# Colours, Primitives, & Drawing

Game 300

# Colours

- In art we learn about the 3 primary colours:
  - Red, blue and yellow
- In computer graphics we use a different 3 primary colours:
  - Red, green, blue or RGB
  - This is because pixels are coloured on our LED screens by emitting light.
  - Light colours do not work the same way as paint does.
    - Paint is **subtractive** while light colours are **additive**.
    - We will discuss lighting and its effects with colour more in future lectures

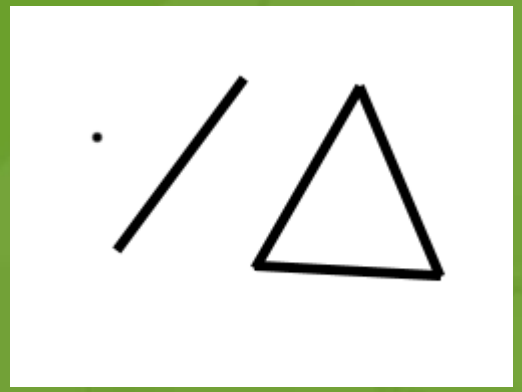
# Colours Cont & Transparency

- Typically colours range from 0-255 for each value.
  - Where 0 is no colour and 255 is full colour.
  - 255,255,255 = white
  - 0,0,0 = black
- Colours are often represented as a Vector3 or Vector4 of float values ranging from 0.0f-1.0f.
  - This is known as **normalized** values
- A fourth value can be added to our colour codes known as Alpha.
  - Alpha is the level of transparency applied

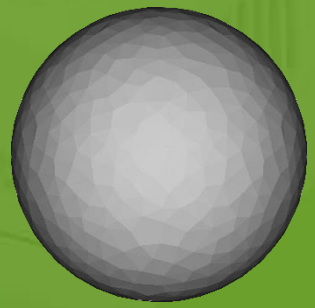
# Primitives



- Fundamental unit of rendering is known as a "Primitive“
  - Points
  - Lines
  - Triangles



- Everything in games rendered breaks down into one or more of the above.
  - Spherical objects are actually just a lot of triangles
    - Which is just lines
      - Which is just points…



  - the smoother the sphere or rounded object, the more triangle required.
    - other ways to fake this through textures explained later.

# Vectors

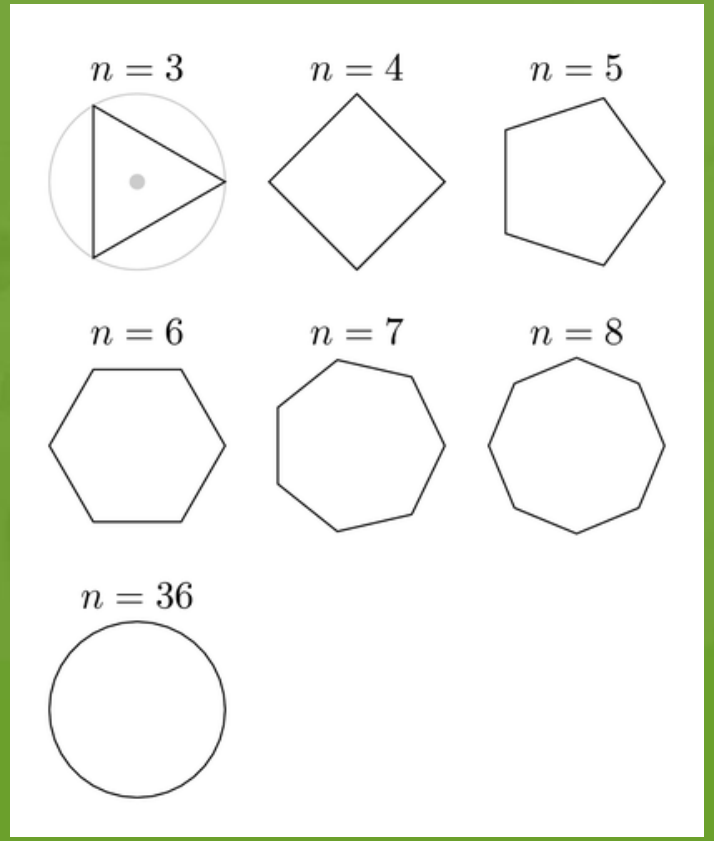
## 3D Terms

- Vectors are used constantly for 3D Calculations
  - Vectors are usually used to specify a position or a direction or length and magnitude.
- Vectors are just 2D arrays of typically 2-4 elements
  - A 2D vector contains an X and a Y value
  - A 3D vector contains an X, Y, and a Z value
  - A 4D vector contains an X, Y, Z, and a W value
- Typically based off floats in 3D programming.
  - Vectors can be of any type.
  - prebuilt in most math libraries
  - GLM is an OpenGL Math library which handles vectors, quaternions, and matrices.

# Polygons



- A series of points connected by lines
  - a triangle is a polygon, not all polys are tris
    - (all polys can be broken down into tris)
- N = num_points
- Min Tris = n-2




- http://www.texample.net/tikz/examples/regular-polygons/

# Convex vs Concave



- convex: (of a polygon)
  - having only interior angles measuring less than 180°.
  - picture displays properly
    - triangle is always convex.
- concave:
  - have an inward section or curve.
  - if a poly has a convex section it generally means it will be comprised of more than 1 triangle.

# Immediate Mode


- Immediate Mode was the first way that OpenGL used to draw shapes and primitives on screen.
  - In 3.0 of OpenGL they created a new way of rendering primitives which use buffers and take more advantage of memory.
- They have marked Immediate mode for removal since 3.0 however, many game engines are still based on immediate mode like rendering.

# Drawing – WHAT

- To Begin with we use a function called glBegin() to tell OpenGL we want to begin drawing something
- glBegin takes in a single parameter of an enum value.
  - This value dictates WHAT it will draw (the primitive).
  - Some example options you can pass glBegin are:

```
GL_POINTS
GL_LINES
GL_TRIANGLES
GL_TRIANGLE_STRIP
```

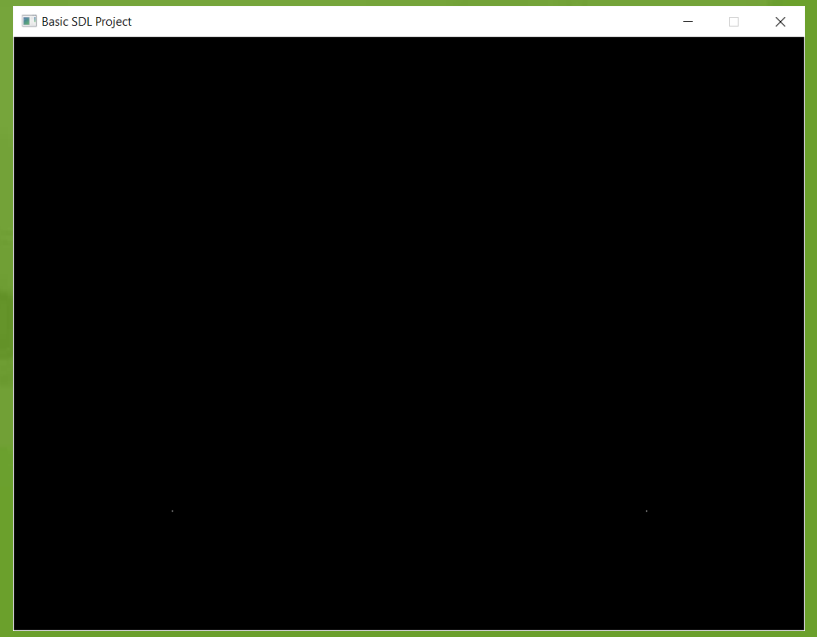
  - So our code might look something like this:

```
glBegin( GL_TRIANGLES );
```

# Drawing - WHERE

- So our program now knows WHAT we want to draw, however it doesn't know WHERE or HOW we want to draw it.
  - The way immediate mode( and graphics in general) works is it used a series of points to determine the locations of our shapes.
  - We can define the points to draw by using the glVertex3f function.

```
glBegin( GL_POINTS );
glVertex3f( -0.5f, -0.5f, 1.0f );
```

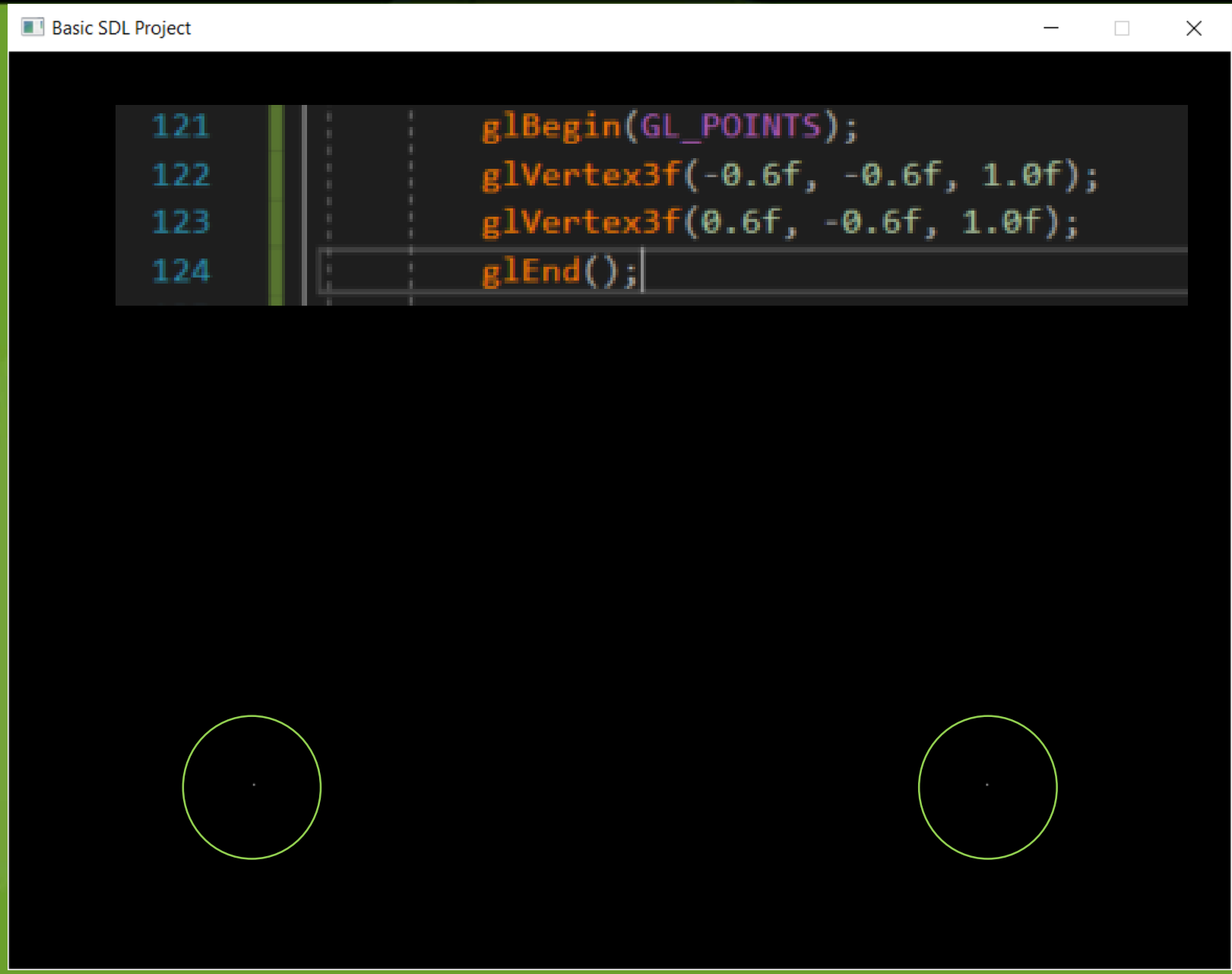



  - glVertex3f takes in 3 parameters, one for each vertex of our shape.
    - In this case our shape is only a point so it has only a single vertex.
  - Lastly to close out our shape and tell OpenGL we are done defining it's properties, we call glEnd();
    - glEnd takes no parameters.

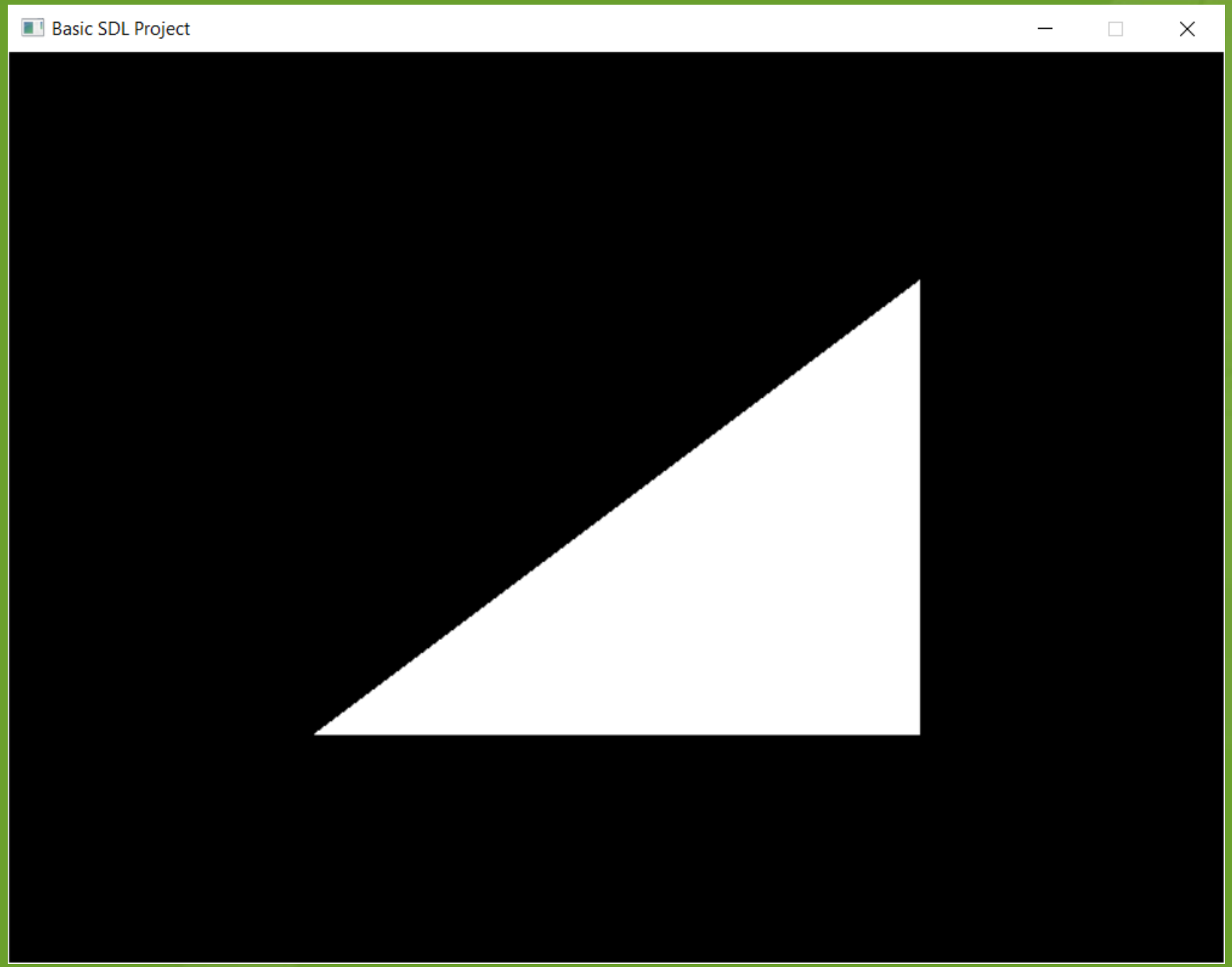
```
glBegin( GL_POINTS );
glVertex3f( -0.5f, -0.5f, 1.0f );
glEnd();
```

# Drawing - WHERE

- Notice that with the following code we have two tiny white dots drawn to screen near each bottom corner.
  - The screen coordinate system for openGL by default is defined as -1.0f -> 1.0f.
  - This is known as a normalized value.



# Drawing - WHERE



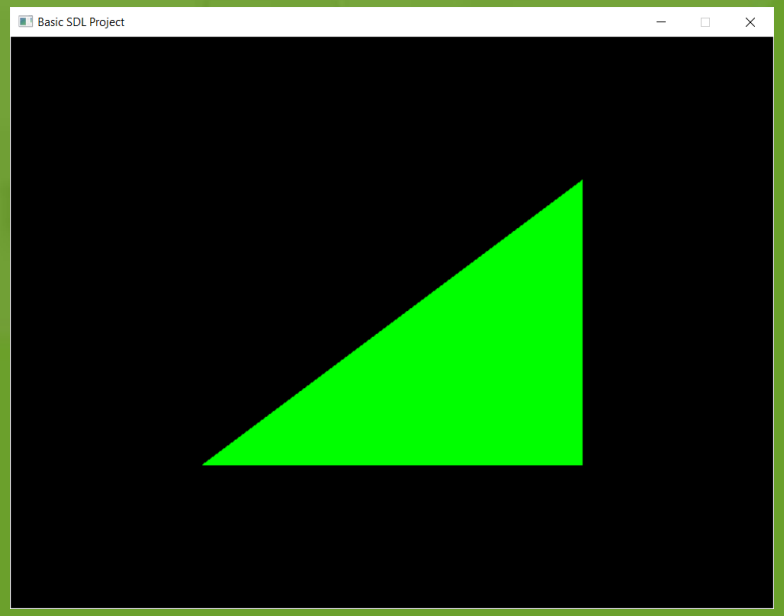

- We can use a series of 3 defined vertices inside a glBegin call to draw a triangle.
  - Each glVertex3f() corresponds to a point on the ends of the triangle.

```
glBegin(GL_TRIANGLES);
glVertex3f(-0.5f, -0.5f, 1.0f);
glVertex3f(0.5f, -0.5f, 1.0f);
glVertex3f(0.5f, 0.5f, 1.0f);
glEnd();
```

# Drawing - HOW

- The HOW of drawing is one of the more variable areas of our shape.
- Here we can set the size of our points, change the colour, define how it interacts with lights, textures etc…
  - For now we will focus on two easier concepts, point size and colour.
- To Change the Colour of our primitives we can use the glColor4f() function.
  - This function takes in 4 values for R,G,B & A

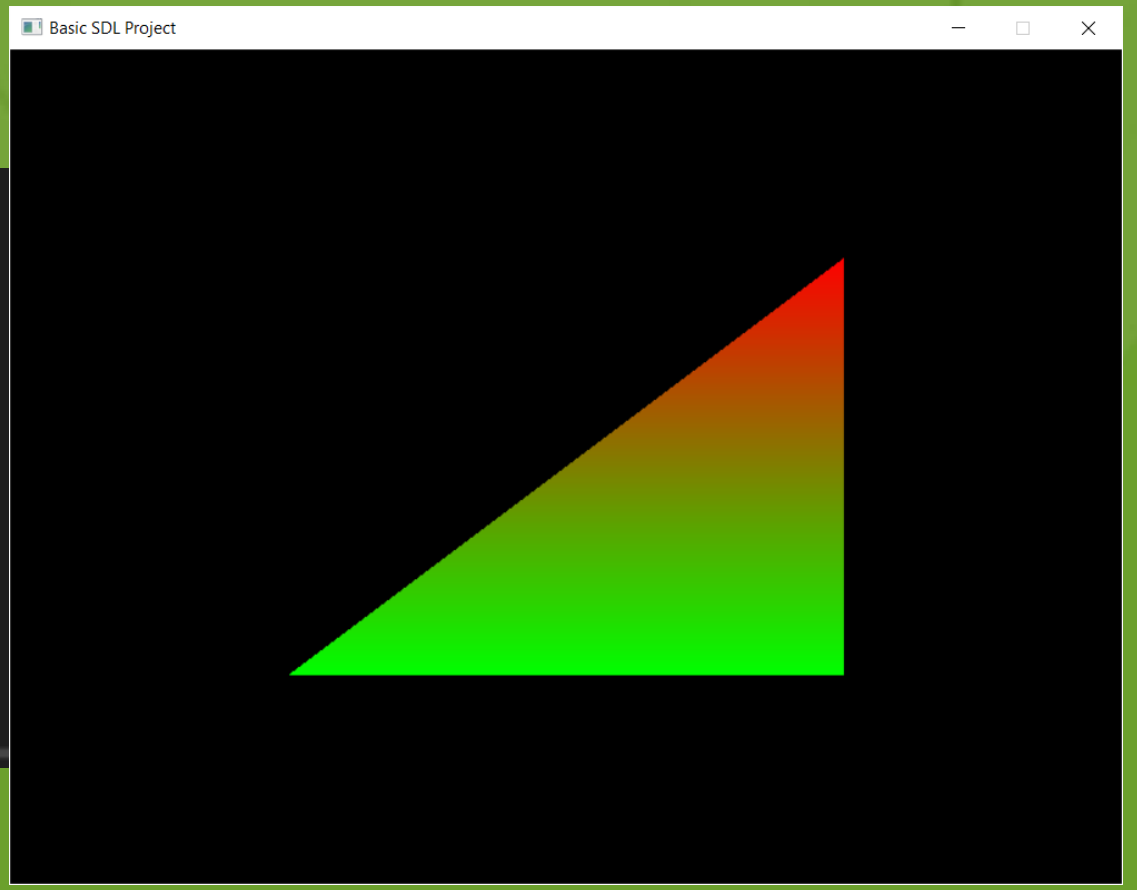
```
glBegin(GL_TRIANGLES);
glColor4f(0.0f, 1.0f, 0.0f, 1.0f);
glVertex3f(-0.5f, -0.5f, 1.0f);
glVertex3f(0.5f, -0.5f, 1.0f);
glVertex3f(0.5f, 0.5f, 1.0f);
glEnd();
```



# Drawing - HOW

- Note that if we change colour after a vertex is defined all subsequent vertices will use this colour:
  - We are changing the state of all future colouring.

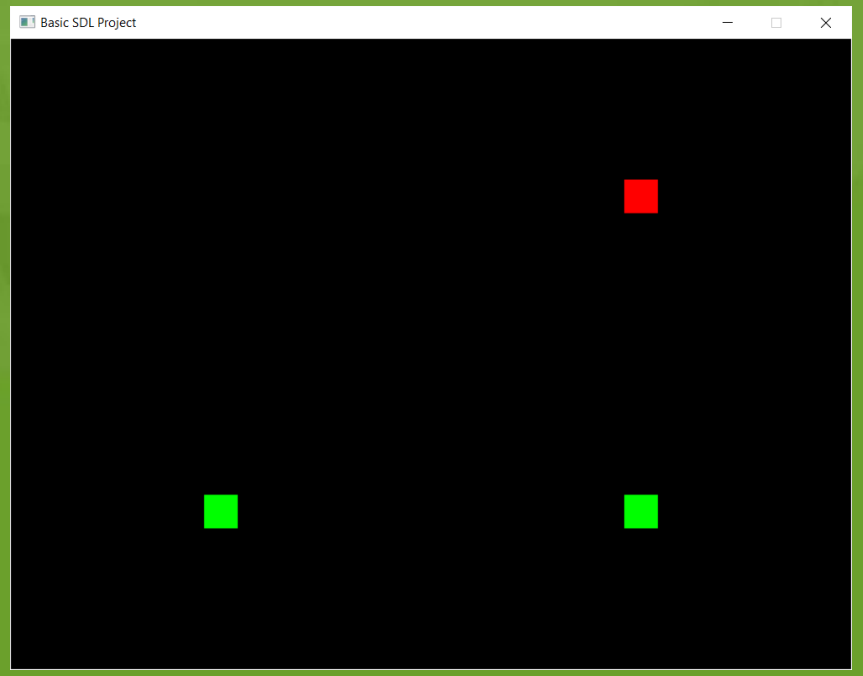
```
glBegin(GL_TRIANGLES);
glColor4f(0.0f, 1.0f, 0.0f, 1.0f);
glVertex3f(-0.5f, -0.5f, 1.0f);
glVertex3f(0.5f, -0.5f, 1.0f);
glColor4f(1.0f, 0.0f, 0.0f, 1.0f);
glVertex3f(0.5f, 0.5f, 1.0f);
glEnd();
```



# Drawing - HOW

- Outside of colouring, we can also change the size of our points.
  - For this we use the glPointSize() function which takes in a floating point size for the points themselves.
    - Again, this is state based so all subsequent points will be the size defined until the value is changed.

```
glPointSize(32.0f);
glBegin(GL_POINTS);
glColor4f(0.0f, 1.0f, 0.0f, 1.0f);
glVertex3f(-0.5f, -0.5f, 1.0f);
glVertex3f(0.5f, -0.5f, 1.0f);
glColor4f(1.0f, 0.0f, 0.0f, 1.0f);
glVertex3f(0.5f, 0.5f, 1.0f);
glEnd();
```



# SUMMARY

- We discussed simple 2D drawing in OpenGL:
  - Shapes
  - Colours
  - Concave vs Convex
  - Immediate mode