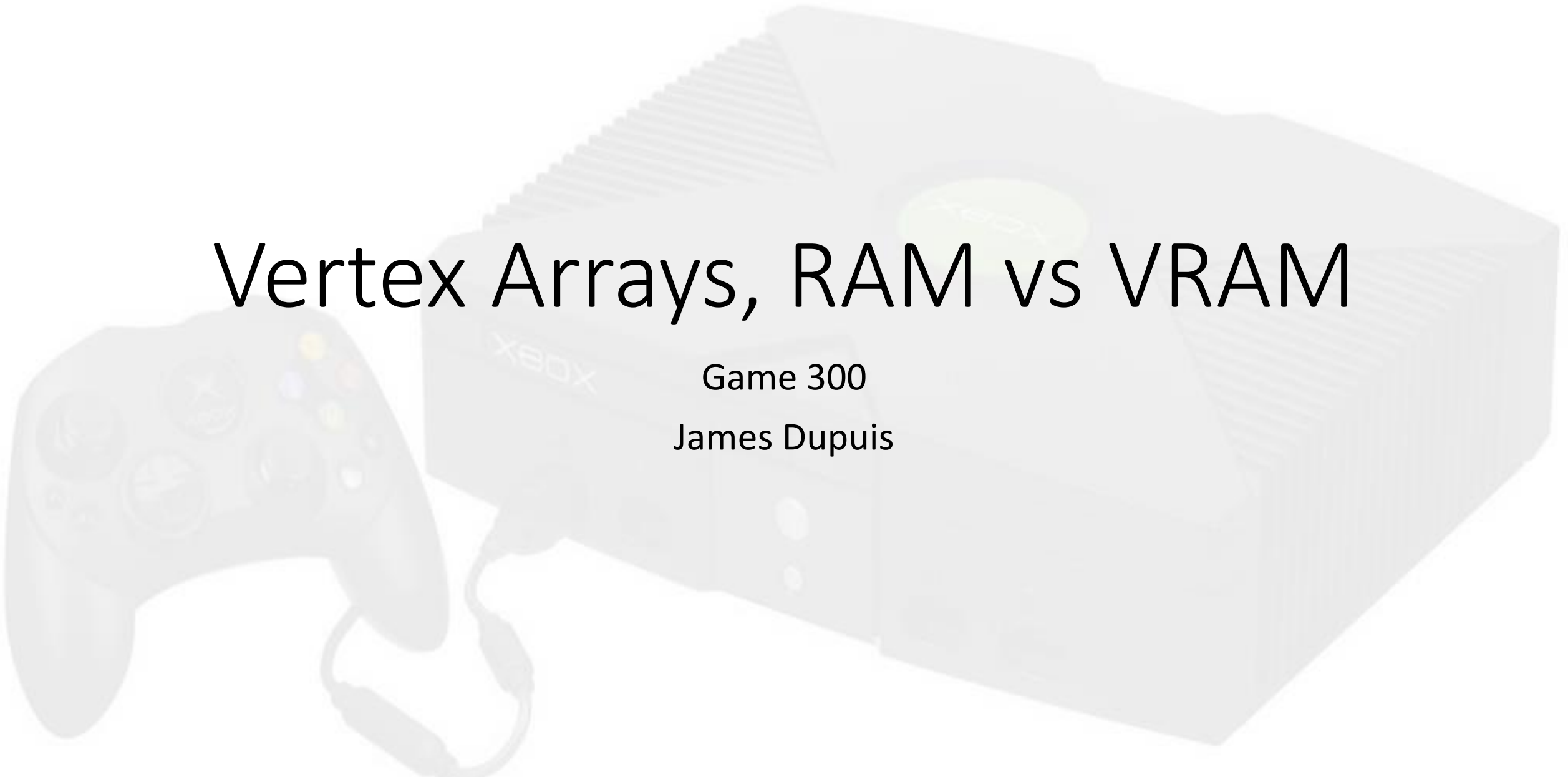
Vertex Arrays, RAM vs VRAM



Game 300

James Dupuis

Objectives

- Recap drawing in Immediate Mode.
- Change the size of lines drawn
- Use Vertex Arrays to send data to our openGL application
- Understand the difference between RAM and VRAM
- Learn ways to optimize data storages
- Use Buffers to store data in VRAM

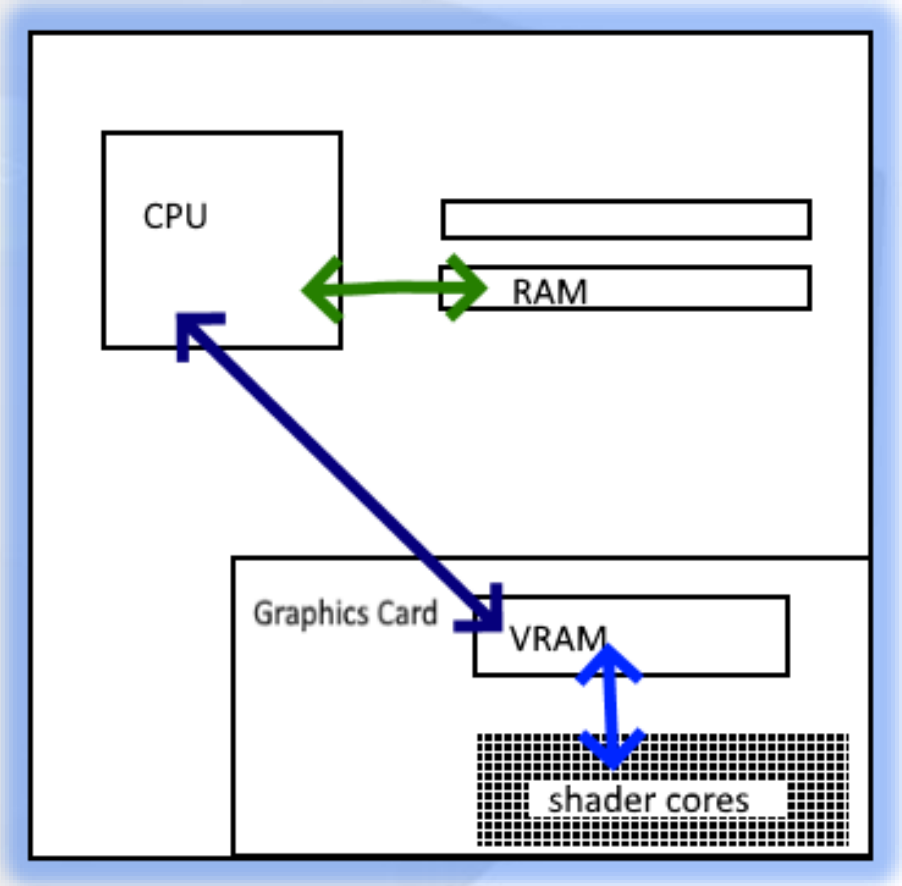
Line Sizes

- Just as we can change the size of our points we can also change the size of lines drawn with GL_LINES.
- To set the line size state we use the following function:
  - glLineWidth(float width);

```
glPointSize(32.0f);
glLineWidth(15.0f);
glBegin(GL_LINES);
glColor4f(0.0f, 1.0f, 0.0f, 1.0f);
glVertex3f(-0.5f, -0.5f, 1.0f);
glVertex3f(0.5f, -0.5f, 1.0f);
glEnd();

glLineWidth(5.0f);
glBegin(GL_LINES);
glColor4f(1.0f, 0.0f, 0.0f, 1.0f);
glVertex3f(-0.5f, 0.5f, 1.0f);
glVertex3f(0.5f, 0.5f, 1.0f);
glEnd();
```

RAM vs VRAM

- When we write a C++ application and allocate memory we occupy memory locations of our RAM.
  - This is the memory slots connected directly on our motherboards.
  - For data to be used by the graphics card it must be transported via the CPU from the RAM to our graphics card.
    - The Graphics card then proceeds to relay this information to the shaders.
  - When the graphics card is sent data to process, it stores this data on it's local memory known as VRAM.
  - VRAM is also used to hold the results of calculations made by the GPU cores.
    - This is often known as a frame buffer.



IMMEDIATE MODE PROBLEMS

- Vertex Arrays are a way of transporting data to the openGL application more efficiently.
- With our Current way in Immediate Mode to draw a triangle we have to specify each vertex individually like so:

```
glBegin(GL_TRIANGLES);
glVertex3f(0.0f, 0.5f, 1.0f);
glVertex3f(0.5f, -0.6f, 1.0f);
glVertex3f(0.5f, 0.6f, 1.0f);
glEnd();
```

- This means we have to call a similar function 3 times to draw a triangle.

VERTEX ARRAYS

- Imagine how many glVertex3f Calls would be required to draw Mario?
- When a model is loaded into a game, it's model information is loaded from a binary file on the hard drive to the computers RAM.
  - This data is typically loaded as an array of vertices.
  - It is often easier to just forward that data on as an array rather than break it apart to send through individual glVertex3f() calls.

- Vertex Arrays are a storage available to openGL applications which allows data to be directly transported to the OpenGL context.
- The code of a basic triangle as an array of vertices may look something like the following:

```
float triangleVertices[9] = {   0.0f, 0.5f, 0.0f,
                                0.5f, -0.6f, 0.0f,
                                0.5f, 0.6f, 0.0f, };
```

ENABLE ARRAYS

- If we want to use vertex Arrays inside of our OpenGL program, we must first let OpenGL know this.
  - By setting a state.
  - Previously we talked about states being set by glEnable() and glDisable and that some states required specific functions to set.
  - Vertex Arrays are one of these special states.
- We can tell OpenGL we want to draw with Vertex Arrays by calling:

```
glEnableClientState(GL_VERTEX_ARRAY);
```

- After we are done drawing we must also tell OpenGL we are done and ensure we reset our state using:

```
glDisableClientState(GL_VERTEX_ARRAY);
```

VERTEX POINTER

```
float triangleVertices[9] = {   0.0f, 0.5f, 0.0f,
                                0.5f, -0.6f, 0.0f,
                                0.5f, 0.6f, 0.0f, };
```

- After we have let OpenGL know that we want to use Vertex Arrays, next we need to tell OpenGL where our data is.
  - We can send our Data to the OpenGL Application using the following function:

```
glVertexPointer(3, GL_FLOAT, 0, &triangleVertices[0]);
```

Parameter Breakdown:

1. The amount of values per vertex. *(2d = 2, 3d = 3 or 4…)*
2. The format of the data.
3. The stride, this is used for when you have more data packed into one array.
   Our array just keeps track of the locations so 0 is fine.
4. The address location of the array.

DRAWING WITH ARRAYS

- Once we have told OpenGL **HOW** we want to draw and **WHERE** the data is, we are free to tell it to actually **draw**.
- We can accomplish this with the **glDrawArrays** function.

```
float triangleVertices[9] = {   0.0f, 0.5f, 0.0f,
                                0.5f, -0.6f, 0.0f,
                                0.5f, 0.6f, 0.0f, };

glEnableClientState(GL_VERTEX_ARRAY);
glVertexPointer(3, GL_FLOAT, 0, &triangleVertices[0]);
glDrawArrays(GL_TRIANGLES, 0, 3);
glDisableClientState(GL_VERTEX_ARRAY);
```

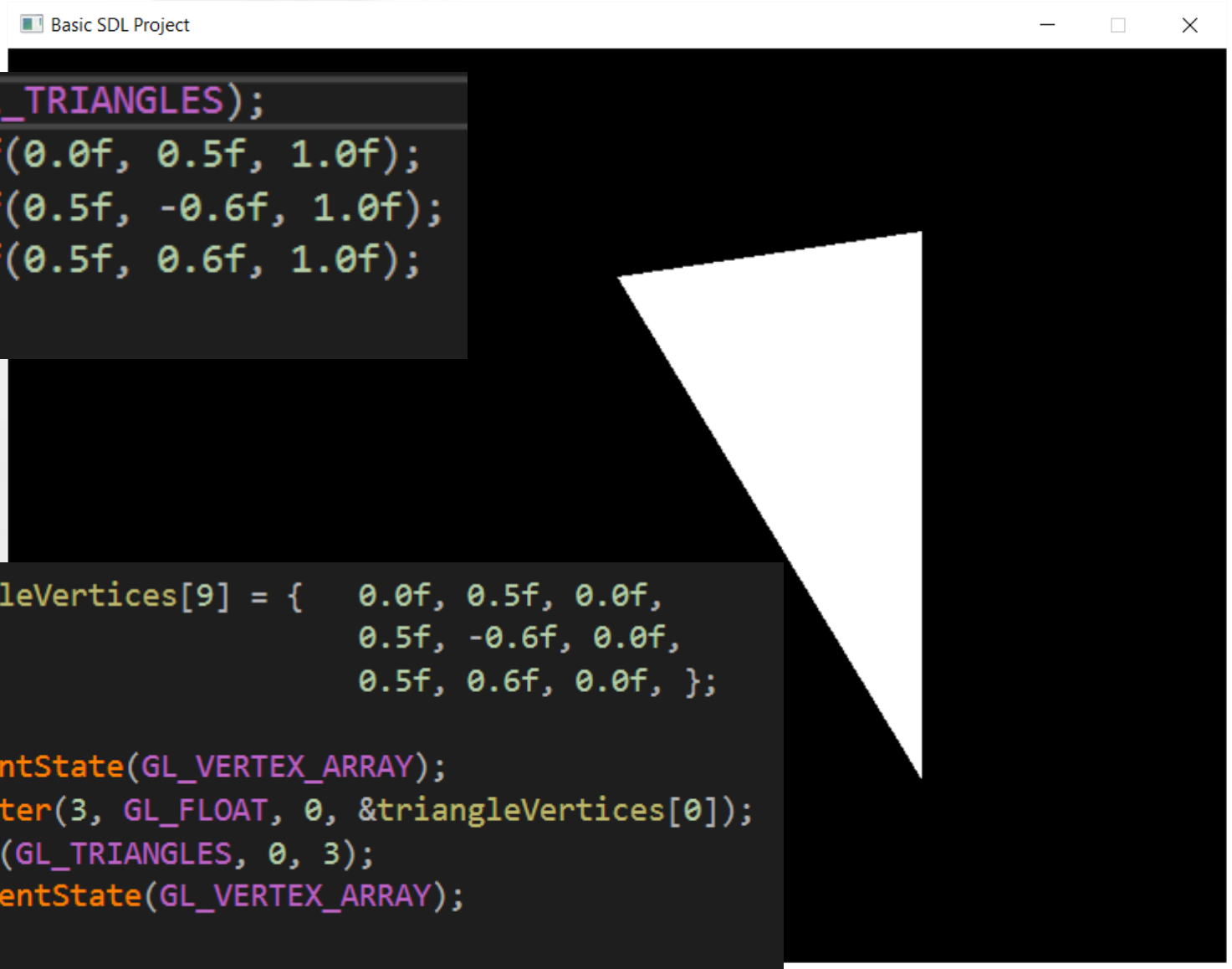
Parameter Breakdown:

1. The type of primitive to draw, *GL_POINTS, GL_LINES, GL_TRIANGLES, etc…*
2. The starting offset to begin drawing from the data supplied through the pointer.
3. The amount to process (3 in our case for number of vertices)

CODE COMPARISON

- The following two segments of code produce the same results.
- The difference is that the bottom code segment preloads all the vertices data at once into the OpenGL program.
  - More efficient.

```
glBegin(GL_TRIANGLES);
glVertex3f(0.0f, 0.5f, 1.0f);
glVertex3f(0.5f, -0.6f, 1.0f);
glVertex3f(0.5f, 0.6f, 1.0f);
glEnd();
```




```
float triangleVertices[9] = {   0.0f, 0.5f, 0.0f,
                                0.5f, -0.6f, 0.0f,
                                0.5f, 0.6f, 0.0f, };

glEnableClientState(GL_VERTEX_ARRAY);
glVertexPointer(3, GL_FLOAT, 0, &triangleVertices[0]);
glDrawArrays(GL_TRIANGLES, 0, 3);
glDisableClientState(GL_VERTEX_ARRAY);
```

COLOUR ARRAYS

- There are other types of arrays outside of vertexArrays.
  - We often still call them vertexArrays as the data often pertains to each vertex.
- Each one of these Arrays has it's own value to be set for ClientState in addition to it's own Pointer function.

```
glEnableClientState(GL_COLOR_ARRAY);
glEnableClientState(GL_VERTEX_ARRAY);
glEnableClientState(GL_NORMAL_ARRAY);
```

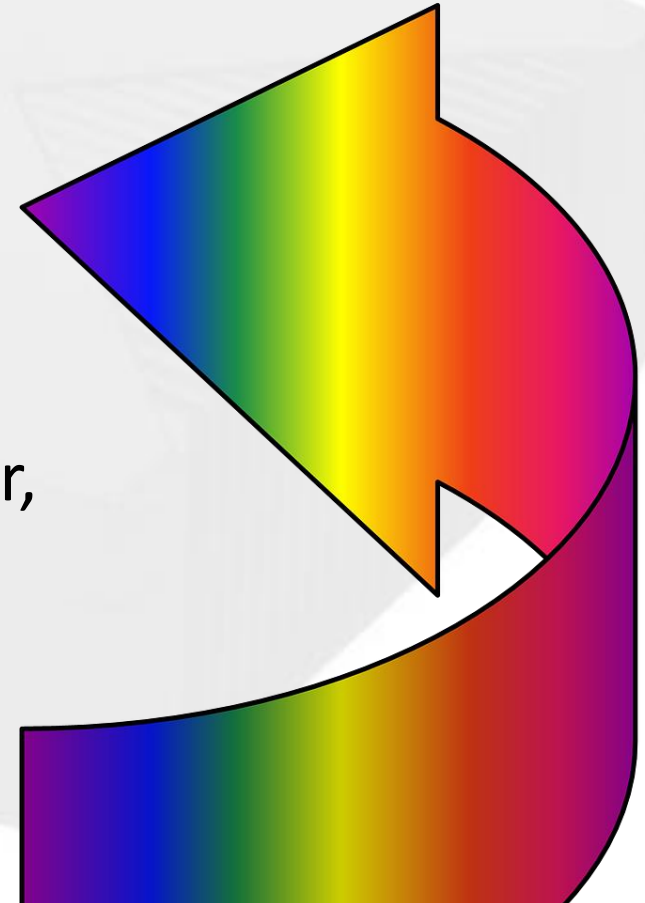
- Note that these values are not Bitwise Or-able.
  - They will each need their own call to EnableClientState() & DisableClientState()

COLOUR POINTERS

- The Pointer functions to set data are very similar to that of the glVertexPointer(), however, each has it's own functionality and slight changes to requirements to parameters.
  - Take a look at the following:

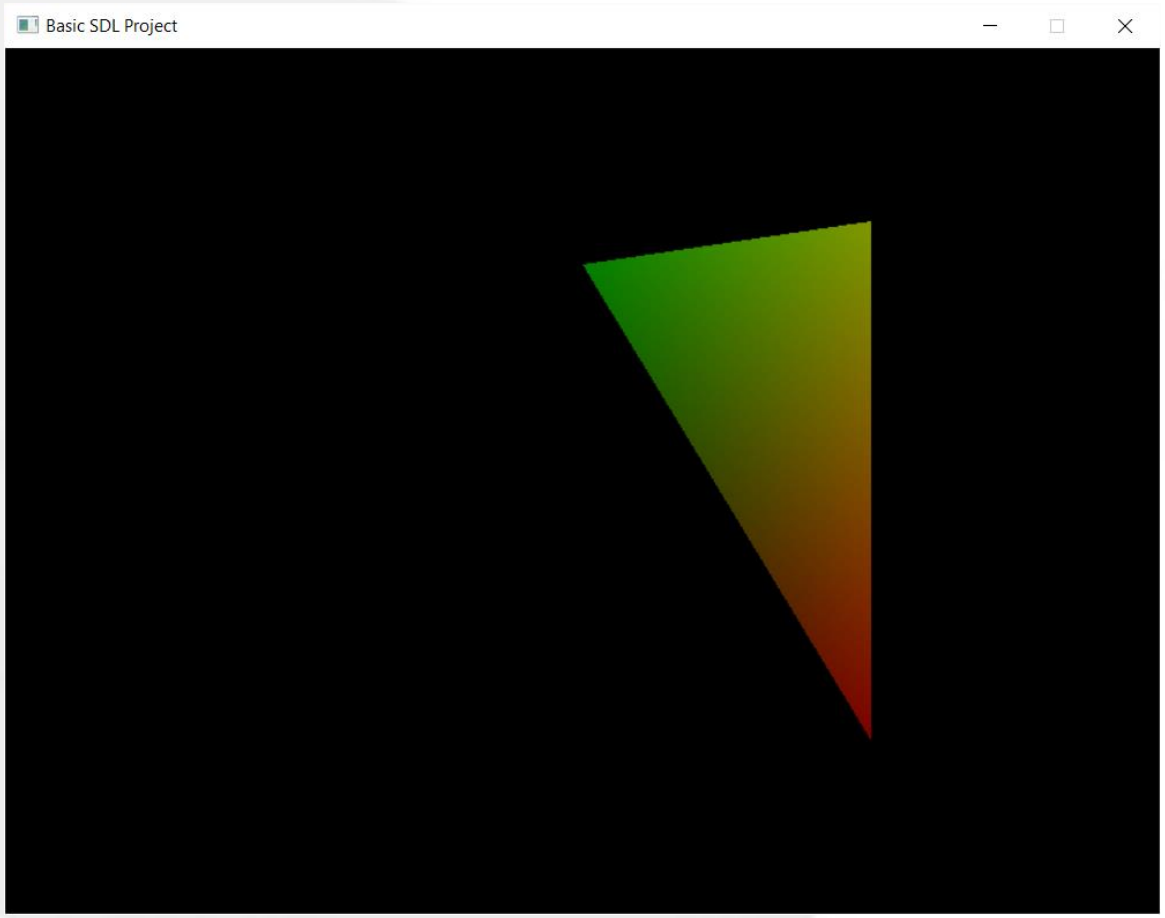
```
glColorPointer(4, GL_FLOAT, 0, &triangleColours[0]);
glNormalPointer(GL_FLOAT, 0, &triangleNormals[0]);
glVertexPointer(3, GL_FLOAT, 0, &triangleVertices[0]);
```

- Note that the normalPointer omits the first parameter, this is because normal values should always be represented in a set of 3 values.

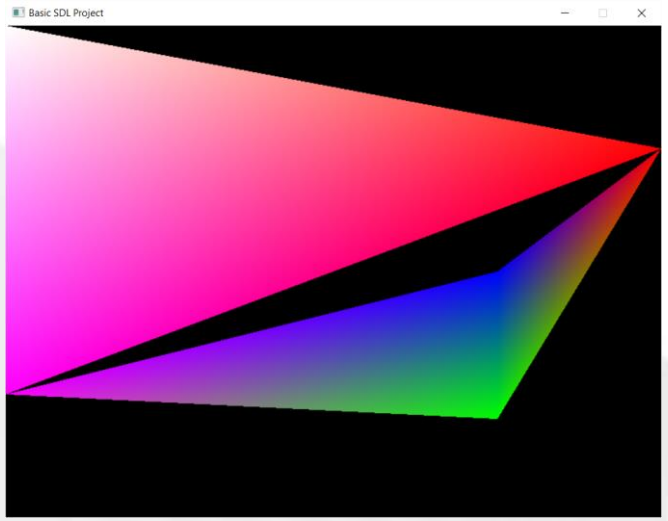


DRAWING COLOURS WITH ARRAYS

```
float triangleVertices[9] = {   0.0f, 0.5f, 0.0f,
                                0.5f, -0.6f, 0.0f,
                                0.5f, 0.6f, 0.0f, };

float triangleColours[12] = { 0.0f, 0.5f, 0.0f, 1.0f,
                              0.5f, -0.6f, 0.0f, 1.0f,
                              0.5f, 0.6f, 0.0f, 1.0f, };

glEnableClientState(GL_VERTEX_ARRAY);
glEnableClientState(GL_COLOR_ARRAY);

glColorPointer(4, GL_FLOAT, 0, &triangleColours[0]);
glVertexPointer(3, GL_FLOAT, 0, &triangleVertices[0]);
glDrawArrays(GL_TRIANGLES, 0, 3);

glDisableClientState(GL_COLOR_ARRAY);
glDisableClientState(GL_VERTEX_ARRAY);
```



Multiple Triangles



- Suppose we wanted to draw more than one triangle and share vertices, there are two ways we can do this.
  - We can use a TRIANGLE_FAN or TRIANGLE_STRIP.
    - But we may not get the same results as we want.
  - We can add more triangles vertices to our array.
  - We would then do the same with our colours.
  - Lastly we would need to update our call to glDrawArrays num vertices:

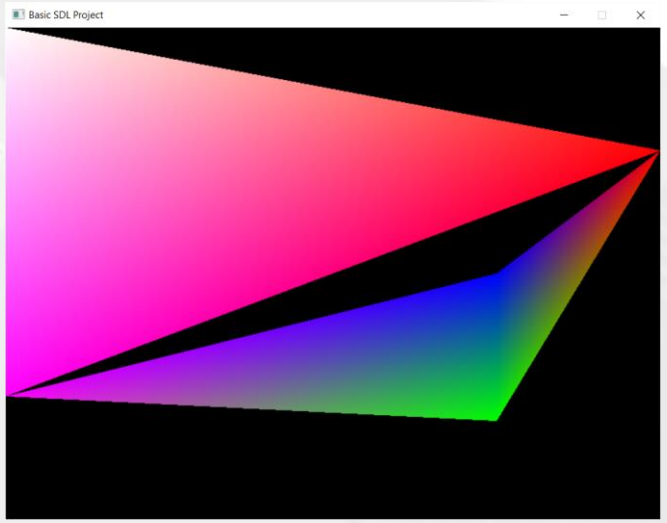
```
glDrawArrays(GL_TRIANGLES, 0, 9);
```

```
float triangleVertices[27] = {  1.0f, 0.5f, 0.0f, // triangle 1
                                0.5f, -0.6f, 0.0f,
                                0.5f, 0.6f, 0.0f,

                                0.5f, 0.6f, 0.0f, // triangle 2
                                0.5f, -0.6f, 0.0f,
                                -1.0f, -0.5f, 0.0f,

                                1.0f, 0.5f, 0.0f, // triangle 3
                                -1.0f, -0.5f, 0.0f,
                                -1.0f, 1.0f, 0.0f };
```

PROBLEMS WITH ARRAYS

- The problem with this is that we are duplicating a lot of data:

```
float triangleVertices[27] = {  1.0f, 0.5f, 0.0f, // triangle 1
                                0.5f, -0.6f, 0.0f,
                                0.5f, 0.6f, 0.0f,

                                0.5f, 0.6f, 0.0f, // triangle 2
                                0.5f, -0.6f, 0.0f,
                                -1.0f, -0.5f, 0.0f,

                                1.0f, 0.5f, 0.0f, // triangle 3
                                -1.0f, -0.5f, 0.0f,
                                -1.0f, 1.0f, 0.0f };
```



- Really all we need here is a total of 5 vertices.

```
float triangleVertices[15] = { 1.0f, 0.5f, 0.0f,
                               0.5f, -0.6f, 0.0f,
                               0.5f, 0.0f, 0.0f,
                               -1.0f, -0.5f, 0.0f,
                               -1.0f, 1.0f, 0.0f };
```

PROBLEMS WITH ARRAYS

- We can reduce it to 5 vertices (15 floats) if we use the function glDrawElements() instead of glDrawArrays().

```
glDrawArrays(GL_TRIANGLES, 0, 9);
```

VS

```
glDrawElements(GL_TRIANGLES, 9, GL_UNSIGNED_INT, &vertIndeces[0]);
```

- glDrawElements has no offset parameter like glDrawArrays as we will dictate this by our last parameter.

Parameter Breakdown (final two):

1. The type of array defining the indices to use.
2. A pointer to the location of an array defining our indices.

BEFORE & AFTER

```
float triangleVertices[27] = {  1.0f, 0.5f, 0.0f, // triangle 1
                                0.5f, -0.6f, 0.0f,
                                0.5f, 0.0f, 0.0f,

                                0.5f, 0.0f, 0.0f, // tri
                                0.5f, -0.6f, 0.0f,
                                -1.0f, -0.5f, 0.0f,

                                1.0f, 0.5f, 0.0f, // tri
                                -1.0f, -0.5f, 0.0f,
                                -1.0f, 1.0f, 0.0f };

float triangleColours[36] = { 1.0f, 0.0f, 0.0f, 1.0f, //
                              0.0f, 1.0f, 0.0f, 1.0f,
                              0.0f, 0.0f, 1.0f, 1.0f,

                              0.0f, 0.0f, 1.0f, 1.0f,
                              0.0f, 1.0f, 0.0f, 1.0f,
                              1.0f, 0.0f, 1.0f, 1.0f,

                              1.0f, 0.0f, 0.0f, 1.0f, // triangle 3
                              1.0f, 0.0f, 1.0f, 1.0f,
                              1.0f, 1.0f, 1.0f, 1.0f };
```

```
float triangleVertices[15] = { 1.0f, 0.5f, 0.0f,
                               0.5f, -0.6f, 0.0f,
                               0.5f, 0.0f, 0.0f,
                               -1.0f, -0.5f, 0.0f,
                               -1.0f, 1.0f, 0.0f };

float triangleColours[20] = { 1.0f, 0.0f, 0.0f, 1.0f,
                              0.0f, 1.0f, 0.0f, 1.0f,
                              0.0f, 0.0f, 1.0f, 1.0f,
                              1.0f, 0.0f, 1.0f, 1.0f,
                              1.0f, 1.0f, 1.0f, 1.0f };


unsigned int vertIndeces[9] = { 0, 1, 2, // tri 1
                                2, 1, 3, // tri 2
                                0, 3, 4  // tri 3
};
```

ALTOGETHER

- Finally our code to draw 3 triangles, with multiple colours optimized for minimized data transfer.

```
float triangleVertices[15] = { 1.0f, 0.5f, 0.0f,
                               0.5f, -0.6f, 0.0f,
                               0.5f, 0.0f, 0.0f,
                               -1.0f, -0.5f, 0.0f,
                               -1.0f, 1.0f, 0.0f };

float triangleColours[20] = { 1.0f, 0.0f, 0.0f, 1.0f,
                              0.0f, 1.0f, 0.0f, 1.0f,
                              0.0f, 0.0f, 1.0f, 1.0f,
                              1.0f, 0.0f, 1.0f, 1.0f,
                              1.0f, 1.0f, 1.0f, 1.0f };


unsigned int vertIndeces[9] = { 0, 1, 2, // tri 1
                                2, 1, 3, // tri 2
                                0, 3, 4  // tri 3
};

glEnableClientState(GL_COLOR_ARRAY);
glEnableClientState(GL_VERTEX_ARRAY);

glColorPointer(4, GL_FLOAT, 0, &triangleColours[0]);
glVertexPointer(3, GL_FLOAT, 0, &triangleVertices[0]);

glDrawElements(GL_TRIANGLES, 9, GL_UNSIGNED_INT, &vertIndeces[0]);

glDisableClientState(GL_COLOR_ARRAY);
glDisableClientState(GL_VERTEX_ARRAY);
```

Summary

- Recapped drawing in Immediate Mode.
- Changed the size of lines drawn
- Used Vertex Arrays to send data to our openGL application
- Understand the difference between RAM and VRAM
- Learn ways to optimize data storages