# Real-Time Rendering and 3D Games Programming

# ASSIGNMENT 1 – REPORT (v1.1)

## Ryan Hefford s3721994

## INTRODUCTION

Which shape did you choose to draw? Did you derive the algorithm on your own or did you find some other resource to help? List any sources used (books, articles, videos, ...).

I chose to draw the Menger Sponge, the algorithm I used was heavily inspired from the YouTube channel The Coding Train, named “Coding Challenge #2: Menger Sponge Fractal” as referenced bellow.

<https://www.youtube.com/watch?v=LG8ZK-rRkXo>

Describe the hardware you used to perform the tests described in this report. Include detailed CPU and GPU information. What screen resolution and refresh rate did you use?

I ran these tests using an Intel Core i7-9700K of which has 8 cores that runs at 3.60 GHz processor frequency.

<https://ark.intel.com/content/www/us/en/ark/products/186604/intel-core-i7-9700k-processor-12m-cache-up-to-4-90-ghz.html>

As for the GPU I am using a MSI GeForce RTX 2060 Super Ventus GP OC 8GB OEM that has 1470 MHz base clock speed.

<https://www.techpowerup.com/gpu-specs/msi-rtx-2060-super-ventus-gp-oc.b7381>

I also have a 1920 x 1080 screen resolution with 60 Hz refresh rate.

Describe your data structure and algorithm. Are you duplicating vertices that are used by multiple triangles or did you implement shared vertices? Are these cases where multiple faces might overlap? Which OpenGL drawing primitive are you using?

The data structure I went with was 2 float arrays, one for the vertex points and the other for the face indices. As for the algorithm, I utilised recursion with the method:

DrawCube (float x, float y, float z, float size, int recursionsLeft) in which the base case is recursionsLeft == 1. And for each call where recursionsLeft > 1, 20 more recursive calls (after excluding the centre cubes) would be made with the new size being a third and recursionsLeft one less.

If the base case is reached a cube is drawn with the position and size parameters.

Unfortunately, as this algorithm adds to the arrays on a cube by cube basis, adjacent cubes will share vertices but the indices their faces reference will be added at separate times, therefore creating duplicated vertices. This will also cause some cube faces to overlap if the cubes are connected.

Only Triangle primitives were used for this sponge.

How did you choose to colour the shape? How many materials did you use and how were they assignment to faces? How did you 'communicate' face material data to the Shaders?

I chose to implement 3 materials: blue, red, and white. The colours were assigned based on the primitive’s ID’s modular value, as the cubes faces were always created in the same order, there was a pattern to the material assignment order. The materials were also communicated to the shaders using a material struct uniform. This struct is defined with ambient, diffuse, specular, and shininess values and is defined within the Lights.cpp file.

How have you decided to position each light source? How did you assign light colours to show off the full capabilities of your lighting model?

For the spotlight positions, I went with one in the centre of the scene to emphasize the specular lighting from all angles. The following spot lights were placed at a position, 1 unit away from the centre in the x,y and/or z axis, so that each position was unique. I did this such that no one light would be blocking another, therefore each light would have its own impact to the sponge. To choose the colours of the lights I had to test the scene a lot, but eventually found values for each light such that their impact could be seen with each addition.

## SCENE 1

Start your testing at subdivision level 1 (base), Lighting On (1 light), Backface Culling On and Depth Testing On.

Create a table showing the average frame rate, number of vertices and number of faces at each level of subdivision that your hardware can handle with a frame rate greater than 1 frame per second.

|  |  |  |  |
| --- | --- | --- | --- |
| Subdivisions | Frames Per Second | Vertices | Faces |
| 1 | ~4000 | 8 | 12 |
| 2 | ~3600 | 160 | 240 |
| 3 | ~1100 | 3200 | 4800 |
| 4 | ~77 | 64000 | 96000 |
| 5 | ~4 | 1280000 | 1920000 |

Draw a chart showing the average frame rate achieved at each level of subdivision.

Run some tests with Lighting Off while keeping everything else as above. Describe the impact this has on frame rate and why? Use a table and a chart to show the data.

|  |  |  |
| --- | --- | --- |
| Subdivisions | FPS with Lighting | FPS without Lighting |
| 1 | ~4000 | ~3950 |
| 2 | ~3600 | ~3550 |
| 3 | ~1100 | ~1075 |
| 4 | ~77 | ~75 |
| 5 | ~4 | ~3.85 |

As the graphs show above, the difference in framerate caused by the lighting seems to be worsening slightly when the lighting is disabled. This may be due to the use of glMaterial() being used for non-lighting instead of glColors(), this may be a frame drop due to the handling for no specular component.

Run some tests with Backface Culling On and Off, while keeping everything else as above. Describe the impact this feature has on frame rate and why? Use a table and a chart to show the data.

|  |  |  |
| --- | --- | --- |
| Subdivisions | FPS with Back Culling | FPS without Back Culling |
| 1 | ~4000 | ~4000 |
| 2 | ~3600 | ~3550 |
| 3 | ~1100 | ~1050 |
| 4 | ~77 | ~76 |
| 5 | ~4 | ~3.9 |

The results show, although slight, the disabling of back face culling causes the framerate to consistently drop. This is likely due to the program still needing to draw faces that will never be visible to the camera unless wireframe is enabled. Meaning the back face culling is far more optimal for the sponge.

Run some tests with Depth Testing On and Off, while keeping everything else as above. Describe the impact this feature has on frame rate and why? Use a table and a chart to show the data.

|  |  |  |
| --- | --- | --- |
| Subdivisions | FPS with Depth Testing | FPS without Depth Testing |
| 1 | ~4000 | ~3900 |
| 2 | ~3600 | ~3550 |
| 3 | ~1100 | ~1100 |
| 4 | ~77 | ~77 |
| 5 | ~4 | ~4 |

The data shows that the activation of the depth testing only seems to have a major effect to the FPS when the subdivisions are lower. This is likely just a side effect of the higher FPS counters being more sporadic than its lower counterpart. Therefore, it is most likely that depth testing does not have a very noticeable effect on the FPS.

Run some tests with Backface Culling On and Off in combination with Lighting On and Off, while keeping everything else as above. When Lighting is On is there a difference in Frame Rate when Backface Culling is On vs Off? Describe Why or Why Not and show data to support your answer. Did you expect there to be a difference? Why?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Subdivisions | FPS with Defaults | FPS without Lights or Backface Culling | FPS without Lighting only | FPS without Back Culling only |
| 1 | ~4000 | ~3950 | ~3950 | ~4000 |
| 2 | ~3600 | ~3500 | ~3550 | ~3550 |
| 3 | ~1100 | ~1075 | ~1075 | ~1050 |
| 4 | ~77 | ~75 | ~75 | ~76 |
| 5 | ~4 | ~3.85 | ~3.85 | ~3.9 |

There seems to be a lower frame rate when the back face and lighting is disabled, this is as expected, considering the previous test results also causing lower frames. However it would seem that overall there is no major difference when compared to the default values, which is expected as the majority of computation effecting the frames will be the subdivisions.

Run some tests with Depth Testing On and Off in combination with Backface Culling On and Off, while keeping everything else as above. When Depth Testing is On is there a difference in Frame Rate when Backface Culling is On vs Off? Describe Why or Why Not and show data to support your answer. Did you expect there to be a difference? Why?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Subdivisions | FPS with Defaults | FPS without Depth or Backface Culling | FPS without Depth Testing | FPS without Back Culling only |
| 1 | ~4000 | ~3900 | ~3900 | ~4000 |
| 2 | ~3600 | ~3500 | ~3550 | ~3550 |
| 3 | ~1100 | ~1075 | ~1100 | ~1050 |
| 4 | ~77 | ~76 | ~77 | ~76 |
| 5 | ~4 | ~3.9 | ~4 | ~3.9 |

The combination of no depth testing or back culling seems to give roughly the addition their differences to the default column. Though in general it seems that depth testing improves framerate with back face culling on or off, likely due to the faces not culled remaining unrendered within the scene.

Discuss the performance characteristics of adding lights to the scene. Include a chart showing impact on frame rate for number of lights from 0 to 9. Discuss the shape of the curve and what it means.

The general curve trend for all subdivisions seems to be a minor decrease in frames as the number of lights is increased. However, there are some cases such as the 1st subdivision, in which the framerate increases slightly as the final lights are added. This may be due to the material of the sponge having already reached a complete white colour, therefore no additional light will change it.

Is there anything you found interesting or unexpected while running the above tests? Explain why.

The lighting being off resulted in a decrease to the framerate was unexpected, as I believed that the lack of computing the specular lighting would result in faster frames. However, it would seem this is not the case and the inclusion of all the lighting components results in a faster FPS.

## SCENE 2

Start your testing at subdivision level 1 (base), Lighting On (1 light), Backface Culling On and Depth Testing On.

Describe how you have decided to handle normal vectors. Are you specifying them per-vertex or per-face? Are you calculating them on the CPU or GPU? If CPU, how do you communicate them to the GPU? Are you storing them in a data structure or are you calculating them when needed in the shader?

The normals are being calculated within the geometry shader, per face on the fly. This is done through the equation, normalize (cross((pos1 – pos0),(pos2 – pos0))). This equation uses each of the three vertex points given to the triangle primitive and passes the normal to the fragment shader.

Vary the subdivision level and move around the scene. Describe the performance characteristics you're seeing at the different levels of subdivision? Is the scene getting smoothly animated as you move around? Does it seem to speed up and slow down depending on what's currently being rendered? Why? At what level of subdivision do you start to notice that your machine is struggling with the drawing load? What are some things that **might** be causing it to 'struggle'?

When increasing the subdivisions of the sponge, the model seems to move faster around the sponge. The increase in subdivisions also seems to cause the movement to become pause for a short time. This is likely due to the computing time increasing for the calculation for the vertex and face arrays.

Create a table showing the average frame rate, number of vertices and number of faces at each level of subdivision that your hardware can handle with a frame rate greater than 1 frame per second.

|  |  |  |  |
| --- | --- | --- | --- |
| Subdivisions | Frames Per Second | Vertices | Faces |
| 1 | ~3400 | 8 | 12 |
| 2 | ~3400 | 160 | 240 |
| 3 | ~3400 | 3200 | 4800 |
| 4 | ~3300 | 64000 | 96000 |
| 5 | ~745 | 1280000 | 1920000 |

Note: The data structure was not allowed to hold more than 5 subdivisions of vectors, therefore I had to stop at 5 subdivisions even though the frame rate didn’t suffer.

Draw a chart showing the average frame rate achieved at each level of subdivision. Compare this to the results you had for Scene 1. What is the data telling you about Immediate Mode vs Modern Mode? What sort of speed-up are you seeing?

It seems that the Immediate mode fps is better with the simpler shapes as it is better than modern mode for Subdivision 1 and 2. However once the number of faces and vertices exceeds 1000 the immediate mode implementation tanks in FPS, whilst the modern approach remains rather linear until subdivision 5.

Run some tests with Lighting Off while keeping everything else as above. Are the performance characteristics similar as for Scene 1? Why or Why Not? Use a table and a chart to show the comparison.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Subdivisions | IM FPS with Lighting | IM FPS without Lighting | IM FPS with Lighting | IM FPS without Lighting |
| 1 | ~4000 | ~3950 | ~3400 | ~3320 |
| 2 | ~3600 | ~3550 | ~3400 | ~3320 |
| 3 | ~1100 | ~1075 | ~3400 | ~3320 |
| 4 | ~77 | ~75 | ~3300 | ~3300 |
| 5 | ~4 | ~3.85 | ~745 | ~780 |

Both scenes follow a similar trend of having slightly lower frames without lights. The only outlier is subdivision 5 for MM where the frames seem to increase once the shape gets too complex

Run some tests with Backface Culling On and Off, while keeping everything else as above. Are the performance characteristics similar as for Scene 1? Why or Why Not? Use a table and a chart to show the comparison.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Subdivisions | IM FPS with Back Culling | IM FPS without Back Culling | MM FPS with Back Culling | MM FPS without Back Culling |
| 1 | ~4000 | ~4000 | ~3400 | ~3400 |
| 2 | ~3600 | ~3550 | ~3400 | ~3400 |
| 3 | ~1100 | ~1050 | ~3400 | ~3375 |
| 4 | ~77 | ~76 | ~3300 | ~3300 |
| 5 | ~4 | ~3.9 | ~745 | ~580 |

Both scenes follow a similar trend of having slightly lower frames without backface culling.

Answer here.

Run some tests with Depth Testing On and Off, while keeping everything else as above. Are the performance characteristics similar as for Scene 1? Why or Why Not? Use a table and a chart to show the comparison.

Answer here.

Run some tests with Backface Culling On and Off in combination with Lighting On and Off, while keeping everything else as above. When Lighting is On is there a difference in Frame Rate when Backface Culling is On vs Off? Describe Why or Why Not and show data to support your answer. Did you expect there to be a difference? Why?

Answer here.

Discuss the performance characteristics of adding lights to the scene. Include a chart showing impact on frame rate for number of lights from 0 to 9. Discuss the shape of the curve and what it means. Is there any difference between these results and Scene 1 results?

Answer here.

Is there anything you found interesting or unexpected while running the above tests? Explain why.

Answer here.

## SCENE 3

Create a table and chart showing the frame rate for each level of subdivision your machine can handle with a frame rate greater than 1 frame per second.

|  |  |
| --- | --- |
| Subdivisions | Frames Per Second |
| 1 | ~3200 |
| 2 | ~3200 |
| 3 | ~3150 |
| 4 | ~1400 |
| 5 | ~95 |

Is this what you expected? Why or Why Not?

This outcome was very unexpected, the frames remaining roughly the same until 4 subdivisions did not follow the trend set by the previous scenes of a steady decline

Use a table and a chart to show the difference in performance between using GL\_STATIC\_DRAW and GL\_DYNAMIC\_DRAW in your calls to glBufferData(). Run the tests manually by changing the code and recompiling your project.

|  |  |  |
| --- | --- | --- |
| Subdivisions | Dynamic FPS | Static FPS |
| 1 | ~3150 | ~3200 |
| 2 | ~3150 | ~3200 |
| 3 | ~3150 | ~3150 |
| 4 | ~1420 | ~1400 |
| 5 | ~95 | ~95 |

Discuss the results and whether it is what you expected and, if the two differ, why you think they differ.

The results show next to no change with the new drawing type which is odd considering the number of computations should be increased as the data should be redrawn multiple times with dynamic draw, which should’ve in turn decreased the frame rate of the scene.

## SCENE 4

Is there any difference in performance compared to Scene 3? Is this what you expected? Why or Why Not?

Both scenes performed equally in all areas. This is rather unexpected as I anticipated the built in instancing to works better than simply calling the draw function in a for loop. However since the number of instances are relatively small, this outcome perhaps should have been expected.

## SCENE 5

There are two sets of position coordinates in your C++ vertex array for this Scene, with three floats each, representing "home position" and "morphed position" for each vertex. You have changed the Vertex Array Object to use the morphed position as the position attribute that is used by the vertex shader. Use RenderDoc to find this data and confirm whether, on the GPU, only the morphed position is being sent across (3 floats) or both the morphed position and the home position (6 floats). Include a screenshot from RenderDoc showing this.

Graphical user interface, application, table

Description automatically generatedIs this what you expected? Why or Why Not?

Yes I anticipated 3 float values as I did all the lerp calculations on the CPU end and updated the array before the GPU received it. Therefore, it does not make sense for the GPU to have that information

## SCENE 6

Show a table and a chart comparing the performance (frames per second) of Scene 5 and Scene 6 at different model subdivisions.

|  |  |  |
| --- | --- | --- |
| Subdivisions | Scene 5 | Scene 6 |
| 1 | ~125 | ~3200 |
| 2 | ~125 | ~3200 |
| 3 | ~120 | ~3200 |
| 4 | ~100 | ~3150 |
| 5 | ~17 | ~750 |

Discuss what the data is showing.

The difference in frame rate appearing here is a clear indicator that computing changes for a vector point is far better to be done on the GPU as it doesn’t require the costly action of reassigning the VBO buffer every frame. The 5th scene required the entire vertex array to be sent over and over to the GPU, meaning that for the larger Subdivisions the amount of float values being sent every frame reached the millions. As for the 6th scene, the vector points are only required to be sent to the GPU every time the subdivisions are updated, instead only sending the relevant timing data in a uniform. This method is evidently far more efficient for the program.