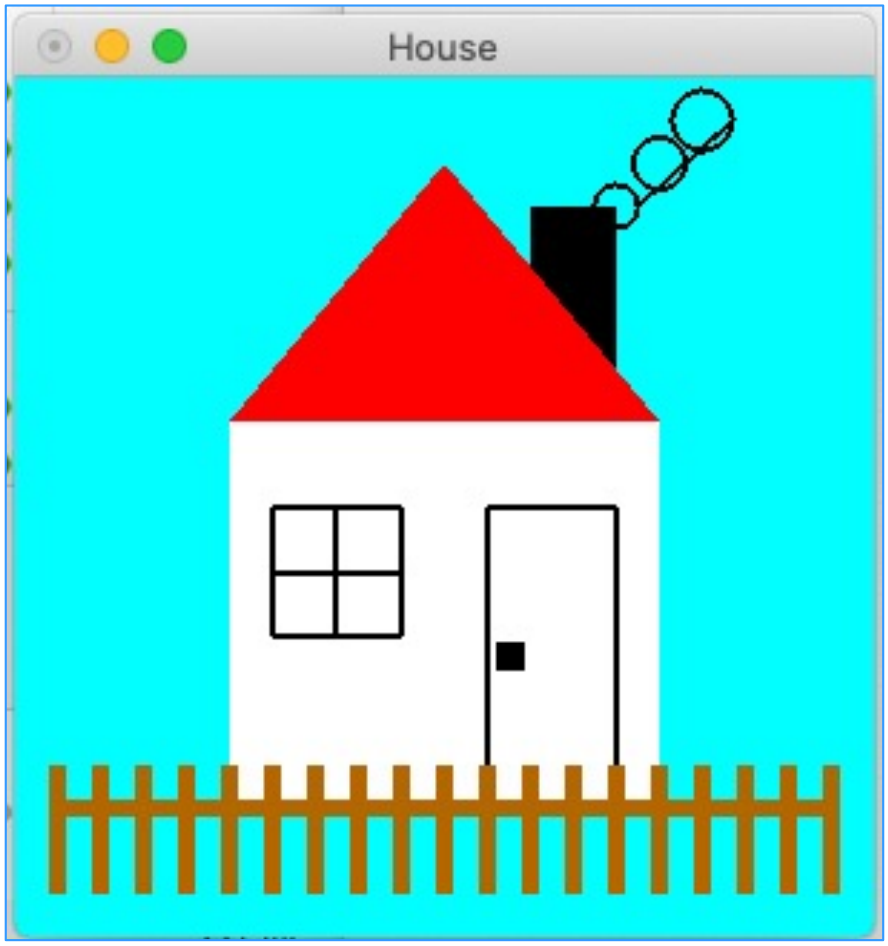
```
#include <GLUT/glut.h>
void display(){
    GLfloat t[3][2] = {{0.0,0.8}, {0.5,0.2}, {-0.5,0.2}};
    GLfloat s[4][2] = {{0.5,0.2}, {0.5,-0.7}, {-0.5,-0.7}, {-0.5,0.2}};
    glClearColor(0.0, 1.0, 1.0, 0.0);
    glColor3f (1.0, 0.0, 0.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_TRIANGLES);
        glVertex2fv(t[0]);
        glVertex2fv(t[1]);
        glVertex2fv(t[2]);
    glEnd();
    glColor3f (1.0, 1.0, 1.0);
    glBegin(GL_POLYGON);
            glVertex2fv(s[0]);
            glVertex2fv(s[1]);
            glVertex2fv(s[2]);
            glVertex2fv(s[3]);
    glEnd();
    glFlush();
}
int main(int argc, char** argv){
    glutInit(&argc, argv);
    glutCreateWindow("Square Lines");
    glutDisplayFunc(display);
    glutMainLoop();}
```

Output:



To Do in Lab 1

- Draw this picture:



Questions?

x.che@qmul.ac.uk