UNIVERSITÉ Concordia UNIVERSITY

COMP 371

Project Report

SuperHyperCube

J. Damian Bowness (sec. CC - 21875841)

Richard Bui (sec. DD - 26632483)

Mingyang Chen (sec. DD - 40026335)

Thomas Gillard (sec. DD - 40031490)

Elijah Mon (sec. DD - 40078229)

August 21, 2021

Contents

1 Introduction

1.1 Objectives

The main objective of this project was to build a game using OpenGL, based on “SuperHyperCube” as seen in the project description handout. Our educational objectives were to further our OpenGL skills by learning how to build and import 3D models, implement sound and render text to the screen.

1.2 Game Description

SuperHyperCube is a straightforward game inspired by the popular Japanese game show activity ”Human Tetris”. The core gameplay loop consists of an number of ”rounds” that each concern a distinct object/wall pair. The objects and walls are models built from cubes such that a single orientation of the object may fit through a hole in the wall. These objects appear in a random order with varied orientations. Each round, players must utilize gameplay mechanics to manage an object’s orientation over a finite period of time and accumulate as many points as possible.

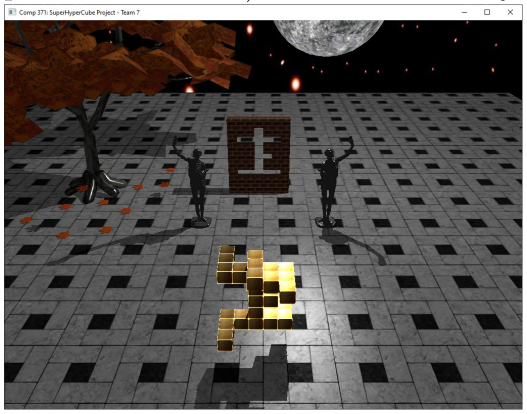
The main objective of the game is to rotate objects as they approach the wall, such that their orientations match, allowing the object the pass through the wall without colliding.

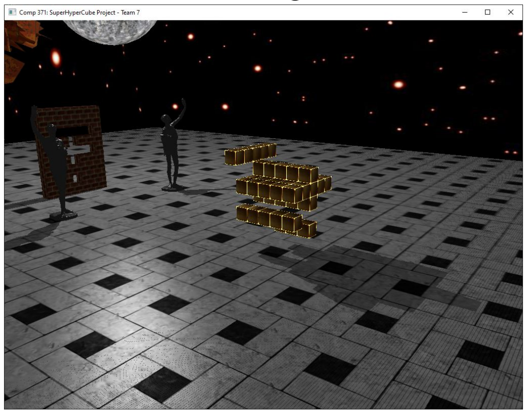
2 Methodology and Design

Due to time constraints and the limited number of puzzles that could be implemented, we decided to focus on the ambience surrounding the puzzles. To accomplish this, we decided to implement a "static" world unified around a single theme as a backdrop to the puzzles. Therefore, beyond building a game, we sought to create an aesthetically pleasing experience. The implementation of this game would be based on the culmination of all of our work from previous class assignments, as well as newer techniques that we learned in class.

2.1 Models

One of the main components of this game, were the game models that we created. Using the game model construction algorithms developed during the class assignments, we created a number of game models (one object/wall pair per team member) to be used in conjunction with the core game mechanics.





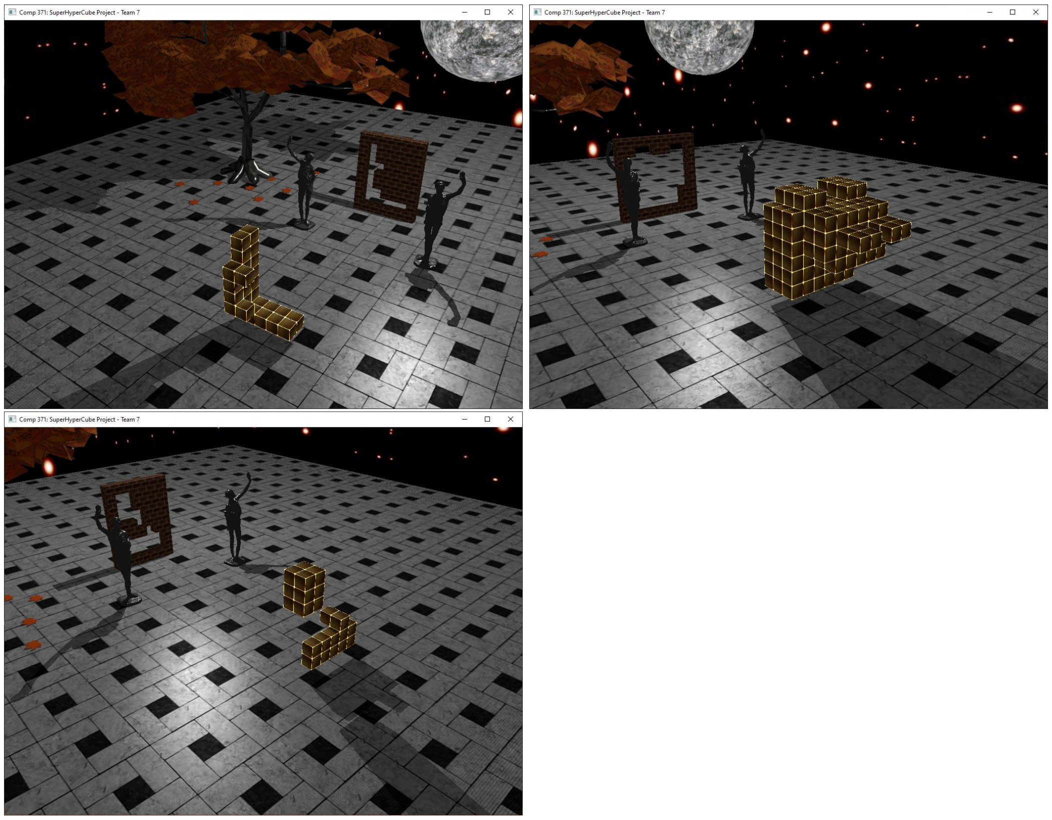

Comp 371: SuperHyperCube Project - Team 7
Comp 371: SuperHyperCube Project - Team 7
Comp 371: SuperHyperCube Project - Team 7

Figure 1: The 5 game models

2.2 User Interface

The game is set on a grey floor with a decoration tree also there are stars and the moon in the background. The actual game is formed by three parts: the golden cubes, two statues working as pass indicators and a wall behind them. The cubes and the wall are matched. If the cubes are able to pass through the wall, then the two statues will light up to indicator a successful passing.

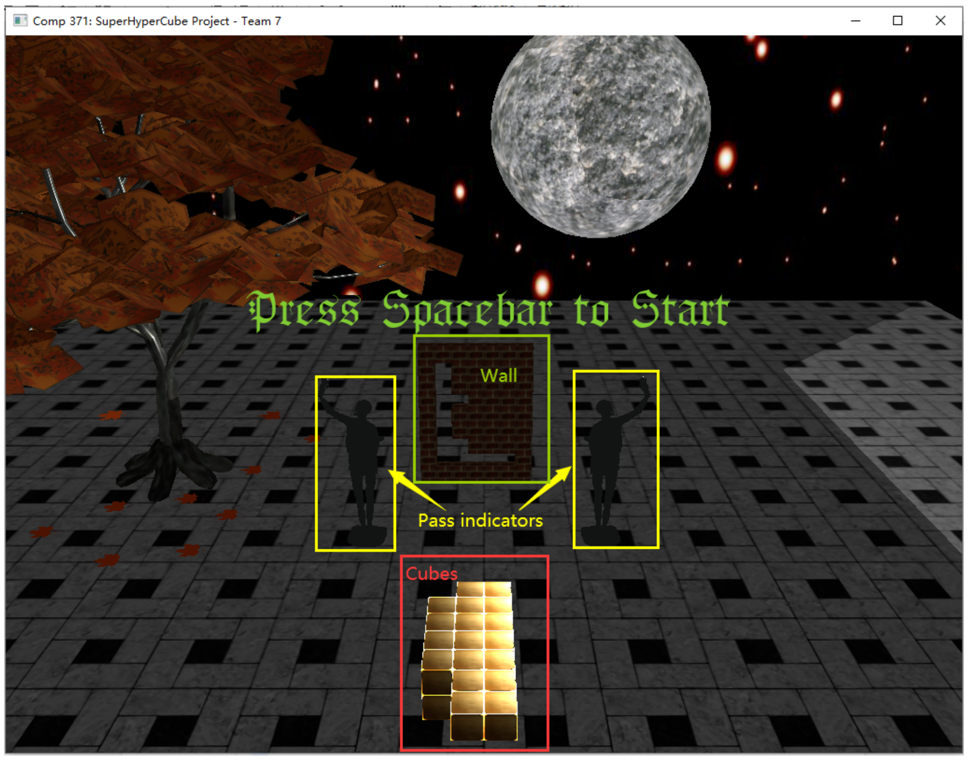


Figure 2: Basic Layout of the Game

Once the game starts, the player can see the score and timer at the top right corner as shown in Figure 2.

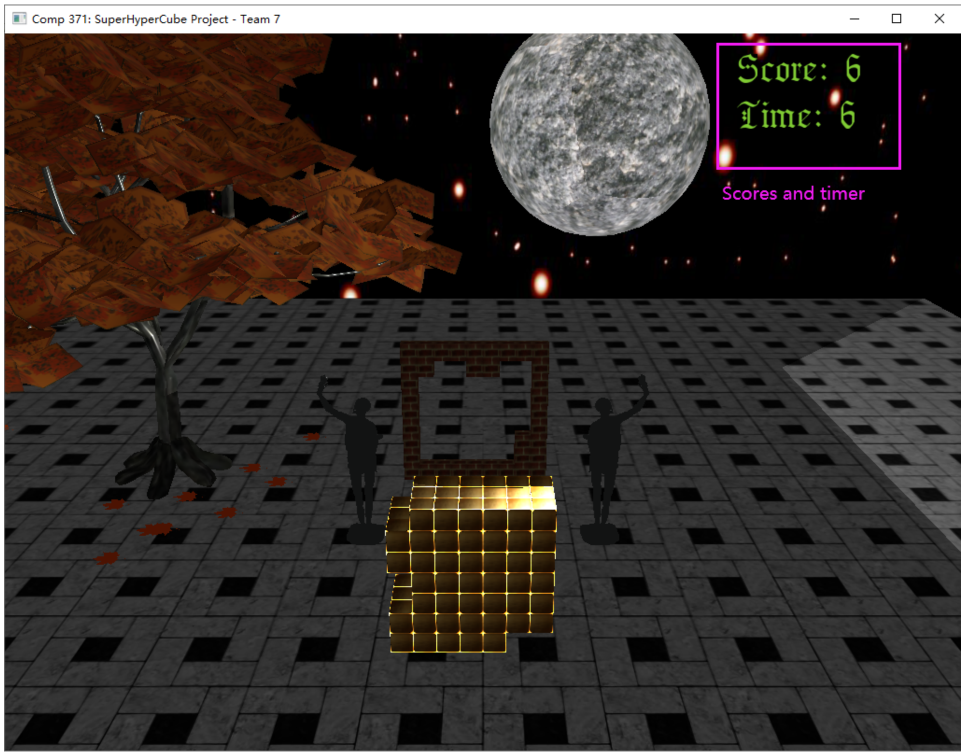


Figure 3: Scoreboard and Timer

If the player presses x, then the player can turn off the textures and see the whole game in a different style, the colour would change automatically.

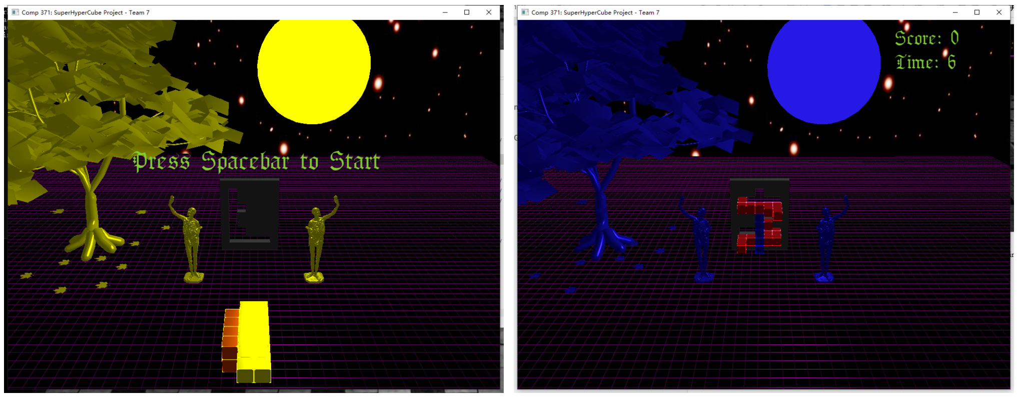

Comp 371: SuperHyperCube Project - Team 7
Press Spacebar to Start
Comp 371: SuperHyperCube Project - Team 7
Score: 0
Time: 6

Figure 4: Display of Different Themes

2.3 Gameplay

Phase 1: Game ready to start When the player enters the game, a “Press Spacebar to Start” would show on the middle of the screen indicating the game is ready to start.

Phase 2: Game starting Once the player presses the spacebar, the cubes on the bottom will start moving forward. And the player needs to press model manipulation keys to control the orientation of the cubes in order to make them able to pass through the wall. The player will have ten seconds to do so, or the player can press the enter key to speed up the moving process.

Phase 3: Score counting When the cubes reach the wall, the game would check if the cubes are able to pass through it:

1. If the passing is successful, a “success” sound would be played and the two statue indicators and the leaves of the tree would light up to inform the player this is a successful round. Then it would add 8 points to the score, and move on to the next round with new cubes and a new wall.

2. If the passing is unsuccessful, a “fail” sound would be played to inform the player this is an unsuccessful round. Then it would move on to the next round with new cubes and a new wall.

2.4 Design Decisions

The game allowed us to build on our work from previous assignments. This gave us the opportunity to further explore previous points of interest in OpenGL such as cube maps, implementing movement, and 3D modeling. In addition to building on our previous knowledge we were also interested in experimenting with new libraries such as Assimp, IrrKlang, and FreeType which allowed us to extend our capabilities with OpenGL. The extended capability of these libraries also motivated us to further our interests in learning third-party tools for editing the resources required by OpenGL. These tools include: Blender, GIMP, and Audacity.

The game consists of two domains of interest: the dynamic moving puzzles and the static surrounding world. The dynamic moving puzzles require concepts in physics and engineering such as motion and boundary collision. While the static world required us to think artistically about how thematic concepts are expressed through design. These two domains provided an opportunity to apply both analytical and creative thinking to the problem solving tasks.

2.4.1 Technical Design

One of the major technical challenges that we faced when implementing this puzzle game was collision detection: How will the game know when the object hits the wall? The object can rotate along any axis but can only translate along the z-axis. The object is also aware of all the orientations that will fit through the hole in the wall. Therefore a collision can only occur when the object is not in the correct orientation at the moment it crosses the plane of the wall. To simplify the solution, we designed each object to have only one correct orientation by eliminating symmetry along each axis.

2.4.2 Artistic Design

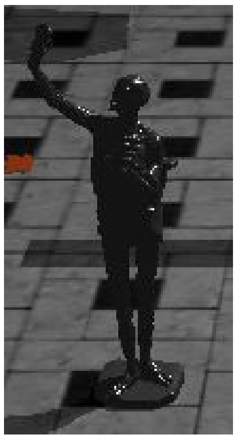


Figure 5. Ligier's *Le Transi de Rene de Chalon*

Thematically, the world is set around the slow march of the object towards the wall. The object's march is a visual reminder of the passing of time while the wall acts as a final destination. The objective of the game is to spin the object into the right position such that it will pass through the wall. Therefore, the central tenet of the game rests on a judgement. These underlying existential dynamics are captured in the scans of the plaster cast of Ligier Richier's funerary monument *Le Transi de Rene de Chalon* from the Musee des Monuments francais in Paris. This sculpture is of an emaciated skeleton with a few shreds of skin left, posed, looking upwards, to a heart held up in his hand with the other hand held to his chest. The skeleton in mid-decay is a static representation of the passage of time imbued by the game mechanics while the gaze upon the heart evokes the existentialism of Hamlet's "to be or not to be" binary. The tender pose of the skeleton's hand upon his chest suggests that skeleton is not just wrestling with existential questions but perhaps also pleading to a higher power. This sentiment compliments the binary nature of judgment the player faces at the wall. If the player succeeds, the object rises to join the heavenly bodies of the moon and stars. If they fail, the object falls to the void below the ground. The binary nature of this judgment is also symbolically reflected in the mirrored skeleton statues on either side of the wall.

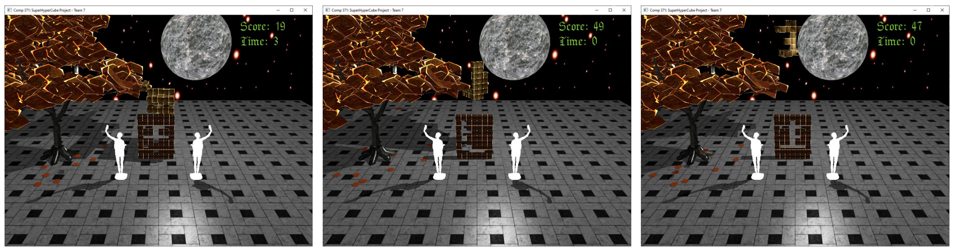


Figure 6: An object ascends upon a player's success

On starting the program, the object is static as the program waits for user input. However, despite the static scene, a leaf falls from an autumnal tree. It was originally decided to only incorporate a single falling leaf to reflect the solitude and isolation of the barren puzzle world but a single leaf was too subtle and it was worried that the TA might miss the animated visual effect during the demo so a second falling leaf was added.

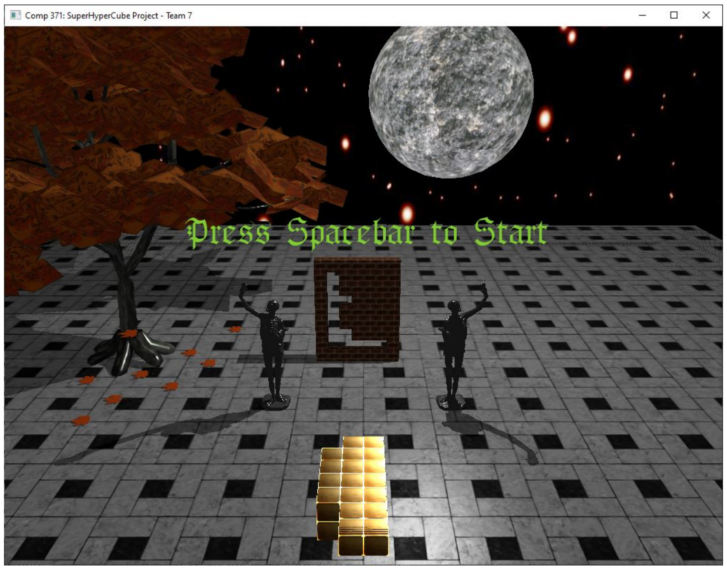


Figure 7: The scene that greets the player at the start of the game. Note the shadow of the skeleton statue on the right

Due to the interpolation of movement by the graphics card, each new round begins with the object moving into view. Therefore, the puzzle world is set at night because the player is perpetually witnessing the object at the end of its journey. In order to produce shadows, a source of illumination was added in the form of a point light housed inside the moon. The moon as pointlight is contradictory to the real world in which it acts as a directional light. However, the moon as directional light would cause the skeleton statue on the right to cast a shadow directly across the path of the object. We decided it is best to keep the path of the object as free of visual clutter as possible in order to give the player a clear view of the game they're playing.

3 Implementation

3.1 Class Overview

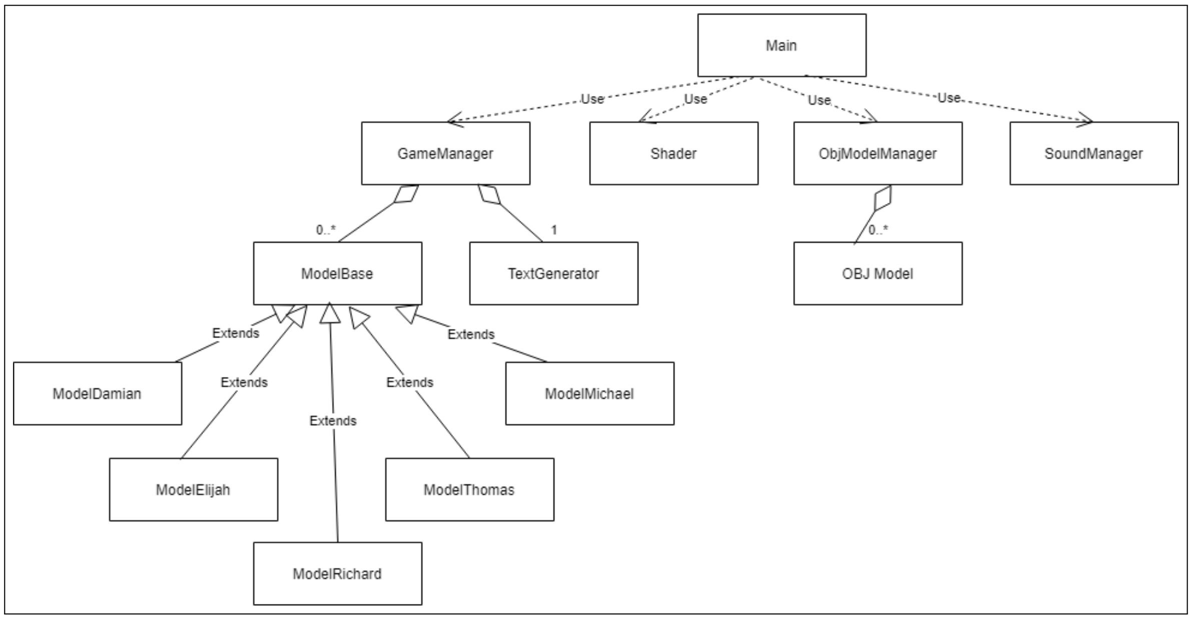


Figure 8: UML Diagram of the Game

ModelBase:	Renders game models
	Tracks game models position and orientation
GameManager:	Tracks time, score, and levels (game models)
	Sends time and score to TextGenerator
	Sends sound effect requests to sound manager
TextGenerator:	Loads and renders imported 3D model from OBJ files
ObjModelManager:	Loads and plays in-game music and sound effects

SoundManager:	Loads and plays in-game music and sound effects
TextGenerator:	Loads font and renders text to screen
Shader:	Loads textures and sets uniforms
Model(name):	Derived from ModelBase
	Defines shape of game models

3.2 Technical Implementation

A main function drives the program. It instantiates and initializes the soundmanager, the objModelManager (imported model manager), and the game manager. The objects, wall, floor, and "world cube" are all instances of a unit cube class on the CPU. However, on the GPU, they all share the same VAO and VBO because they all use the same set of vertices that define the unit cube.

The different object and wall pairs defines the levels of the game. The object and wall are members of the ModelBase class. The classes derived from the ModelBase class specify the shape of the object and its corresponding hole in the wall for the different game models (not to be confused with the imported OBJ models). This allows us to design a variety of game models. All other behaviours of the derived classes are handled by the ModelBase class. As such, ModelBase class knows the position of the game model's base position as well as its object's x, y, z translations.

The game consists of three states: the start state, the moving state, and the end state. The end state itself consists of two substates: the success state and the fail state. These states are managed by the Game Manager. The GameManager queries ModelBase to determine the state of the game models which in turn determines whether the game is in a moving state or an end state. If the game is in an end state the game manager queries ModelBase to determine whether the object is in the appropriate orientation for a success state or a fail state. Depending on the state, GameManager updates the score, tells TextGenerator to update the display, and tells SoundManager to play the appropriate effect.

The GameManager instantiates a TextGenerator but receives a reference to the SoundManager from the main function. This is because the main function invokes SoundManager methods in response key callbacks. Ideally, the SoundManager would be a member of GameManager and require main to access SoundManager through GameManager. This would consolidate all game play effects in GameManger.

The scene consists of game models and imported OBJ models. The GameManger instantiates the game models and renders them while the OBJ models are loaded and rendered by the ObjModelManager.

4 Discussion

Implementing the game as our final project provided many unforeseen challenges. These challenges can be classified into a variety of different categories, mainly technical, organizational and creative.

In terms of technical challenges, boundary collision was a more difficult than we initially thought and it required us to think creatively about how to simplify the problem so that we could submit the project by the deadline. In addition to this, point lights made it difficult to collaborate because we were unaware that geometry shaders are not available on all GPUs. As such, only two of the five team members could see the shadows cast by the point lights (the attentive reader may have already noticed the discrepancy in shadows in the images used in this report). In the future, we will limit the use of point lights and focus more on the implementation of a directional light.

A major organizational hurdle was time zones, having team members spread across the globe made it difficult to coordinate team meetings. However, thanks to modern messaging technology we were able to maintain a rolling conversation about the project on Discord. Ultimately this project proved to be a rewarding experience and we look forward to continuing to tweak and improve this game experience privately in our free time in the future.

References

[1] Angel, E., & Shreiner, D. (2012). Chapter 8: Modelling and Heirarchy. In Interactive computer graphics: A top-down approach with shader-based opengl (6th ed., pp. 425–432). essay, Addison-Wesley.

[2] Bailey, M. (n.d.). A Whirlwind Introduction to Computer Graphics. SIGGRAPH 2020.

[3] Game Technology Group. (n.d.). Https://research.ncl.ac.uk/game/mastersdegree/graphicsforgames/scenegraphs/. Graphics for Games. Newcastle upon Tyne; Newcastle University.

[4] Game Technology Group. (n.d.). Https://research.ncl.ac.uk/game/mastersdegree/graphicsforgames/scenemanagement/. Graphics for Games. Newcastle upon Tyne; Newcastle University.

[5] Kessenich, J. M., Sellers, G., & Shreiner, D. (2017). OpenGL® programming guide: The official guide to learning Opengl®, version 4.5 With Spir-V (9th ed.). Addison-Wesley.

[6] Lengyel, E. (2016). In Foundations of game engine development: Mathematics (Vol. 1, pp. 1–74). essay, Terathon Software LLC.

[7] Lengyel, E. (2019). In Foundations of game engine development: Rendering (Vol. 2, pp. 1–194). essay, Terathon Software LLC. nlguillemot. (2016, November 16).

https://nlguillemot.wordpress.com/2016/11/18/opengl-renderer-design/. https://nlguillemot.wordpress.com/2016/11/18/opengl-renderer-design/.

[8] Parent, R. (2012). In Computer animation: Algorithms and techniques (3rd ed., pp. 219–232). essay, Elsevier, Morgan Kaufmann is an imprint of Elsevier.

[9] Sellers, G., Haemel, N., & Wright, R. S. (2016). Opengl superbible: Comprehensive tutorial and reference. Addison-Wesley.

[10] Stroustrup, B. (2020). Chapter 1: The Basics. In Tour of c++ (2nd ed., Ser. C++ In-Depth Series, pp. 11–14). story, ADDISON-WESLEY.

[11] Vries, J. de. (2020). Learn opengl: Learn modern opengl graphics programming in a step-by-step fashion. Kendall & Welling.

[12] YouTube. (2014). #5 Intro to Modern OpenGL Tutorial: 3D Motion. YouTube. https://www.youtube.com/watch?v=Xe7FmplKAF0&t=238s&ab_channel=thebennybox.

[13] YouTube. (2014). Update: "Proper" Obj Loading and Modern OpenGL. YouTube. https://www.youtube.com/watch?v=PTTN9i5IUbA&t=188s&ab_channel=thebennybox.

[14] YouTube. (2017). OpenGL. YouTube. https://www.youtube.com/playlist?list=PLlrATfBNZ98foTJPJ_Ev03o2oq3-GGOS2&index=5.

[15] YouTube. (2018). WebGL2 : 081 : Tron Line Shader. YouTube. https://www.youtube.com/watch?v=DI498yX-6XM&ab_channel=SketchpunkLabs.

[16] YouTube. (2019). H t t p s : / / w w w . y o u t u b e . c o m / w a t c h ? v = Q Y v i 1 a k O _ P o & a b _ c h a n n e l = B r i a n W i l l. YouTube. https://www.youtube.com/watch?v=QYvi1akO_Po&ab_channel=BrianWill.

[17] YouTube. (2020). Blender 2.90 - How to Make Exact Leaf Shape (Modelling + Texturing) Tutorial —— Blender+Gimp ——. YouTube. http ://www.youtube.com/watch?v=GjJfpLFI9GY&ab_channel=Welcomet ocreativeworld