The concept of 4D within OpenGL

J Y HOOD
10/09/24

The following you are about to read is the project I wish to make to improve my skills in the coding field after a long break. I want to get back into the swing of coding, I found redoing the basics to be pretty boring and wished to challenge myself just enough to get said confidence to possibly start a new project involving game development somehow, it should be mentioned while I do plan to use OpenGL https://en.wikipedia.org/wiki/OpenGL

I hope to return to the game engine of unity and perhaps do the odd thing with GL in the future. Iv'e already started the project however after using Github https://en.wikipedia.org/wiki/GitHub Trying to switch between my main PC and my old laptop
(where all the libraries for GL were created)

Cloning the project back to my PC broke all the current files which is why I'm taking the opportunity to write this document explaining the process, perhaps thinking of it more like a diary of sorts.

Currently the project sits at I would roughly 12 to 15% Now despite the implication the title the project is only 4D *in concept* the application will not run an actual 4D effect rather an 3D model of a tesseract https://en.wikipedia.org/wiki/Tesseract

I'll proceed this doc with the code explanation and while I use ChatGPT and other resources to confirm things, I'll be still writing this in my own words! This is an experiment within itself so I can remember more concepts and context. I'm so sick of not being able to remember these things so I'm taking the last route I can think of. This may take up more time but if done right I'll always have a resource to fall back on.

Now the project is simple

A window pop up that the user can control left and right (the x axis) the user will be shown

A 2D plane

A 3D cube

A 4D tesseract (as mentioned above)

Like a virtual library it shows an example of 2D to 4D but also OpenGL at it's best.

1. The libraries .cpp

```
VitualWalkthrough                                   (Global Scope)
1  #include<iostream> // This includes the Input/Output Stream library
2  #include<glad/glad.h> // This includes the GLAD (OpenGL Loader) library
3  #include<GLFW/glfw3.h> // This includes the GLFW library, which is used for creating windows
```

I start with the use of *iostream* this is really used in pretty much every c++ console program as it allows the basics such as input and output to the main console.

“glad/glad” is OpenGL's main library of source (none of the syntax would work without it)
Styax https://en.wikipedia.org/wiki/Syntax_(programming_languages)

“GLFW/glfw3.h” is a version of the GLFW library, this is pretty including everything with the application windows (not to be confused with OpenGL which is the graphics library)

```
23  int main() // Creates and sets as main function
24  {
25      glfwInit(); // creates window
26
27      glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3); // This ensures that the window will use OpenGL version 3.x (major versio
28      glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3); // This ensures that the window will use OpenGL version 3.3 (minor versio
29      glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE); // GL needs to know which profile to use
```

The main function is written “int main” which of course will hold everything needed in the context of the project

“glfwInit()” is declared as a function, think of it as an on button for my tools within GLFW I’m about to use (aka anything related to the graphics) (ignore the comment I need to fix!)

Next you’ll see three different types of “glfwWindowHint” These all have slight differences
Major
Minor
And Core Profile

Major ensures that the popup window application is using that major version 3.
Minor is the .3 on the end making it 3.3 the version OpenGL will use

The same pretty much applies to the core profile
(the core profile uses the main graphics pipeline, source code etc and the last release of this was in 2014)

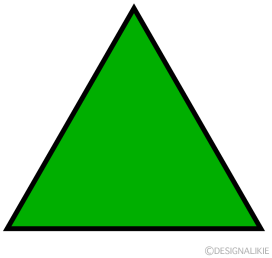
```
GLfloat vertices[] = // An array that will concist of the vertex position points, GLfloat like any float is
{
    -0.5f, -0.5f * float(sqrt(3)) / 3, 0.0f,
    0.5f, -0.5 * float(sqrt(3)) / 3, 0.0f,
    0.0f, 0.5f * float(sqrt(3)) * 2 / 3, 0.0f
};
```

Things get a bit complicated here but what you read is “GLfloat vertices[]” is an array of floating point numbers https://www.freecodecamp.org/news/floating-point-definition/

Now the *GLfloat* is already defined via both the library and “glfwInit” (I think) however the keyword “vertices” is defined here (this could be called anything) but of course this relates to the name in the context

The Vertex Shader https://www.khronos.org/opengl/wiki/Vertex_Shader

Think of a triangle! That’s what all these -0.5 and 0.5 are related to (coordinates)



X
Y
Z

The basics of math in computers but in this case the vertices are the points you would see in the triangle. *Tdlr you are playing Connect the dots only you place the dots to begin with.*

Now as you know I’m trying to create a 2D *Plane* here, at the current moment in time I’m simply following a tut on how all this syntax so I’ll be back with an updated result, again keep in mind that this is the coordinates but it won’t spawn the graphics of the triangle itself.

We can’t play connect the dots until we find the pencil.

(not a fucking clue what sqrt *square route* is meant to be doing though)

```
39
40      GLFWwindow* window = glfwCreateWindow(800, 800, "Vitual4DSpace"/*is name of window!*/, NULL, NULL); //
41      if (window == NULL) // fail safe net
42      {
43          std::cout << "Failed to create GL Window" << std::endl;
44          glfwTerminate(); // closes/terminates window
45          return -1;
46      }
```

GLFWwindow is another already defined command in the syntax much like the *GLfloat*