Computer Graphics Coursework 2

Dillan Kerai : sc21dmk : 201492302
Aaditya Kiran Gowda : sc21akg : 201580219
Adam Zureiki : sc21az : 201525746

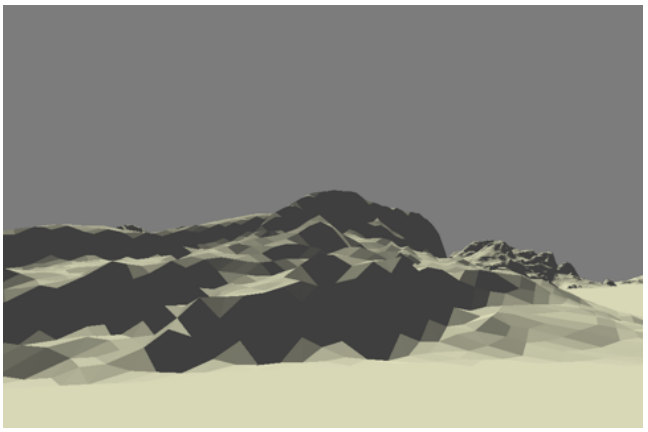
1.1 Matrix/Vector Functions

The following test cases are intended to verify a wide range of mathematical operations concerning vectors and 4x4 matrices. Each test case has the following brief explanation:

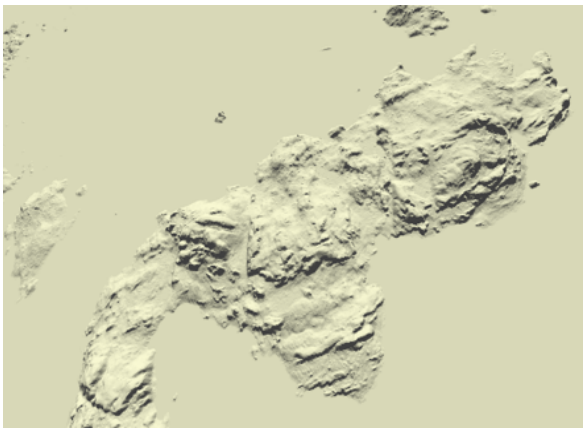
Test 1: Matrix Multiplication (Mat44f * Mat44f)
This test case was designed to check the correctness of multiplication of two matrices. Since accurate Matrix Multiplication is essential for transformation of objects in 3D space. It ensures that the transformation of combining two matrices together is done correctly. The reason behind doing this test is to ensure these calculations are applied accurately to the objects in the world.
Test 2: Matrix Multiplication with an Identity Matrix (Matt 44f * Mat44f Identity)
This test case was designed to check the correctness of multiplication of a matrix with an Identity Matrix. It's a rudimentary test that shows that the basis functions of a matrix multiplication are applied correctly. Doing this test will indicate any issues in the matrix operations, which may cause further issues during the use of the scene.
Test 3: **Symmetric Test**
This test verifies that the matrix multiplication is not symmetric. Primarily ensuring that the matrices being used are non-commutative. It will help ensure that any further matrix implementations are in the correct order, preventing any incorrect sequence of transformation.
Test 4: Matrix-Vector Multiplication (Mat44f * Vec4f)
This test case includes two sub-tests: Multiplication of an identity matrix with a 4d Vector and Multiplication of a non-identity matrix with a 4d vector. It helps ensure that the matrices are transformed correctly. This test was developed to help reduce any incorrect scaling in terms of the spaceship and the project camera world.
Test 5: Rotation Matrix
This test case verifies the correct calculations of the rotation matrices for four different angles namely: 0, 90, -90 and 180 around X, Y and Z angles using WithinAbs to bound the values to a certain level of degree.
Test 6: Translation Matrix Test
It verifies the creation of a translation matrix from 3D Vector. It also checks whether the translation matrix represents the translation vector accurately. With an accurate translation vector, it will ensure the positioning of the objects are correct in the scene, and they calculate the intended movement of moving around the scene
Test 7: Perspective Projection Test
This test validates the creation of the perspective projection matrix. It also includes an edge case with extreme values and a normal case with average values. This test is vital in mapping 3D coordinates to 2D screen coordinates because it will produce unrealistic objects and inaccurate colors.

1.2 3D Renderer Basics

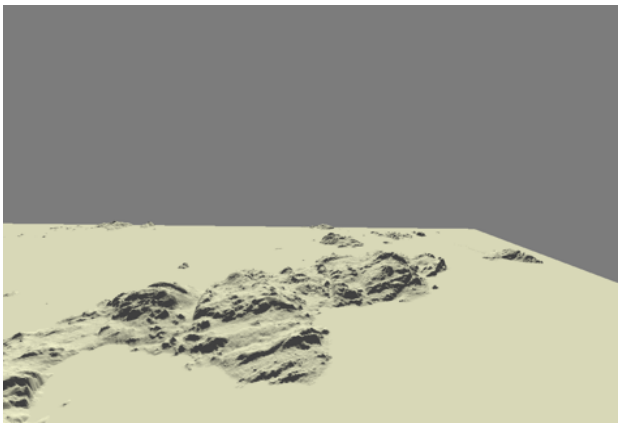
GL_RENDERER	NVIDIA GeForce RTX 4070/PCIe/SSE2
GL_VENDOR	NVIDIA Corporation
GL_VERSION	4.3.0 NVIDIA 545.23.08
SHADING_LANGUAGE_VERSION	4.30 NVIDIA via Cg compiler



1.2.1. Zoomed-in view of a mountain

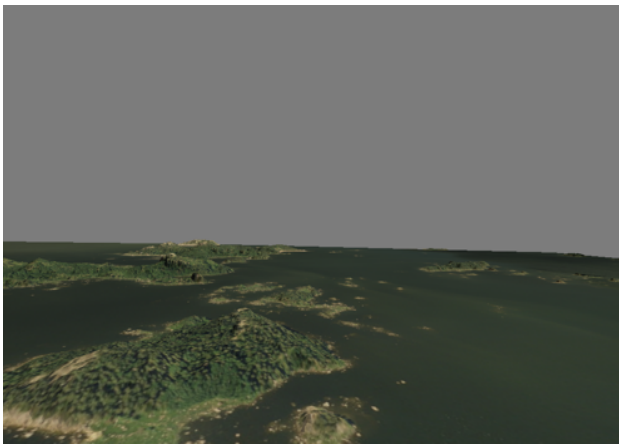


1.2.2. Top-Down view of the Landscape

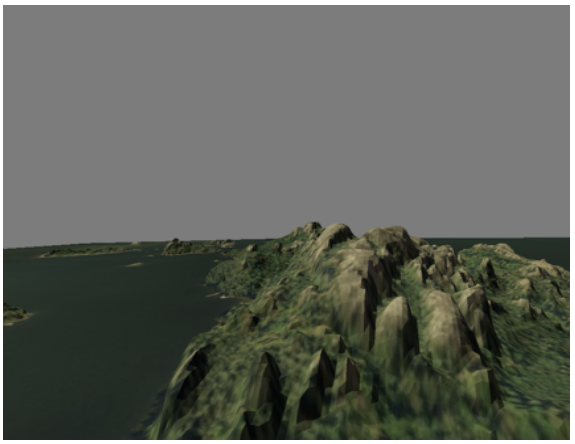


1.2.3 Wide angle view of the Landscape ground level.

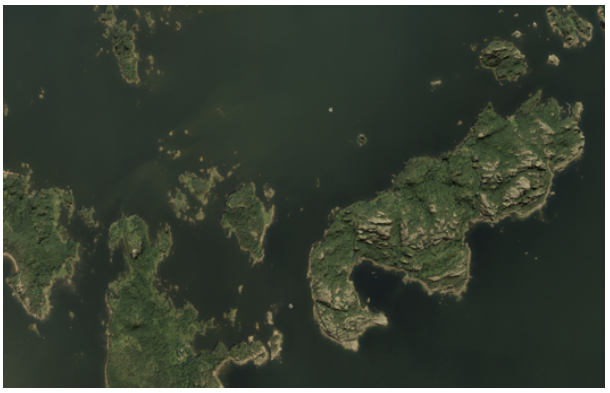
1.3 Texturing



1.3.1. Wide angle view of the Terrain



1.3.3 Elevated view of a mountain

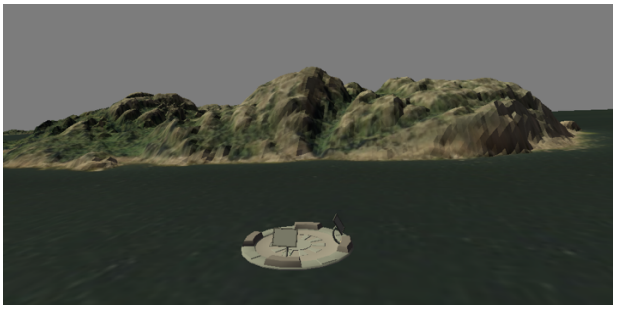


1.3.2. Top – down view of the Terrain

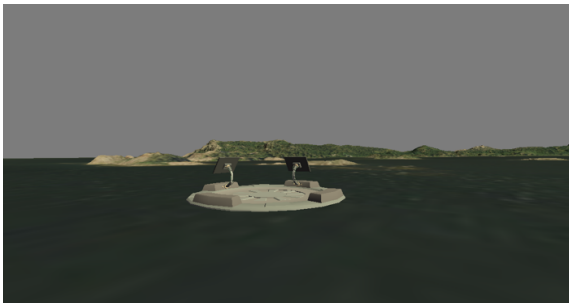
1.4 Simple Instancing

The coordinates for pad one is (0.0, -0.97300, 20.0f) where the -0.97300 is the y-coordinate plane of the map

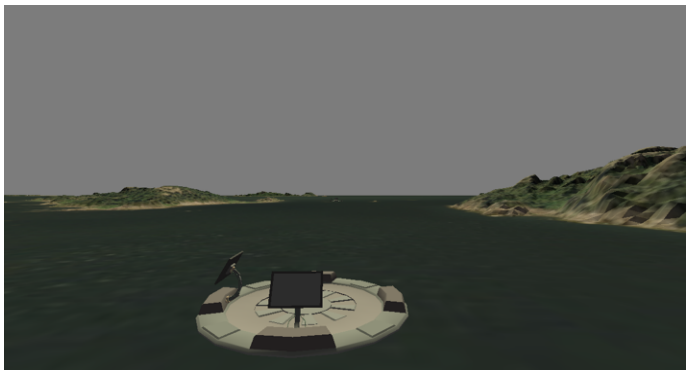
The second pad coordinate is (10.0, - 0.97300, -2.5f) where the -0.97300 is the y-coordinate plane of the map



1.4.1. First view of the launchpad from a high angle



1.4.2. Second view from behind the screens

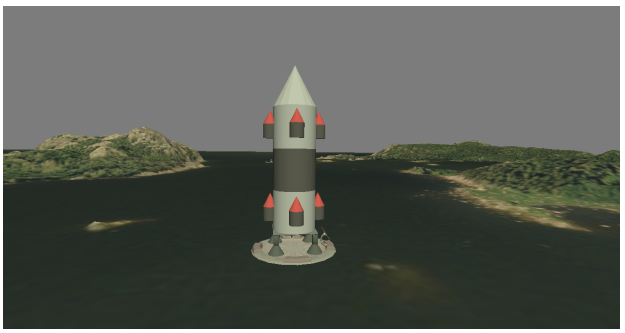


1.4.3. Third view close up

1.5 Custom Model

The space vehicle was placed on the second launch pad.

1.5.1 Space Vehicle view 1

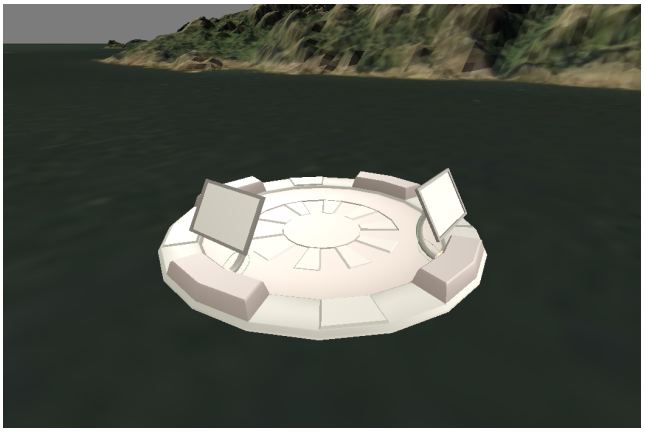


1.5.2 Space Vehicle from view 2

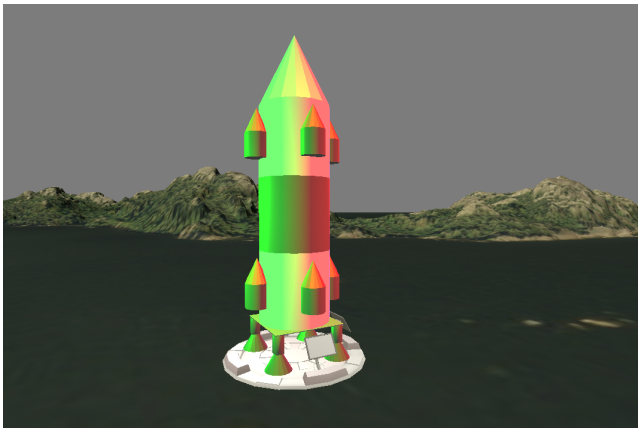
1.5.3 Space Vehicle from top view

1.5.4 Space Vehicle from view 4

1.6 Local light sources

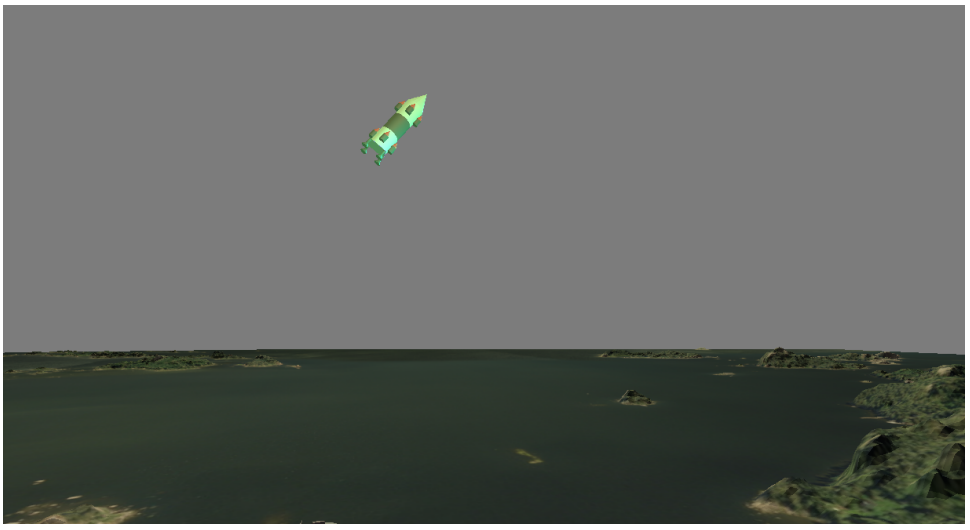


1.6.1. First view of lit up launchpad



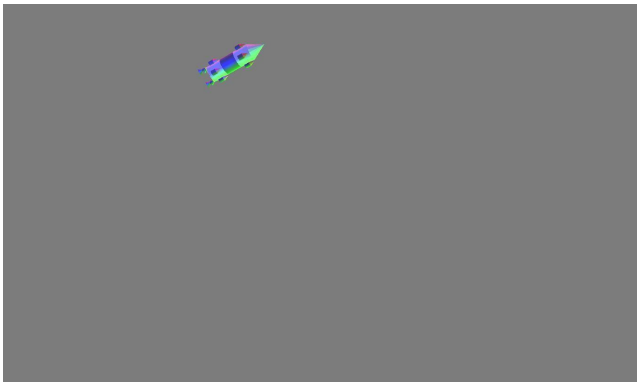
1.6.2 View of lit up space vehicle

1.7 Animation

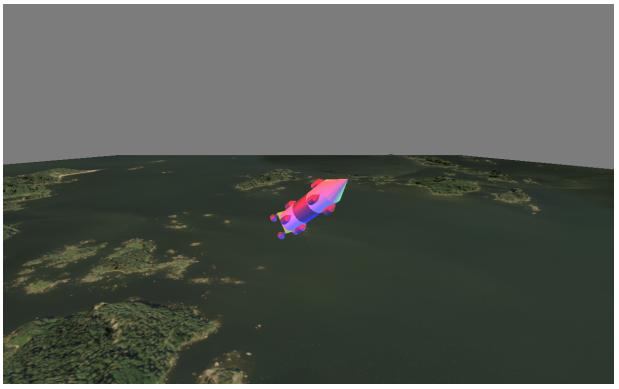


1.7 View of the spaceship at animation curve

1.8 Tracking Cameras

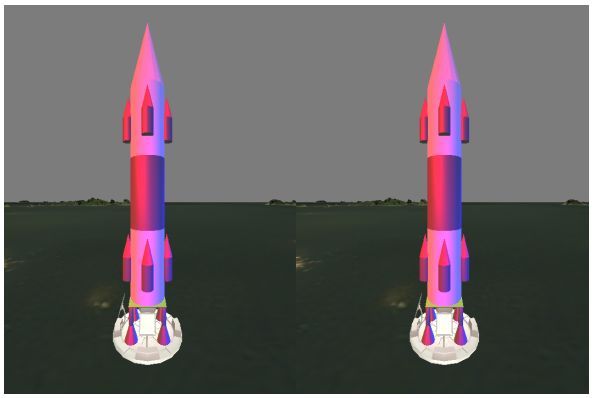


1.8.1 Space Vehicle from the ground camera

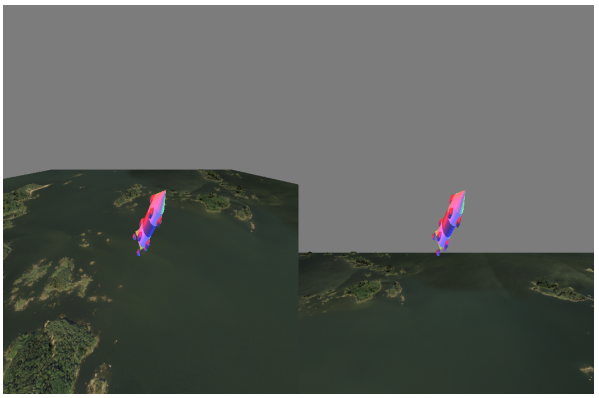


1.8.2 Space Vehicle from fixed camera

1.9 Split Screen



1.9.1 Split screen from the initial camera position



1.9.2 Split screen from fixed camera mode

Split screen was implemented by creating two different view ports and using the initial code to create a replica of the main scene into two different windows. This was done to visualize a split screen effect. Shift C key command was added to control the right side of the screen. V key command was added to enable split screen mode. Two different projcameraworlds were added to control either side of the split screen. By implementing this, we achieved a visually engaging split screen functionality.

1.12 Measuring performance

Rendering Performance Type	Section (Without Movement)	Time (ms)
GPU	Full Scene	3.59117
GPU	Task 1.2 3D Renderer Basics	3.55942

GPU	Tasks 1.4 Simple Instancing	0.016384
GPU	Tasks 1.5 Custom Model	0.01024

Rendering Performance Type	Section (Without Movement)	Time to submit rendering commands (ms)	Frame-to-Frame
CPU	Full Scene	18.7224	0.0185371
CPU	Task 1.2 3D Renderer Basics	12.4919	0.0123682
CPU	Task 1.4 Simple Instancing	3.70947	0.00367224
CPU	Task 1.5 Custom Model	2.38927	0.00236561

To enable less duplicate lines of code in performance benchmarking, we reuse the existing tests but instead, we move around the scene to retrieve the movement benchmarking values. This helps save time, and simplifies the code that is being used.

Rendering Performance Type	Section (With Movement)	Time (ms)
GPU	Full Scene	3.46419
GPU	Task 1.2 3D Renderer Basics	3.4304
GPU	Tasks 1.4 Simple Instancing	0.019456
GPU	Tasks 1.5 Custom Model	0.01024

Rendering Performance Type	Section (With Movement)	Time to submit rendering commands (ms)	Frame-to-Frame
CPU	Full Scene	16.0933	0.015934

CPU	Task 1.2 3D Renderer Basics	10.3238	0.0102216
CPU	Task 1.4 Simple Instancing	3.43025	0.00339629
CPU	Task 1.5 Custom Model	2.09671	0.00207595

Our implementation uses both the CPU and GPU testing methods to benchmark results and show the comparisons. For benchmarking the CPU , we used OpenGL chrono library to be able to detect the time it takes to run each frame and calculate the times it takes to submit all of the rendering commands.

In terms of GPU performance, the data is much lower than the CPU across all of the tasks. This was expected as GPUs are mainly designed to handle graphical computations therefore having a robust machine to run it on (machine containing NVIDIA GeForce RTX 4070), which allows for comfortable rendering of 3d objects. For the full scene, the rendering time has decreased from 3.59117ms (without movement) to 3.46419ms implying that the used graphics card is efficient in handling dynamic change to the scene. Also from task 1.2, the time decreases from 3.55942ms to 3.4303ms, with a better running time during the movement element of the scene , indicating that the graphics card is capable of handling dynamic changes to the scene. This was not accepted as you would accept the time to be higher due to changes that are being made to the scene within each frame, e.g moving around the scene. With simple rendering such as the pads and the space vehicle, the GPU benchmarking results are highly effective. Values such as 0.016384ms and 0.01024ms shows the incredibly fast performance the GPU has on this.

When we look at the CPU performance, the overall rendering times are slightly high which is expected as the CPU is not made to handle 3D graphics and movement. It's more of a general-purpose however the results obtained may contradict the reasoning. For the full scene, the time drops from 18.7224ms to 16.0933ms. In this case, the CPU is able to handle the dynamic movement, the GPU is a better option.

Given the high-end specifications of the graphics card being used (NVIDIA GeForce RTX 4070), the L2 cache is much larger than the previous versions which allows more temporary data which reduces the need to access data from the main memory. In this case when we compile the code for the first time, it stores some of the files in the L2 cache therefore when re-compiling the code, the program does not need to access all of the data from the main memory, allowing for more optimized CPU and GPU rendering tasks.

The benchmarking code has been commented out to allow further use of the scene.

2. Appendix

Tasks	Member	Task Contribution
1.1 Matrix Vector Function	Dillan Kerai	Implemented the Mat44f and Vec4f file. Created test case for the functions and reported the reasoning
1.2 3D Renderer Basics	Aaditya Kiran Adam Zueriki Dillan Kerai	Adam implemented the speed (shift and control), Q,E and WASD controls. With contribution from Dillan development camera matrix. We all developed the textured map through group-programming using similar code from the exercises

1.3 Texturing	Adam Zueriki Dillan Kerai	Adam created the template for the texturing by modifying the frag and shader files. He also created a function which processes the textures. Dillan edited the main file to bind the texture from the object file and be able to map the textures to the Parlahti object.
1.4 Simple Instancing	Dillan Kerai	Implemented the pads and created a location for them. Screenshotted the position of the pads. Handled it by having different shaders program to handle the landing pads
1.5 Custom Model	Adam Zureiki Dillan Kerai	Dillan created the base of the space vehicle which includes 1 cuboid, 4 cubes and 4 cones. From there, Adam was able to create the space vehicle body consisting of 11 cylinders and 9 cones. We concatenated the main body onto the base to merge it into one object.
1.6 Local Light Sources	Dillan Kerai	Dillan used the blinn-phong method to implement the three point lights on the pad and space vehicle. This was added on main, in addition to the frag vert files.
1.7 Animation	Aaditya Kiran Adam Zureiki Dillan Kerai	Firstly, Aaditya created animations of the spaceship moving in the y-direction. Secondly Adam started creating an animation based on the bezier curve. He activated this using the F and R keys. Furthermore, Dillan fixed the animation to enable the space vehicle to move in a calculated curved path with the top of the space vehicle facing the correct direction of movement.
1.8 Tracking Camera	Aaditya Kiran Adam Zureiki	Aaditya created two different camera modes to the default. One that travels with the space vehicle as the animation happens. The other camera mode stayed on the ground and watched the space vehicle as it is animated. For both new camera modes, the user was not able to move freely. Adam then fixed some issues in both cameras, the first issue for the first mode had to do with the camera angle getting messed up when the space bar was clicked. The second issue fixed was to do with the ground camera. Initially the camera was not looking at the space vehicle, that was fixed by correcting the angle calculations.
1.9 Split Screen	Dillan Kerai	Dillan created a split screen view that vertically split the windows into two different screens. He did this by creating two different view ports and using the initial code to create a replica of the main scene into two different windows.
1.10 Particles	Not done	Not done

1.11 Simple 2D Elements	Not done	Not done
1.12 Measuring Performances	Dillan Kerai	Dillan created benchmarking tests in the code to calculate the CPU and GPU values to be used for further evaluation

2.1 References

- LearnOpenGL. Camera. Accessed from https://learnopengl.com/Getting-started/Camera [Online]
- Davies, J. (n.d.). Animated Bézier. Accessed from https://www.jasondavies.com/animated-bezier/ [Online]
- Wikipedia contributors. (n.d.). *Bézier curve*. Accessed from https://en.wikipedia.org/wiki/B%C3%A9zier_curve [Online]