Sean O’Hagan

GyroScope

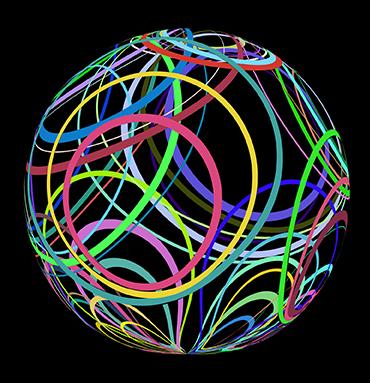
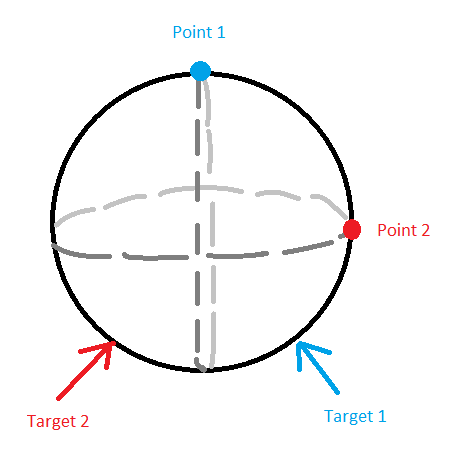


Image Designed by Kjpargeter / Freepik

# Background

The “core” of this project is a (cross-platform) imaging app with a single, stationary 3-dimensional plane upon which a single object can be rotated. In the application’s menus, the method of rotation (Euler angle, Matrix, Quaternion) will be selectable and tooltips will provide explanations for how the math is being calculated behind the scenes (including advantages/drawbacks of each method).

The application will center around a simplistic game: there will be one single object, floating centered in the plane, with 2 distinctly colored points on its surface, placed at intersections of the object’s canonical x, y, and z axes. In the plane, target points will appear - the object must be rotated so that its colored points align with the correct target point (checks will occur to ensure target points are valid - requiring rotations, but not scales or transforms for the points to align).



# Data Store Design

Data necessary for the program to run will be stored as binary files in the project’s ‘Utility’ directory - this includes shader programs written in GLSL (.vert, .frag, .comp) and vertex data for models (.obj). A shader management subsystem will allow for hot loading, compiling, and error checking of shader files. A simple loader function will import the vertex data to a memory buffer for use in the rendering process.

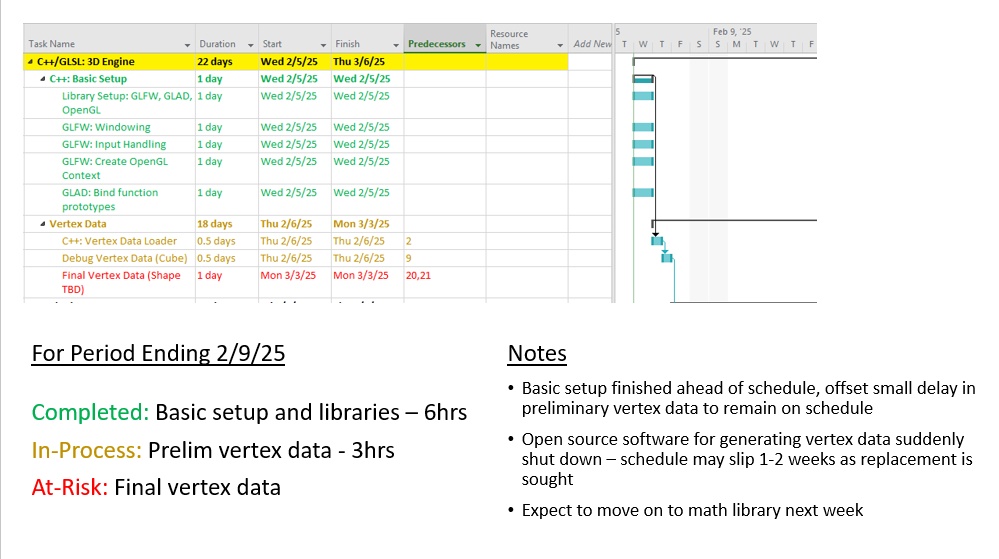
|  |
| --- |

# Screen Mockups

| Application Screen |  |
| --- | --- |
|  | This screen will be the core of the application. Other screens may be added if time allows, but only after this central interface has been implemented |

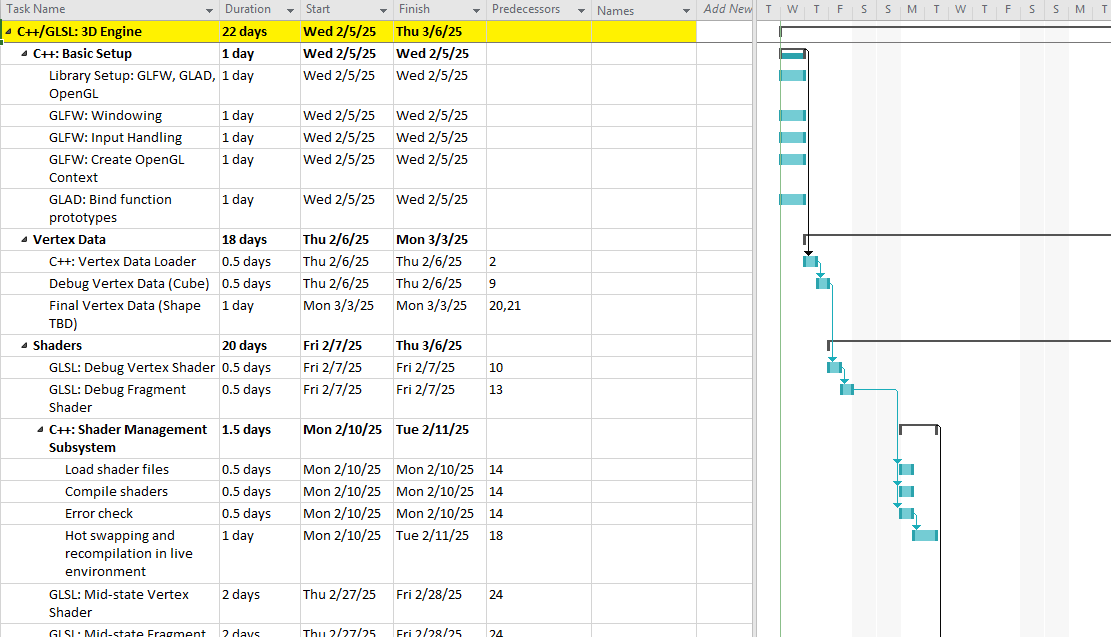
# Report Mockups

See O’HaganS\_ReportTemplate.pptx



# Work Breakdown Structure (WBS)

See O’HaganS\_MasterSchedule.mpp



# Summary

Currently, the core project is scheduled to conclude roughly 1 month prior to the delivery date (see MasterSchedule.mpp) with possible Stretch Goals estimated to extend 1 month beyond the delivery date, if pursued.

The rationale here is that there’s enough wiggle room in the core project to robustly address unforeseen issues without putting the delivery date at-risk. But, if all goes well and the core project finishes early, there are enough tasks to ensure development is active and purposeful all the way through to the delivery date.

Of note, many of these tasks are much more parallelizable than the schedule reflects, due to the fact that only one developer will be working on the project. If multiple developers were available, the schedule could be pulled in by parallelizing the core (yellow highlighted) modules first, and then any subsystems (in bold) as staffing allows.