Technical Design Document

Sync and Ecco

Contents

Version History

Date (D/M)	**Name/s**	**Notes**
23/07	Lochie	Made the document and filled in basic information
14/08	Tom	Updated Subtitle. Updated Game Concept Added Technical Goals: - Custom CPP Engine. - OpenGL Render Pipeline. - Custom Physics Engine. Added Technical Risks: - Custom CPP Engine. - OpenGL Render Pipeline. - Custom Physics Engine. Added Engine Systems (not filled out)
22/08	Lochie	Added risk mitigation to each of the Technical Risks - Custom CPP Engine Risk Mitigation - OpenGL Risk Mitigation - Custom Physics Engine Risk Mitigation Added Third-Party Programming Libraries / Tools
27/08	Lochie	Added links for Input and Engine Systems
1/09	Tom	Added Render System
4/12	Lochie	General revision for the whole of the document
5/12	Lochie	Added Windows

Game Details

Game Name: **Sync & Ecco**

Team Name: **Basalt Formations**

Team Members

Name	**Titles**
Lochlan (Lochie) McDonald	Programmer
Thomas (Tom) O'Brien	Programmer
Brodie Frazier	Designer
Austin Lanyon	Designer
Kian Kelly	Designer
Rachel Missingham	Artist
Lauren Trinh	Artist
Hachi Dinh	Artist

Game Concept

Sync and Ecco is a 2-player local co-op twin-stick top-down shooter, emphasising asymmetric gameplay and co-op reliance.

Technical Goals

Custom CPP Engine

- Build an engine in C++, using no out-of-the-box engine.
- Have the engine primarily use an Entity Component System.
- Empower designers and artists with tools to interact within the engine.

OpenGL Render Pipeline

- Rendering through OpenGL, using in-house created shaders.
- Explore and create lighting and material shaders typically inaccessible or challenging in out-of-the-box render pipelines.

Custom Physics Engine

- Create a custom physics engine to suit the needs of the project.

Technical Risks

Custom CPP Engine

- Dedicating development time to the engine will limit the time allowed for developing gameplay functionality, this could lead to not enough time for gameplay and too much on the engine.

Risk Mitigation

- We should ensure that gameplay milestones are getting hit.
- Not all programmers should be working directly on engine stuff simultaneously, other team members should always have a programmer to ask about the progress of gameplay mechanics.
- Plan out the features required for the engine as far ahead of time as possible to ensure that any future engine feature will not be a surprise and can prepared for.

OpenGL Render Pipeline

- Making the render pipeline steps ourselves could lead to inefficient and hamstrung methods that are obfuscated or overcome in out-of-the-box rendering systems.

Risk Mitigation

- Confer with people familiar with rendering and engine architecture, especially for anything complex.
- After any rendering change, make sure that there is no significant drop in performance.

Custom Physics Engine

- Using a custom physics engine could lead to mistakes being made that would be solved by using a library from the start.

Risk Mitigation

- Physics should also be tested in a game-less environment, ensuring that the limits of the implemented physics engine are known and should not be hit in normal gameplay.
- Most of the physics should be done early, so other features that rely upon it are tested correctly to reveal potential issues.

System Requirements

Recommended PC Specifications

Operating System Version	Windows 10 or later
CPU	Intel(R) Core(TM) i7-6700 CPU or better
GPU	NVIDIA GeForce RTX 3050 Ti or better
Additional Requirements	Requires Microsoft Visual C++ Redistributable runtimes

Required Hardware

- 2 Xbox Controllers

Third-Party Tools

Tools	**Purpose**
Visual Studio Community 2022	IDE
Google Docs	Documentation
Google Sheets	Scheduling, Documentation
Google Slides & Microsoft Powerpoint	Presentations, Documentation
HacknPlan	Team Management
Google Forms	QA
Github & Github Desktop & Source Tree	Version Control
Adobe Photoshop	Art
Adobe Illustrator	Art
Autodesk Maya & Blender	Art
Substance Painter	Art
Substance Designer	Art
Zbrush	Art
Draw.io & Lucidchart	Documentation, flow charts
Microsoft Teams	Group Communication
OneDrive & Google Drive	Documentation & Cloud Storage
FMOD & Audacity	Sound Editing
Soundsnap	Sound Sources
Itch	Deployment of Build
RenderDoc	Debugging

Third-Party Programming Libraries / Tools

GLFW & Glad	OpenGL API
Dear ImGui & ImGuizmo	Development UI
stb_image_write & stb_image	Saving and Loading Images
GLM	Mathematical Operations
Assimp	Model Loading
soloud	Audio
toml	Saving and loading

File Formats

File Type	Format
Audio	.wav
Model	.fbx / .obj
Animation	.dae
Material	.mat
Image	.png
Texture	.tga
Script	.cpp / .h
Code documentation	.pdf
Game level	.level
Vertex shader	.vert
Fragment shader	.frag
Shader meta info	.shader
Texture meta info	.texture
Material meta info	.material
Animation meta info	.animation
Model meta info	.model
Prefab	.prefab
Engine style info	.style
Camera system info	.cameraSystem
Health system info	.healthSystem
Enemy system info	.enemySystem
Engine preferences	.prefs
Imgui window placement	.ini

Source Control

Version Control System	Git
Source Control Repository Host	GitHub
Source Control Tools	GitHub Desktop & Source Tree & Fork
Source Control Remote Repository URL	GitHub Link

Setup

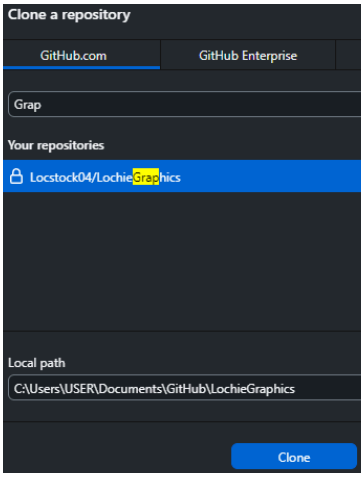
Downloading and setting up the project on a computer.

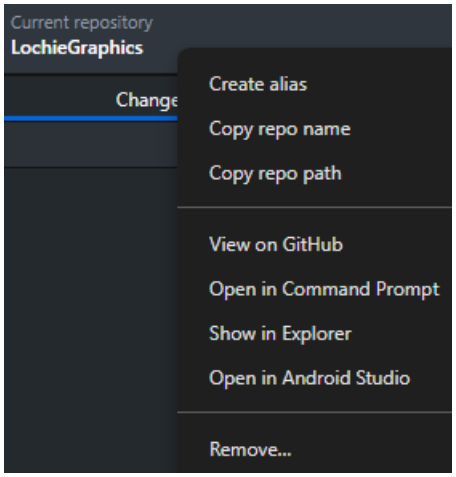
First Time

1. Ensure you have a GitHub account.
2. Contact Lochie to be added to the project.
3. Go to your GitHub Notifications, click on the notification invite from Lochie to be added to the project and accept the invitation.

Per computer

1. Ensure you have GitHub Desktop or another source control program of your choice.
    a. The following steps are specifically with GitHub Desktop in mind
2. Clone the repository LochieGraphics
3. Open the local location of the repository, this can be done by right-clicking the top left ‘Current Repository’ And clicking ‘Show in Explorer’
4. In the folder, there should be an ‘.exe’ that you can now run





Additional Repository Notes

Do not do anything related to the repository without first communicating with Lochie.

Input Methods

Input Methods and game controls can be found at GDD Input
For development within the engine, tools are available to test without physical controllers.

Engine Systems

The following should explain how to set up and use engine tools. For any more information, contact a programmer.

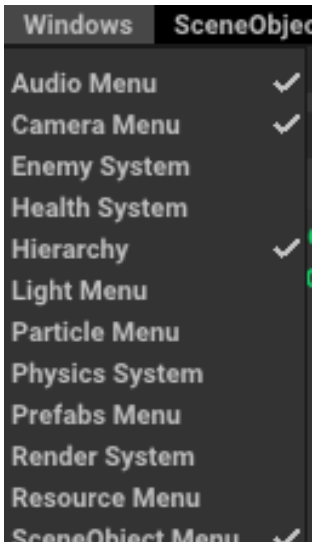
Interface

Menu Bar

Once the engine is launched, menu buttons are available at the top of the screen. They are:

- **File**, for saving and loading. The save button can be pressed with CTRL+S and loaded with CTRL + L.
- **Windows**, this is for opening and closing different windows. See Windows for more info.
- **SceneObject**, this contains a button to create a new scene object.
- **Open in Explorer**, pressing this opens the project directory in a new file explorer.

Windows



Windows can be opened through the menu bar button.

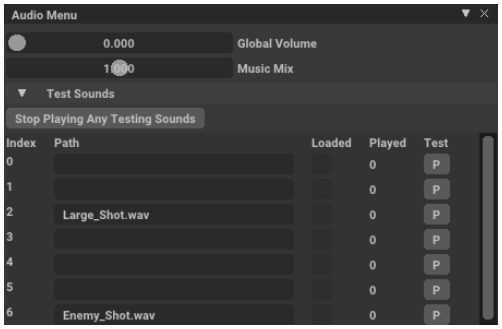
Each window contains information about different systems and means to interact with the engine.

Opening or closing the scene object menu or the hierarchy will do the same to the other if matched.

The window layout is stored as a file named “imgui.ini”

Audio Menu

This menu contains volume settings for any mixers, and information about audio such as the path, whether or not it has been loaded yet and how many times it has been played since the window was opened. This menu can also be used to test sounds. Although the paths can be changed live here, they do not save so this option is only for testing.



On the engine launch, a warning will appear for audio slots with no set path.

Camera Menu

The camera menu contains information about the current camera and details on the camera system settings.

FOV	The cameras FOV.
Near Plane	The near plane of the camera.
Far Plane	The far plane of the camera.
Mode	How the camera is controlled and moved around, there are separate modes designed for the level editor and the art tool, as well as ones specific to gameplay.
Orthographic Mode	To make the camera orthographic.
Orthographic Scale	The ‘size’ of the orthographic camera, this option is only visible if the camera is orthographic.
Camera System Filename	The name of the file for the camera system, and set camera system values can be stored and loaded from a file by using the “**Save**”, “**Load**”, or “**Save as**” buttons.
Minimum Camera Zoom	The closest the camera can be during gameplay.
Maximum Camera Zoom	The furthest the camera can be during gameplay.
Zoom Intensity While targeting	This value is used for the old camera system gameplay values and is now obsolete.
Zoom In Speed	How fast the camera can move in during gameplay.
Zoom Out Speed	How fast the camera can move out during gameplay.
Camera Move Speed	How fast the camera moves (across the XZ plane) during gameplay.

While targeting	
Camera Fov	The camera FOV during gameplay
Camera Zoom Closest Distance	The distance the players have to be for the camera to be the closest distance away.
Camera Zoom Furthest Distance	The distance the players have to be away from each other for the camera to be as far as it can be.

Enemy System

The enemy system can be saved and loaded with **Save**, **Load**, **Save as**. The enemy system has all the values for interacting with the enemies, such as how they move, the damage they do, and their health.

Health System

The health system menu is primarily for changing details about the healing ability.

Hierarchy

The hierarchy shows the transform hierarchy for all the scene objects within the level.

Clicking on an object will select that object. Multiple objects can be selected at once by using shift for a range of selections or control to toggle the selection of individual objects.

Light Menu

Options for the directional light such as the colour and direction. Individual light settings are contained on their respective parts.

Particle Menu

For debugging and previewing particles, this menu shows information about any currently live particles.

Physics System

The physics menu visualises the physics layers. It also has an option to display all of the colliders.

Prefabs Menu

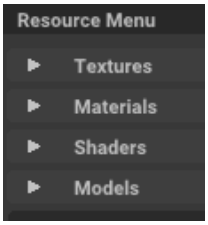
The prefabs menu has a selection of the selected prefab. It also has buttons to refresh the selected prefab instance, refresh all prefab instances within the level, and save any prefab origins.

Render System

The render system menu contains options for editing the SSAO, viewing a specific buffer index, adjusting the exposure and ambient light intensity, adjusting certain light values, turning super sampling on, and tone mapping. These values are for previewing and cannot be saved, if they wish to be changed let a programmer know.

More information about the render system specifically is in the Render System.

Resource Manager



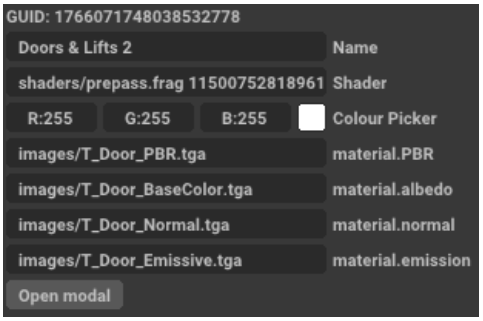
The resource manager menu contains a list of all loaded textures, materials, shaders and models. It also shows information on recent frame times

Textures show their path, type, whether or not it is flipped, its GUID, its GL ID, and a button to reload the texture.

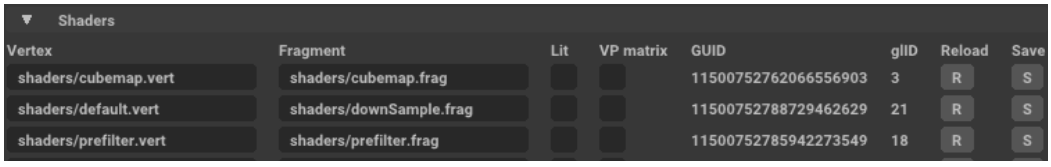
The texture path can be changed live, however this will not force the texture to be reloaded.
At the end of the list of the textures, there is a button available to force reload all of the textures (will not load any ‘generated’ textures; ones without a path)

Materials show their GUID, name, shader, colour, texture slots, and float values.

The texture and float slots are uniforms accessed from the shader, any shader variables that are prefixed with “material.” will be shown here. Pressing the open modal button will open a window to edit material values, this is the same window that is displayed when creating a material.



Shaders show their vertex shader path, fragment shader path, flags, GUID, GL ID, a button to reload, and a save button.

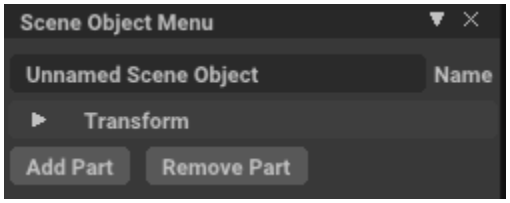


Pressing the save button will create a shader asset of that name, shader assets are not automatically loaded like other assets. There is also a button to create a new shader at the end of the list of shaders.

Models show a list of all loaded models with their GUID, path, mesh count, min and maxes, material IDs, model hierarchy info, and bone information.

At the bottom of the resource manager, there is a graph showcasing the amount of time the last 300 frames took, the average frame time across those, the average FPS, the latest frame time, the latest FPS and the minimum frame time which makes the engine pretend that the frame time can never be longer than this value.

SceneObject Menu



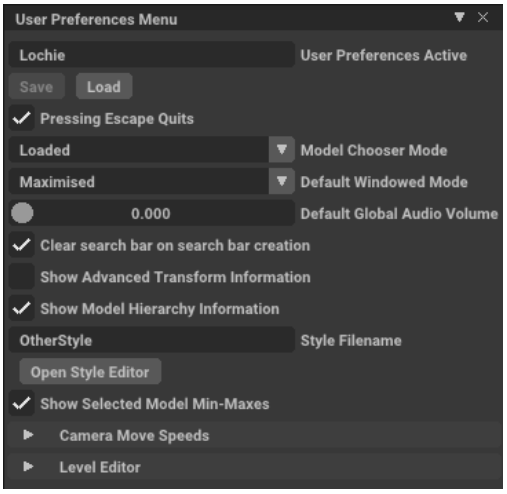
The Scene Object menu displays information for the currently selected object and the parts that belong to it.

All scene objects have a transform.

Style Editor Menu

For creating and editing styles for the editor.

User Prefs



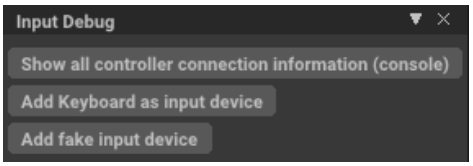
User preferences have a wide range of options for interacting with the engine. There should be user preferences specifically made for the build.

The last user preference used is remembered and automatically assigned on engine startup.

Imgui Demo

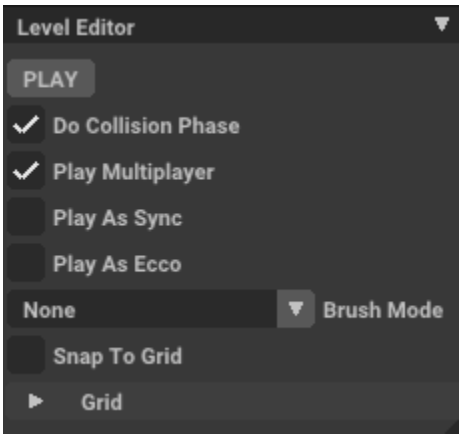
A demo window with examples of possible UI options, mainly there for the development of the engine and not intended for the user.

Input



The input menu contains the status of currently connected input devices and supports options for the ability to test without a physical input device, by adding the keyboard as an input device or adding a fake input device that is controlled using a GUI.

Level Editor



Contains the play button. Allows the user to switch controller (for the first input device) to specifically a single player for testing.

Brush mode has 5 different modes, being:

- **None**
- **Brush**, for placing ground tiles
- **Model Placer**, placing models
- **Prefab Placer**, placing prefabs
- **View Select**, selecting in a top-down view

Level Editor

Selecting an object will show a transform gizmo, ‘G’ toggles between translation and rotation, and ‘H’ toggles between local and global.
You can click an object to select it while in no brush mode, the selection is based on the bounding box for the model, or a metre cube if the object has no model.

Ecco

Although only one Ecco can function in a scene, multiple can be created, so it is important to only make one. Ecco can be found in the scene object hierarchy, their tweakable values include:

Screenshot from Ecco GUI

- **Car move speed:** Changes the acceleration of Ecco.
- **Car reverse speed:** Changes the reversing acceleration of Ecco.
- **Max Car move speed:** Changes the max speed of Ecco
- **Turning Circle Scalar:** Changes how tight the circle is, a smaller number is a larger circle.
- **Max Wheel Turning Angle:** Changes the maximum angle at which the wheels can point off the forward axis of Ecco.
- **Wheel Turn Speed:** Changes how fast the wheels turn to the new angle, a smaller number is a slower speed.
- **Sideways Wheel Drag**: Changes scale of drag induced by travelling in a different direction to the wheels' alignment.
- **Stopping Wheel Drag:** Changes scale of drag induced by not putting acceleration inputs in.
- **Local Steering:** Toggles whether the steering is based on the global position of the joystick or its local one,

Render System

The render system is written in C++ and GLSL using OpenGL. It currently supports the following features:

1. Ambient level texture
2. Albedo level texture
3. Skinned Mesh Rendering with blending
4. HDR
5. Bloom
6. SSAO
7. PBR
8. Point Lights
9. Spot Lights
10. Line lights
11. Beams (For sync shots)
12. Aim guide
13. Frustum culling
14. Particles
15. Spotlight culling & optimised shadowing
16. Shadow mapping for spotlights.
17. Contact shadows point lights
18. Optimised shader swapping.
19. Tone mapping
20. Supersampling
21. Deferred rendering
22. Static & dynamic pass
23. Decals
24. Shadow wall
25. Lights colour over time texture
26. Ecco face animations

The system takes Model Renderer Components and renders them in the scene, each being able to use distinct and non-distinct materials, meshes, and shaders.