# Real-Time Rendering and 3D Games Programming

# ASSIGNMENT 1 – REPORT (v1.1)

## 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, ...).

<https://github.com/rosamarco/menger-sponge>

<https://thecodingtrain.com/CodingChallenges/002-mengersponge.html>

These are the resources used to help me create the Menger Sponge, where they helped with the initial process of creating cube objects to help describe the concept more cleanly. By having loops that went through making 20 cubes from 1 cube that made the basis of a single Menger Sponge at subdivision 1, it helped conceptualize the idea in programming. However, this left the idea of making the cubes being created at the right locations and with the right vertices, so I had to add my own algorithm of having each cube be created around the vertices of the center cube, as it had the most consistent vertices throughout each subdivision.

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?

CPU: Intel(R) Core(TM) i5-8500 CPU @ 3.00GHz

GPU: NVIDIA GeForce GTX 1080

Screen Resolution: 1920 x 1080

Refresh Rate: 60 Hz

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?

I majorly used c++’s vectors, which are flexible arrays that did not need to be manually resized, as I figured it would be better for the cpu to handle the data structure whenever it needed to, rather than personally pinpointing every step of when to manually resize a regular array. This allowed further flexibility for when I had to calculate and replace vertices, and store them within other vectors, which resulted in some 2D and 3D vectors. Furthermore, I have duplicating vertices within my common, shared data structure, although for some certain scenes I had to manipulate the common shared data structure within the individual scenes to make them work with the various OpenGL drawing methods. Although generally, all scenes had data structures with shared vertices. Additionally, I used GL\_TRIANGLES as the main drawing primitive for all my cube rendering.

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?

All faces of a cube had the same colours across each, although the colours present in scene 1 will differ from those in scene 2-5. Furthermore, the ambient and diffuse materials were assigned per vertex, with the specular added if there was only light active. Additionally, scene 1 used immediate mode material assignment through glMaterialfv, whereas the materials for scene 2-5 was communicated through uniforms in the fragment shader.

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?

Generally, I have positioned my light sources, at first, around each of the cubes, with one point light each. The remainder of the point lights were positioned at around the outmost vertex points, to help ensure there’s lighting in most areas of the cube.

## 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.

Lighting On & Backface On & Depth On

|  |  |  |  |
| --- | --- | --- | --- |
| Subdivision | Average FPS | # Vertices | # Faces |
| 1 | 60 | 8 | 6 |
| 2 | 60 | 160 | 120 |
| 3 | 60 | 3200 | 2400 |
| 4 | 12 | 64000 | 48000 |

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.

Lighting Off & Backface On & Depth On

|  |  |  |  |
| --- | --- | --- | --- |
| Subdivision | Average FPS | # Vertices | # Faces |
| 1 | 60 | 8 | 6 |
| 2 | 60 | 160 | 120 |
| 3 | 60 | 3200 | 2400 |
| 4 | 20 | 64000 | 48000 |

The affect with no lighting compared to lighting is minimal at best, with a slight increase in FPS with no lighting, which is understandable as there would be more processes ran with lighting on. Even with the massive drop in FPS when going from subdivision 3-to-4, the curve is very similar between both data graphs, indicating that performance is not severely impacted by having lighting on, but can be a major factor with further subdivisions.

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.

Lighting On & Backface Off & Depth On

|  |  |  |  |
| --- | --- | --- | --- |
| Subdivision | Average FPS | # Vertices | # Faces |
| 1 | 60 | 8 | 6 |
| 2 | 60 | 160 | 120 |
| 3 | 60 | 3200 | 2400 |
| 4 | 12 | 64000 | 48000 |

There seems to be little to no difference between having backface culling on or off, at least to the 4th subdivision in my implementation.

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.

Lighting On & Backface On & Depth Off

|  |  |  |  |
| --- | --- | --- | --- |
| Subdivision | Average FPS | # Vertices | # Faces |
| 1 | 60 | 8 | 6 |
| 2 | 60 | 160 | 120 |
| 3 | 60 | 3200 | 2400 |
| 4 | 11 | 64000 | 48000 |

The effect of having Depth test off, in comparison to have it on seems to be very minimal, as it only has a difference of -1 FPS when the depth test is off.

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?

Lighting Off & Backface Off & Depth On

|  |  |  |  |
| --- | --- | --- | --- |
| Subdivision | Average FPS | # Vertices | # Faces |
| 1 | 60 | 8 | 6 |
| 2 | 60 | 160 | 120 |
| 3 | 60 | 3200 | 2400 |
| 4 | 20 | 64000 | 48000 |

While lighting is on, there is no difference between having backface on/off, shown by prior data. Hence, judging by the effect of having lighting on/off and backface on/off, there seems to be an FPS difference if lighting is on/off, but backface does not seem to make any noticeable difference on the FPS. By having prior data kept, there should be no surprise that backface does not impact performance, at least in comparison to the much more impactful lighting effects.

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?

Lighting On & Backface Off & Depth Off

|  |  |  |  |
| --- | --- | --- | --- |
| Subdivision | Average FPS | # Vertices | # Faces |
| 1 | 60 | 8 | 6 |
| 2 | 60 | 160 | 120 |
| 3 | 60 | 3200 | 2400 |
| 4 | 11 | 64000 | 48000 |

The effect as shown with having a combination of backface on/off and depth testing on/off, is that having depth testing on provides a slight FPS increase of 1 FPS, whereas the backface culling has little to no noticeable effect, which is consistent to the results it has provided in prior questions. Therefore, there should be no major difference expected from backface and depthtesting.

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.

From my implementation of the lights in scene 1, it seems that there is a consistent FPS rate of 60 across the board, regardless of the amount of lights I have added to the scene. There can only be two conclusions from this; the first is that adding any amount of lights will seemingly have no effect on FPS, or the more reasonable conclusion is that my light source implementation is not working at all and will have no effect on the scene as it is not working properly. Hence, there is perhaps no surprise to why the lights have no impact on the FPS.

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

The major surprise is discovering that backface culling has little to no impact on the FPS, as I figured any changes to an object would have some impact, yet backface produced nothing noticeable.

## 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?

In this scene, I have made the decision of having each of my vertices hold one normal that is calculated based on the face direction of the cube, so the shaders would receive them in the form of layout position 2 in the vertex shaders. Hence, these normal have to be stored prior, where they are stored in the common shared data structure that all scenes share.

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'?

Around subdivision levels 1-3 there seems to be no significant performance drops, although scene 3 suffers from more noticeable drops. It is only from subdivision 4 onwards that there are more major performance drops, which causes the movement to appear more sluggish and unmoveable at times. Especially for subdivision 5, as the movement would either stay static, or teleport to another part of the scene, although the FPS counter is still above 0 somehow. The biggest offender for causing these performance drops would be the implementation of the cube drawing, as it requires a modification to the base, shared data structure in order to use it more effectively, as all the drawing is done through multiple for-loops and no recursion.

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.

Lighting On & Backface On & Depth On

|  |  |  |  |
| --- | --- | --- | --- |
| Subdivision | Average FPS | # Vertices | # Faces |
| 1 | 60 | 8 | 6 |
| 2 | 60 | 160 | 120 |
| 3 | 60 | 3200 | 2400 |
| 4 | 43 | 64000 | 48000 |
| 5 | 2 | 1280000 | 960000 |

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?

The biggest difference is that scene 2 can achieve the 5th subdivision without having its FPS counter go all the way down to 0, which is a massive improvement over scene 1 who could not achieve that without major impacts to performance. Hence, Modern Mode offers, from the data, a much faster performance in comparison to Immediate Mode, ensuring all items run much more smoothly and at renders objects at a faster speed without impacting performance as much.

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.

Lighting Off & Backface On & Depth On

|  |  |  |  |
| --- | --- | --- | --- |
| Subdivision | Average FPS | # Vertices | # Faces |
| 1 | 60 | 8 | 6 |
| 2 | 60 | 160 | 120 |
| 3 | 60 | 3200 | 2400 |
| 4 | 44 | 64000 | 48000 |
| 5 | 2 | 1280000 | 960000 |

While the curvature between scene 1 and 2 are similar, there is still the difference of scene 2 being able to produce the 5th subdivision with an FPS counter above 0. Furthermore, while the curves may be similar, scene 1 has a much more significant drop in FPS in the 4th subdivision, whereas scene 2 suffers a more minor drop in comparison. All of these comparisons continue to show that Modern Mode has better capabilities to render objects at a faster rate.

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.

Lighting On & Backface Off & Depth On

|  |  |  |  |
| --- | --- | --- | --- |
| Subdivision | Average FPS | # Vertices | # Faces |
| 1 | 60 | 8 | 6 |
| 2 | 60 | 160 | 120 |
| 3 | 60 | 3200 | 2400 |
| 4 | 46 | 64000 | 48000 |
| 5 | 2 | 1280000 | 960000 |

As shown prior with scene 1, and with the knowledge that backface has relatively no effect on performance, scene 2 has a better performance overall in comparison to scene 1’s performance with the same characteristics, even though the curves are similar.

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.

Lighting On & Backface On & Depth Off

|  |  |  |  |
| --- | --- | --- | --- |
| Subdivision | Average FPS | # Vertices | # Faces |
| 1 | 60 | 8 | 6 |
| 2 | 60 | 160 | 120 |
| 3 | 60 | 3200 | 2400 |
| 4 | 45 | 64000 | 48000 |
| 5 | 2 | 1280000 | 960000 |

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?

Lighting On & Backface Off & Depth On

|  |  |  |  |
| --- | --- | --- | --- |
| Subdivision | Average FPS | # Vertices | # Faces |
| 1 | 60 | 8 | 6 |
| 2 | 60 | 160 | 120 |
| 3 | 60 | 3200 | 2400 |
| 4 | 45 | 64000 | 48000 |
| 5 | 2 | 1280000 | 960000 |

With the prior knowledge of face culling having little to no impact on performance, it is no surprise that the curves between having lighting on and switching between on/off face culling would have no impact at all.

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?

Surprisingly, adding additional light sources does not have any impact on performance at all, as the counter stayed at 60 FPS with the default testing requirements, which makes it have a similar effect to scene 1. There seemed to be only performance drops with an increase in subdivision, but the lighting itself seemed to have no effect.

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

The most interesting thing from doing the tests for scene 2 is discovering that apparently adding additional lights to the scene does not impact performance whatsoever, but rather it is the amount of cubes being rendered in the scene that has a more significant impact. While my implementation of scene 1 point lights may be incorrect, it may seem that any light processes do not hinder performance as I originally thought.

## 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.

|  |  |
| --- | --- |
| Subdivision | Average FPS |
| 1 | 60 |
| 2 | 60 |
| 3 | 58 |
| 4 | 21.4 |

Is this what you expected? Why or Why Not?

While I did expect a performance drop, I was not expecting the slight dip in subdivision 3, before subdivision 4 suffered the major performance drop. As a result, my scene 3 could not render subdivision 5 before it completed crashed, which is not surprising once subdivision 4’s FPS was known.

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.

GL\_STATIC\_DRAW

|  |  |
| --- | --- |
| Subdivision | Average FPS |
| 1 | 60 |
| 2 | 60 |
| 3 | 58 |
| 4 | 21.4 |

GL\_DYNAMIC\_DRAW

|  |  |
| --- | --- |
| Subdivision | Average FPS |
| 1 | 60 |
| 2 | 60 |
| 3 | 59 |
| 4 | 3 |

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

While both curves are similar in shape, GL\_DYNAMIC\_DRAWS has the bigger drop in performance when going from subdivision 3-to-4. This is to be expected as due to the nature of dynamic draws, it will modify the data it receives and render the object as many times as it must, whereas static draws only need to modify the data once before it will render the object at the same rate as dynamic draw. Therefore, as dynamic draws must repeat an action that static draw only does once, it will incur more memory usage and “calls”, meaning more resources are used and allocated to this function. Hence, having dynamic draws suffer a major performance drop, in comparison to static draws, is within scope of how the two calls function.

## SCENE 4

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

|  |  |
| --- | --- |
| Subdivision | Average FPS |
| 1 | 60 |
| 2 | 60 |
| 3 | 60 |
| 4 | 11 |

Using the average FPS data as a benchmark around subdivision 4, where the performance usually drops across all my scenes, it can be determined that glDrawArraysInstanced suffers a further performance drop in comparison to the calls made to glDrawElements. This is expected, as there must be more memory allocated for the instance call, due to requiring further memory to store the vertices, whereas drawing elements trades that memory allocation for element buffer objects, which just uses the face data to draw the required triangles. Hence, due to the nature of instance calls, scene 4 having the greater drop in performance is not unexpected.

## 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

Description automatically generated

These were the only values that caught my attention of having 3 floats that contained positions, which is around 8 of them in the glUniform3f.

Is this what you expected? Why or Why Not?

This was to be expected as my vertex implementation only sent 3 floats per vertex to the shaders, and these floats were only the modified floats.

## SCENE 6

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

|  |  |  |
| --- | --- | --- |
| Subdivision | FPS(Scene 5) | FPS(Scene 6) |
| 1 | 60 | 60 |
| 2 | 60 | 60 |
| 3 | 60 | 60 |
| 4 | 12.85 | 45 |
| 5 | N/A | 2 |

Discuss what the data is showing.

By comparing the data between scene 5 and 6, it can be determined that scene 6 had the stronger performance overall, which helps us determine that doing calculations in the shader/GPU is much more efficient compared to doing them in the classes/CPU. Hence, letting the GPU doing the calculations in comparison to the CPU will apparently result in a stronger performance overall, although it can still suffer from the same performance drops, but to a lesser degree.