NEA Project

Contents

[1 Analysis Section 1](#_Toc176359764)

[1.1 Project Identification and Background 1](#_Toc176359765)

[1.2 Objectives 2](#_Toc176359766)

[1.3 Research 3](#_Toc176359767)

[1.3.1 Interviews/Survey 3](#_Toc176359768)

[1.3.2 Technical/Skills Research 3](#_Toc176359769)

[1.4 Research Table 3](#_Toc176359770)

[1.5 Story Board 3](#_Toc176359771)

[1.6 Prototypes 3](#_Toc176359772)

[1.6.1 Justification of chosen solution 3](#_Toc176359773)

[2 Development Evidence 3](#_Toc176359774)

[2.1 Complex Code 3](#_Toc176359775)

[2.2 Classes 3](#_Toc176359776)

# Analysis Section

## Project Identification and Background

My project will be a 4-dimensional (4D) graphics engine inspired by 4D toys ([4D Toys. An interactive toy for 4D children.](https://4dtoys.com/)). It will allow the user to explore a 4D space and add/remove/interact with 4D hyper-shapes. Since visualising 4D space is not possible, I will instead show 3D ‘slices’ of the 4D space and allow the user to move along the 4th dimension to change which ‘slice’ they can see. In addition to this, I will have an analogous way of showing a 3D space by using 2D slices. This will be to help me and the user to understand better what is going on in the 4D version. I will use a 3D graphics API (OpenGL) in order to show the 3D slices and will use this as well to show the 2D slices as it will give me greater control over the vertices of the shapes as opposed to the primitive shape types in JavaFX. OpenGL is a common graphics API in C++ but my project will be in java so I will use LWJGL (Light weight java game library), a java wrapper for OpenGL. I will also use JOML (Java OpenGL maths library) for the vector maths that I will use. The user will have complete control over the camera, being able to move through all of the available dimensions and in the 4D case, rotate the camera as well (most likely rotation along the 4th dimension will cause lots of confusion and render control impossible so this will be tried but likely not possible in the final version of the program. My motivation for making this is primarily educational, trying to understand how the 4th dimension works through extrapolating how a 3 dimensional world is viewed from a 2 dimensional perspective. This desire to learn this was initially inspired by 4D toys. I also have an interest in 3D graphics so researching and implementing it is also educational.

## Objectives

1. Set up GLFW window
   1. GLFW (Graphics Library Framework) should be initialised. This handles everything to do with the window.
   2. The window should be created with width and height dependant on the size of the users screen.
2. Game loop
   1. Call an update method periodically based on ups (updates per second), updates are for game states themselves, e.g. for updating object positions
   2. Call a render method periodically based on fps (frames per second), frames are the images that get displayed on the screen.
3. 3D Graphics
   1. Draw Triangles
      1. Shaders should be able to be loaded, compiled and attached to an OpenGL program. Shaders are what is used to control the frames.
      2. Relevant data needs to be buffered to the GPU
         1. Each mesh in the scene has a VAO (Vertex array object which holds all the information about the vertices) and multiple VBOs (Vertex buffer objects which store one type of information about the vertices e.g. positions, colour)
            1. The position of each vertex relative to the mesh needs to be buffered.
            2. The corresponding colour of these vertices needs to be buffered.
            3. Instructions on how to combine these vertices into triangles to make up the mesh needs to be buffered. This is in the form of indices pointing to the positions of the vertices. They will come in groups of 3s to make triangles.
      3. OpenGL needs to be told how to interpret the data and then draw it.
   2. Draw 2D models
      1. Unique models should be objects in code which abstracts away the information about VAOs and VBOs and how they interact with OpenGL. They should have a mesh which contains all of this information as well as a unique identifier (id) to easily attach the model to entities.
      2. The scene should be populated with entities which each have a model (multiple entities can share the same model).
         1. The entities should have a position, rotation, and scale which should be combined into a single model matrix.
         2. The model matrix is used to transform the positions of the vertices in the mesh (stored in the model used by the entity) from their local coordinates (defined relative to the mesh) to world coordinates (defined relative to an arbitrary origin in the scene).
         3. Entities should also store the id of the model that they use.
      3. The Renderer should loop through all of the entities in the scene, extract the models that they use and render the corresponding mesh.
   3. Draw 3D models
      1. The scene should have a perspective projection matrix which simulates a camera, being defined by an FOV, aspect ratio, and near and far clipping planes.
      2. This matrix is used to transform the vertices in world space to screen space to be displayed on the screen. The matrix makes further-away objects appear smaller than closer objects of the same size in world space.

## Research

### Interviews/Survey

### Technical/Skills Research

<https://ahbejarano.gitbook.io/lwjglgamedev> - OpenGL (3D graphics API)

## Research Table

## Story Board

## Prototypes

Attempt at creating a 3D ‘game’ based off of Minecraft where you can place cubes in a 3D world. Helped to learn a lot about OpenGL and graphics rendering as opposed to copying from a tutorial.

A brick wall with grass and bricks

Description automatically generated

This also helped with learning how to abstract the information about vertices into a single class as well as how to apply textures. This means that I am able to use textures in my models if I want to.

Another project where I explored OpenGL in more detail was a program where you can drive a car around a world and the floor will ‘follow’ you. By this I mean as you move, sections of the floor get added and removed dynamically based on your position.

### Justification of chosen solution

# Development Evidence

## Complex Code

## Classes