Real-Time Rendering and 3D Games Programming

ASSIGNMENT 1 – REPORT (v1.0)

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 make the Menger Sponge, after reviewing some approaches online for 2d and 3d solutions I created my own version of the generation algorithm. Below I have listed the main recourse that has influence in my algorithm.

The Coding Train

https://youtu.be/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?

GPU: RTX 3060

CPU: Intel Core i9-10900K CPU @ 3.70GHz 3.70 GHz

RAM: 16 GB

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?

Data Structure:

The data structure used to store the Menger Sponge is a simple **c++** std::vector of custom data type Vertex. Vertex contains a **glm::vec3 Position**, and a **glm::vec3 Normal**.

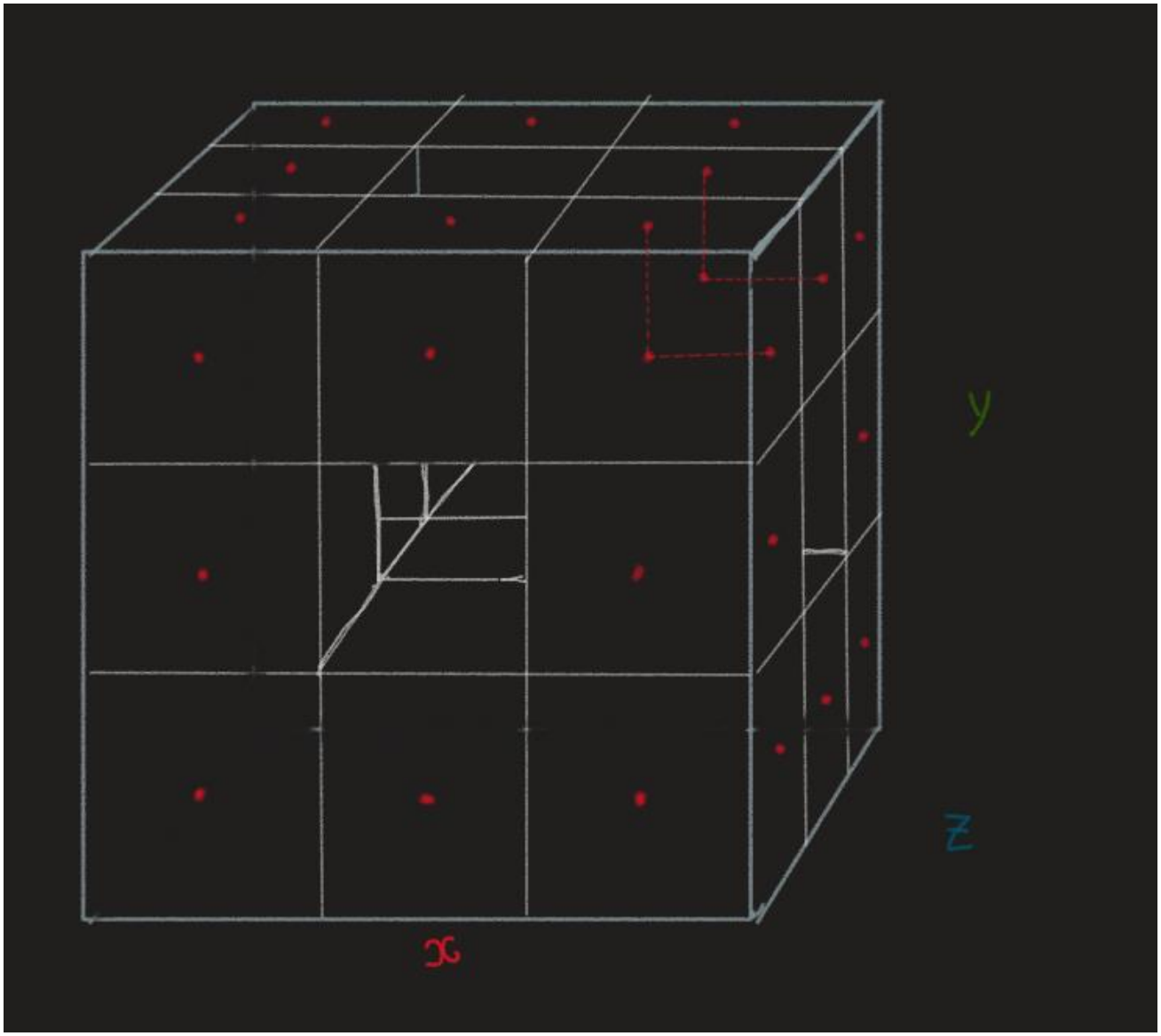
Secondly a std::vector of uint32_t is used to store the indices of all the triangles that make up the final mesh.

Algorithm:

Step 1) Subdivision

Using three nested for loops where: **x = -1 y = -1 z = -1** and the termination condition is **value < 2** we can create an origin point for 27 unit cubes.

With a simple condition check in the body where if the absolute value of x + y + z > 1 we can filter any origin points that have two or more axis values of **0**, thus removing any origin points that belong to the "cross" boolean difference that we wish to remove from the cube.



We then use a given size **S** to scale the current origin point by **S / 3** and once checking if we have reached the desired subdivision, we either construct (Compile) the Menger Sponge or recursively call **Subdivide** with each new origin, size **S / 3** and **current division – 1**

Step 2) Compilation

Once a recursive branch has reached its termination condition where the current division == **0** we construct the triangles, vertices and indices for a cube at the current origin point of the divided size.

This construction method uses indices such that each cube will have no duplicate vertices, please note however the final collection of “cubes” will have internal faces and duplicate vertices.

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?

Using the normal direction calculated in a geometry shader, I am using a simple function that finds the greatest absolute value of a 3-component vector and returns a material index from 0 – 2, thus giving allowing the dynamic allocation of three materials to parallel faces / triangles.

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?

The point light positions are determined by a spherical distribution function that randomly finds a point on a sphere of radius **R**, similarly the values of the lights are generated as a random float between 0.0 and 1.0.

This is a simple way to add n lights to a scene in a random but predictable fashion.

SCENE 1

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

Subdivision	Frame Rate	Vertex Count	Face / Triangle Count
1	3305	160	240
2	572	3200	4800
3	32	64000	96000
4	1.65	1280000	1920000

average frame rate achieved at each level of subdivision with lights, back face culling and depth testing.

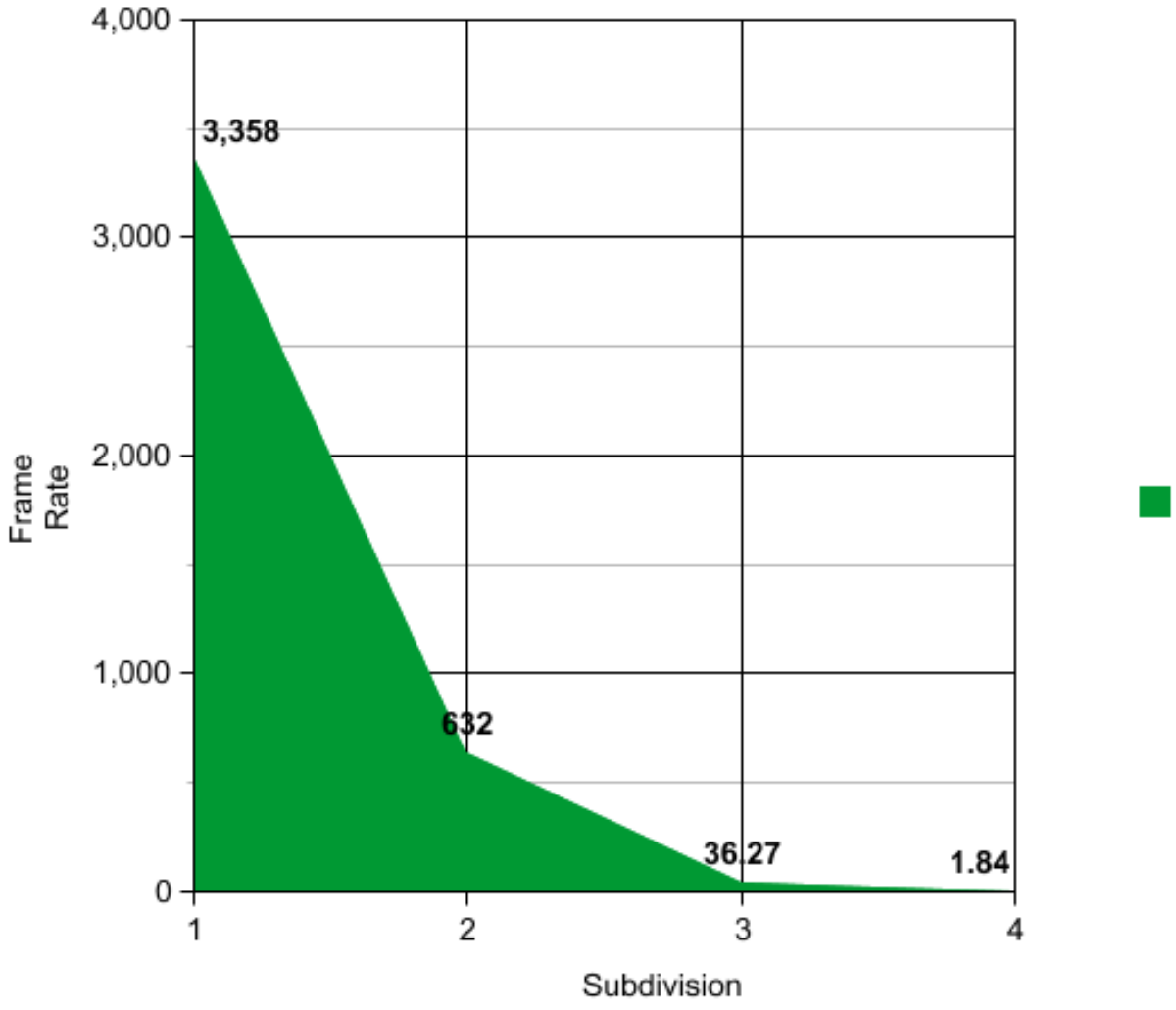
Run some tests with Lighting Off while keeping everything else as above. Describe the impact this has on frame rate and why?

Subdivision	Frame Rate	Vertex Count	Face / Triangle Count
1	3321	160	240
2	629	3200	4800
3	36.27	64000	96000
4	1.82	1280000	1920000

average frame rate achieved at each level of subdivision with back face culling and depth testing.

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.

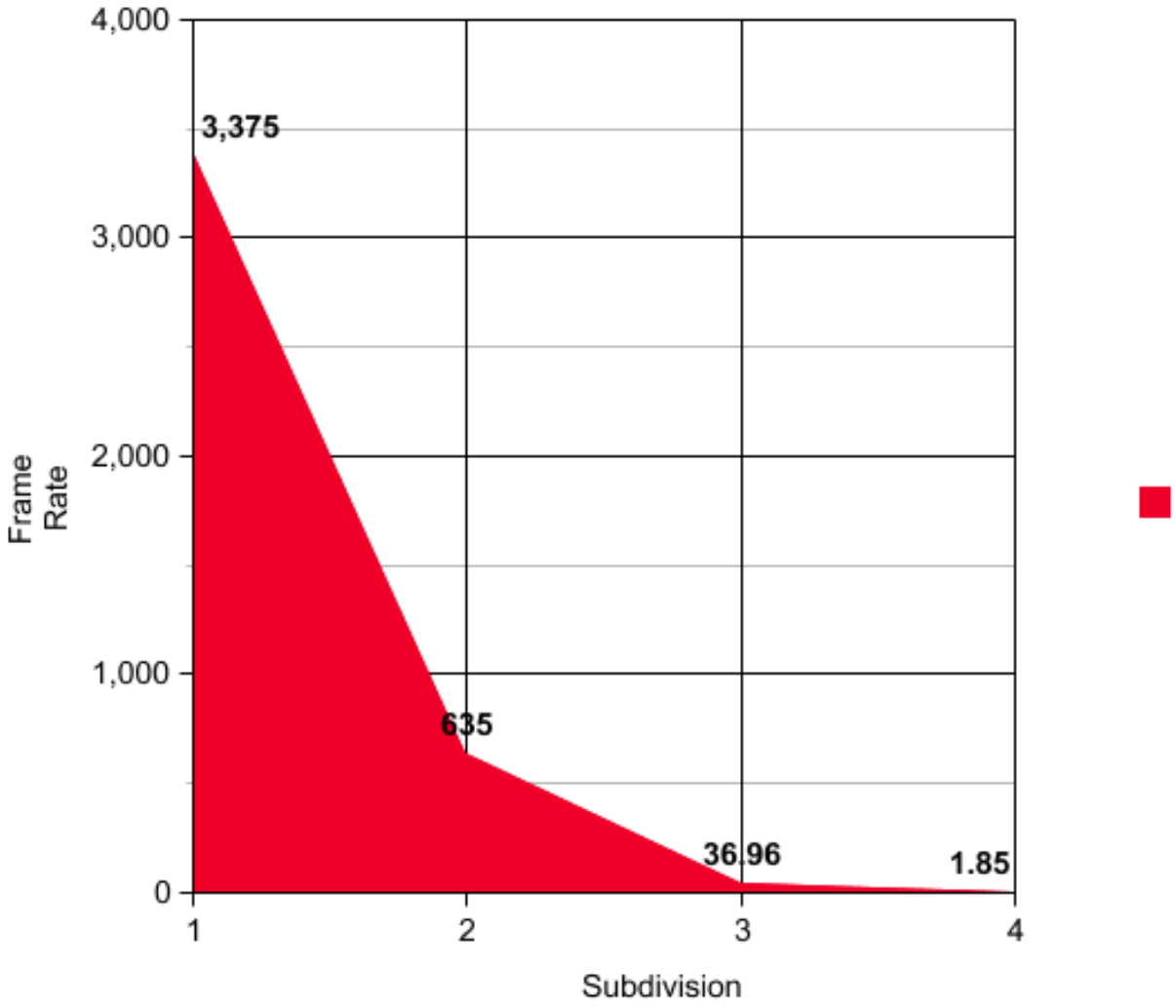
Subdivision	Frame Rate	Vertex Count	Face / Triangle Count
1	3358	160	240
2	632	3200	4800
3	36.27	64000	96000
4	1.84	1280000	1920000



average frame rate achieved at each level of subdivision with depth testing.

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.

Subdivision	Frame Rate	Vertex Count	Face / Triangle Count
1	3375	160	240
2	635	3200	4800
3	36.96	64000	96000
4	1.85	1280000	1920000



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?

Backface Culling On / Lighting On

Subdivision	Frame Rate	Vertex Count	Face / Triangle Count
1	3256	160	240
2	572.4	3200	4800
3	32.64	64000	96000
4	1.61	1280000	1920000

Backface Culling Off / Lighting On

Subdivision	Frame Rate	Vertex Count	Face / Triangle Count
1	3280	160	240
2	574	3200	4800
3	32.74	64000	96000
4	1.67	1280000	1920000

Backface Culling On / Lighting Off

Subdivision	Frame Rate	Vertex Count	Face / Triangle Count
1	3352	160	240
2	626	3200	4800
3	36.28	64000	96000
4	1.84	1280000	1920000

Backface Culling Off / Lighting Off

Subdivision	Frame Rate	Vertex Count	Face / Triangle Count
1	3371	160	240
2	628	3200	4800
3	36.42	64000	96000
4	1.85	1280000	1920000

When lighting is on and many triangles are being rendered, I would expect to see an increase in FPS if back face culling were enabled, however at the maximum level of division in this immediate mode scene any difference in frame rate is negligible.

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?

Depth Testing On / Lighting Off / Back Face Culling On

Subdivision	Frame Rate	Vertex Count	Face / Triangle Count
1	3366	160	240
2	627	3200	4800
3	36.35	64000	96000
4	1.84	1280000	1920000

Depth Testing On / Lighting Off / Back Face Culling Off

Subdivision	Frame Rate	Vertex Count	Face / Triangle Count
1	3372	160	240
2	625	3200	4800
3	36.4	64000	96000
4	1.85	1280000	1920000

Depth Testing Off / Lighting Off / Back Face Culling On

Subdivision	Frame Rate	Vertex Count	Face / Triangle Count
1	3351	160	240
2	628	3200	4800
3	36.1	64000	96000
4	1.84	1280000	1920000

Depth Testing Off / Lighting Off / Back Face Culling Off

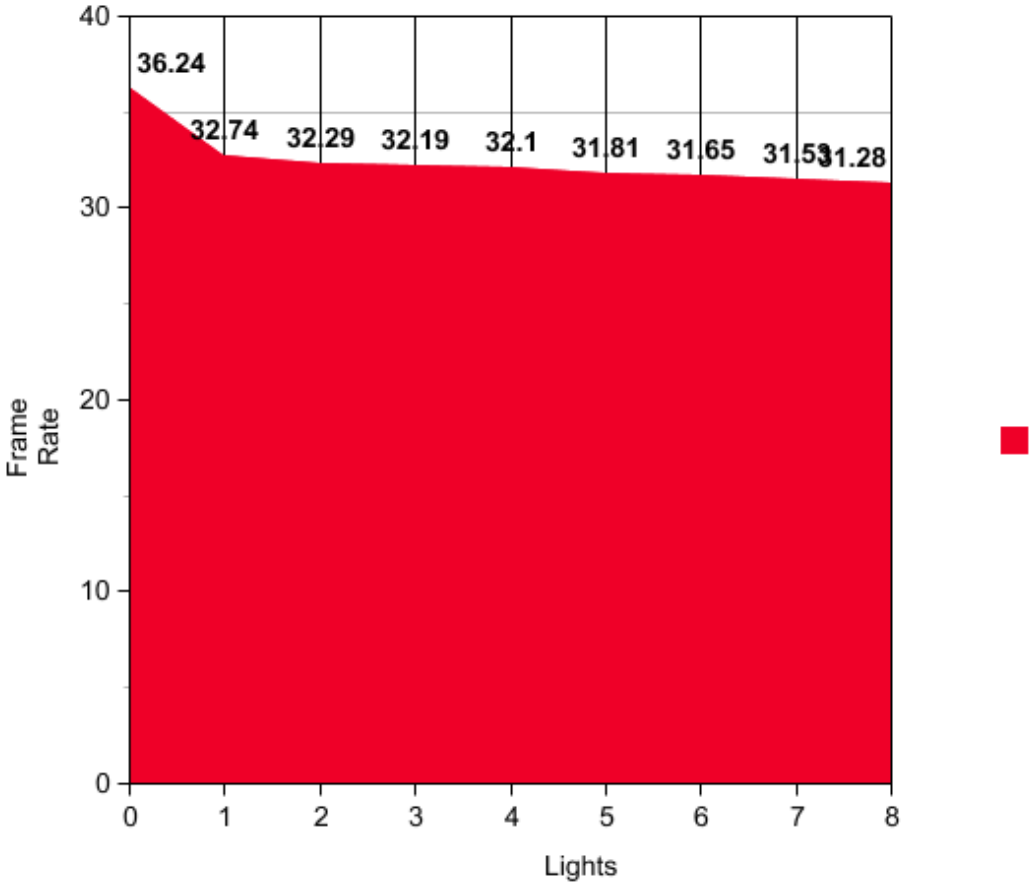
Subdivision	Frame Rate	Vertex Count	Face / Triangle Count
1	3380	160	240
2	630	3200	4800
3	36.5	64000	96000
4	1.84	1280000	1920000

As depth testing is a computationally heavy operation, I would expect to see a general decrease in frame rate and with back face culling enabled to gain some of that performance back especially as the scene grows in number of triangles.

However as seen in the previous test the immediate mode scene does not appear to be allow so many triangles to see the benefits of back face culling or the performance hit from depth testing. I assume that if this test was run on an older generation hardware the changes would be more prominent.

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.

Subdivision	Lights	Frame Rate	Vertex Count	Face / Triangle Count
3	0	36.24	64000	96000
3	1	32.74	64000	96000
3	2	32.29	64000	96000
3	3	32.19	64000	96000
3	4	32.1	64000	96000
3	5	31.81	64000	96000
3	6	31.65	64000	96000
3	7	31.53	64000	96000
3	8	31.28	64000	96000



The frame rate of the immediate mode scene decreases by a small margin as the lights increase, I presume that again the difference would be more noticeable on an older generation graphics card.

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

Although the hardware I'm using is the most recent generation, I expected to find a greater measurable result between tests however the immediate mode scene reaches it's limits very quick.

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 vertex normal direction unit vectors are being calculated in the GLSL geometry shader using the cross product of two edges of a triangle, because they are calculated in the geometry shader on the GPU they are passed down the pipeline into the fragment shader for use during lighting calculations.

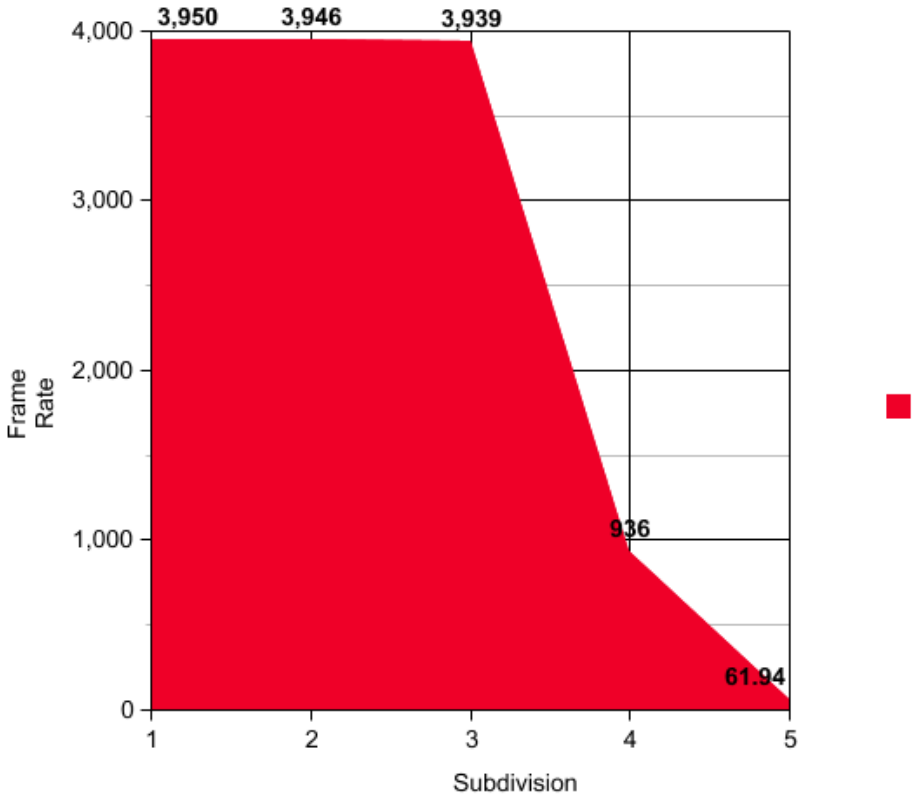
*Vary the subdivision level and move around the scene. Describe the performance characteristics you're seeing at the different levels of subdivision? Is the scene getting smoothly animated as you move around? Does it seem to speed up and slow down depending on what's currently being rendered? Why? At 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'?*

Moving around the scene at subdivisions < 5 I see no difference in the smoothness, however at level 5 subdivision when moving around the scene if all pixels are in the frame there are some noticeable changes in the smoothness intermittently which effect the movement and frame rendering time slightly.

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.

Subdivision	Frame Rate	Vertex Count	Face / Triangle Count
1	3950	160	240
2	3946	3200	4800
3	3939	64000	96000
4	936	1280000	1920000
5	61.94	25600000	38400000

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 modern scene can reach level five subdivision at a comfortable 60 frames per second, the immediate mode scene reaches its limit at 4 subdivisions and is already unusable. The modern pipeline can be pushed further than we have here however on my hardware I'm limited by RAM and thus cannot calculate a level 6 Menger Sponge to test it.

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.

Subdivision	Frame Rate	Vertex Count	Face / Triangle Count
1	3980	160	240
2	3955	3200	4800
3	3925	64000	96000
4	948	1280000	1920000
5	62.39	25600000	38400000

// TODO CHART

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.

Backface Culling Off

Subdivision	Frame Rate	Vertex Count	Face / Triangle Count
1	3951	160	240
2	3914	3200	4800
3	3784	64000	96000
4	624.72	1280000	1920000
5	53.67	25600000	38400000

// TODO CHART

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.

Culling Off / Depth Off

Subdivision	Frame Rate	Vertex Count	Face / Triangle Count
1	3990	160	240
2	3940	3200	4800
3	3952	64000	96000
4	575	1280000	1920000
5	49.1	25600000	38400000

// TODO CHART

Culling Off / Depth On

Subdivision	Frame Rate	Vertex Count	Face / Triangle Count
1	3960	160	240
2	3821	3200	4800
3	3570	64000	96000
4	578	1280000	1920000
5	48.8	25600000	38400000

// TODO CHART

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?

Backface Culling Off / Lighting Off

Subdivision	Frame Rate	Vertex Count	Face / Triangle Count
1	3960	160	240
2	3821	3200	4800
3	3570	64000	96000
4	578	1280000	1920000
5	48.8	25600000	38400000

Backface Culling Off / Lighting On

Subdivision	Frame Rate	Vertex Count	Face / Triangle Count
1	3920	160	240
2	3870	3200	4800
3	3650	64000	96000
4	559.45	1280000	1920000
5	46.3	25600000	38400000

Backface Culling On / Lighting Off

Subdivision	Frame Rate	Vertex Count	Face / Triangle Count
1	3966	160	240
2	3928	3200	4800
3	3833	64000	96000
4	887	1280000	1920000
5	59.1	25600000	38400000

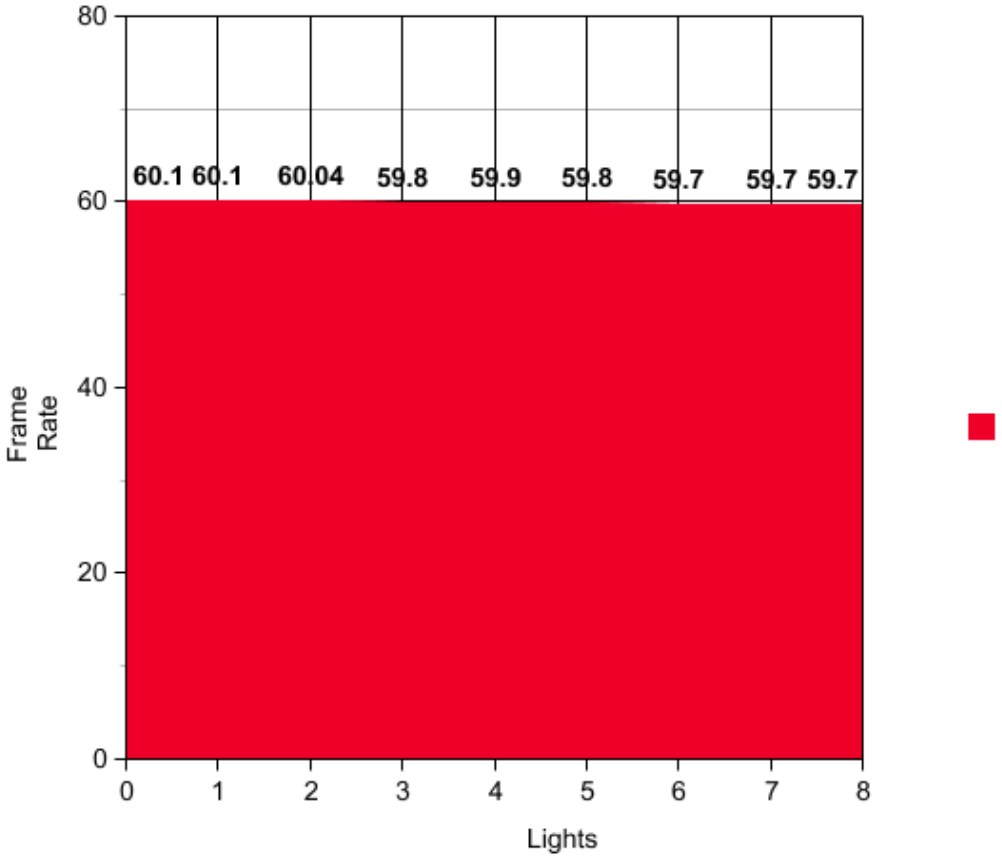
Backface Culling On / Lighting On

Subdivision	Frame Rate	Vertex Count	Face / Triangle Count
1	3946	160	240
2	3931	3200	4800
3	3876	64000	96000
4	882	1280000	1920000
5	57.7	25600000	38400000

As expected, when lighting is enabled, there is a gradual increase in performance as the number of triangles grow in the scene if back face culling is enabled. This is an expected result as the trade-off of calculating and removing back faces and calculating lighting for those faces will become more valuable as the number of faces increase.

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?

Subdivision	Lights	Frame Rate	Vertex Count	Face / Triangle Count
5	0	60.1	25600000	38400000
5	1	60.1	25600000	38400000
5	2	60.04	25600000	38400000
5	3	59.8	25600000	38400000
5	4	59.9	25600000	38400000
5	5	59.8	25600000	38400000
5	6	59.7	25600000	38400000
5	7	59.7	25600000	38400000
5	8	59.7	25600000	38400000



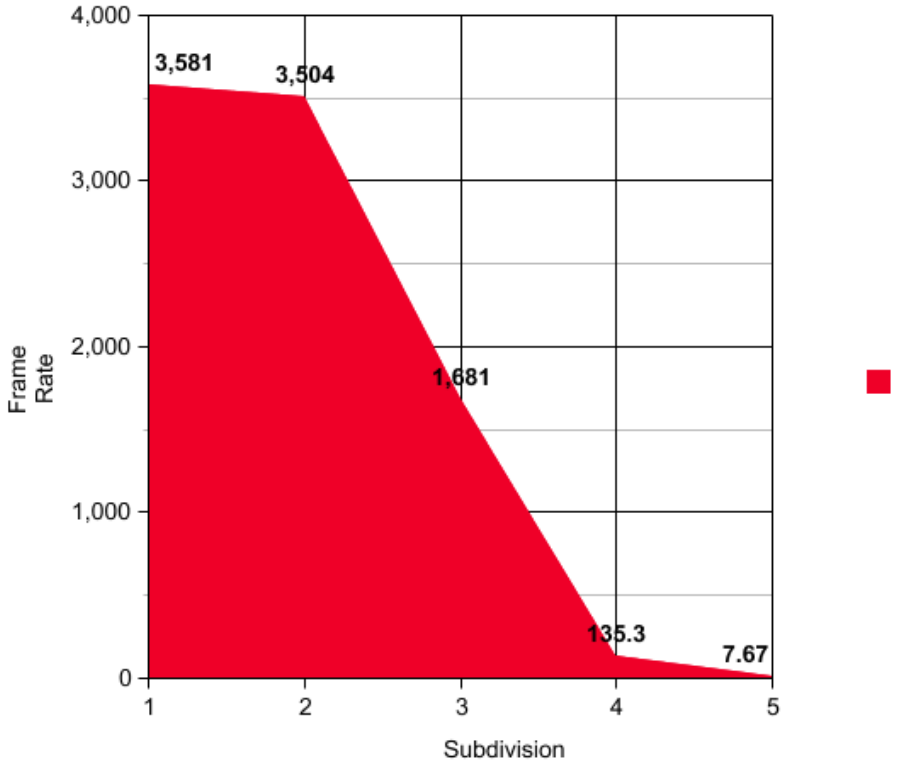
There is a very small decrease in frame rate as the number of lights increase towards 9, once we reach 100 lights in a level 5 scene the frame rate slows down to 11 fps.

Although the graph looks very close to constant, there is a point where it becomes parabolic.

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	Frame Rate	Vertex Count	Face / Triangle Count
1	3581	160	240
2	3504	3200	4800
3	1681	64000	96000
4	135.3	1280000	1920000
5	7.67	25600000	38400000



Is this what you expected? Why or Why Not?

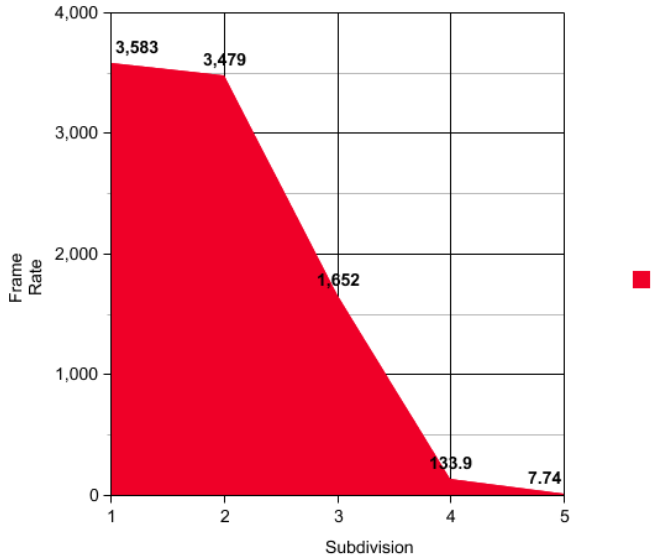
In comparison with the graph of from the modern non-instanced scene we see a steep drop off from subdivision two whereas with the non-instanced scene the drop off happens after the third division.

This is not unexpected because we are still making 9 draw calls to the GPU and thus cannot expect the same result.

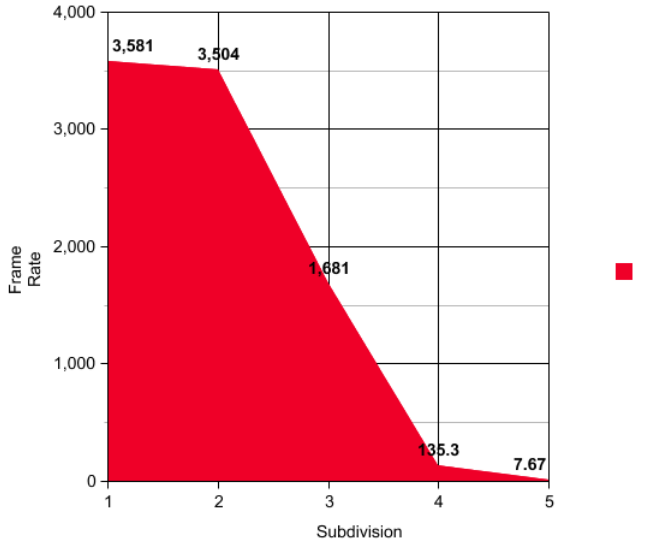
Use a table and a chart to show the difference in performance between using GL_STATIC_DRAW and GL_DYNAMIC_DRAW in your calls to glBufferData(). Run the tests manually by changing the code and recompiling your project.

DYNAMIC_DRAW

Subdivision	Frame Rate	Vertex Count	Face / Triangle Count
1	3583	160	240
2	3479	3200	4800
3	1652	64000	96000
4	133.9	1280000	1920000
5	7.74	25600000	38400000



STATIC_DRAW



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

There is a small variability between the results of DYNAMIC and STATIC draw in this case, however, not enough to make a definitive statement about their application. DYNAMIC draw is suggested for data that is regularly updated to the GPU and we can presume that this would be more apparent in some circumstances.

SCENE 4

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

Subdivision	Frame Rate	Vertex Count	Face / Triangle Count
1	3809	160	240
2	3799	3200	4800
3	1455	64000	96000
4	103.79	1280000	1920000
5	5.73	25600000	38400000

As the number of triangles increase the difference between scene 3 and 4 becomes larger such that the frame rate in scene 4 is slightly less. This isn't necessarily unexpected because the value of instancing on the GPU is the decrease in draw calls, however if the triangle count of a single instance is very large it's not unreasonable to see a worse performance overall.

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.

Buffer Contents

Element	Position0			pad[0]	pad[1]	pad[2]
0	-5.48277	-5.9214	-5.9214	00000000	00000000	00000000
1	-5.62467	-5.62467	-6.07464	00000000	00000000	00000000
2	-5.35112	-5.35112	-5.35112	00000000	00000000	00000000
3	-5.62467	-6.07464	-5.62467	00000000	00000000	00000000
4	-5.77921	-5.77921	-5.77921	00000000	00000000	00000000
5	-5.9214	-5.9214	-5.48277	00000000	00000000	00000000
6	-6.07464	-5.62467	-5.62467	00000000	00000000	00000000
7	-5.9214	-5.48277	-5.9214	00000000	00000000	00000000
8	-5.62467	-6.07464	-5.62467	00000000	00000000	00000000
9	-5.35112	-5.35112	-5.35112	00000000	00000000	00000000
10	-5.49328	-5.49328	-5.05381	00000000	00000000	00000000
11	-5.76595	-6.22723	-5.30468	00000000	00000000	00000000
12	-5.9214	-5.9214	-5.48277	00000000	00000000	00000000
13	-6.06243	-6.06243	-5.16429	00000000	00000000	00000000
14	-6.22723	-5.76595	-5.30468	00000000	00000000	00000000
15	-6.07464	-5.62467	-5.62467	00000000	00000000	00000000
16	-5.76595	-6.22723	-5.30468	00000000	00000000	00000000
17	-5.49328	-5.49328	-5.05381	00000000	00000000	00000000
18	-5.63408	-5.63408	-4.73262	00000000	00000000	00000000
19	-5.9053	-6.37772	-4.96045	00000000	00000000	00000000
20	-6.06243	-6.06243	-5.16429	00000000	00000000	00000000
21	-6.201	-6.201	-4.823	00000000	00000000	00000000
22	-6.37772	-5.9053	-4.96045	00000000	00000000	00000000
23	-6.22723	-5.76595	-5.30468	00000000	00000000	00000000
24	-5.62467	-5.62467	-6.07464	00000000	00000000	00000000
25	-5.76595	-5.30468	-6.22723	00000000	00000000	00000000

Buffer Format

Format

```
float3 Position0;
xint pad[3];
```

Buffers

Slot	Buffer	Stride	Offset	Divisor
0	**Buffer 134**	24	0	0

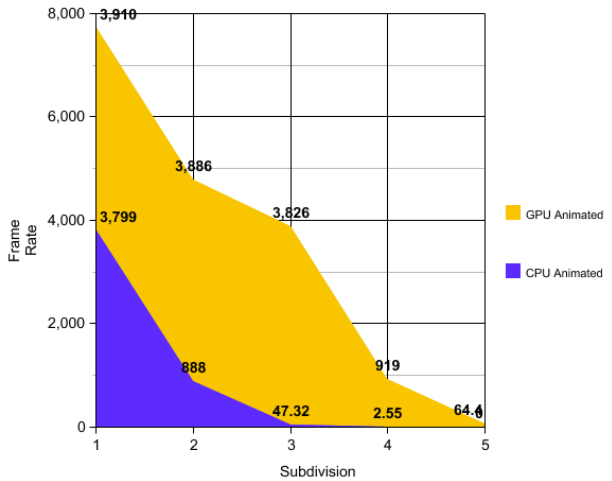
Is this what you expected? Why or Why Not?

As you can see, we only have a single buffer on the GPU with a 3 float vector.

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	Frame Rate Scene 5	Frame Rate Scene 6
1	3799	3910
2	888	3886
3	47.32	3826
4	2.55	919
5	~	64.4



What we observe in the graph is that the rate at which the CPU animated scenes frame rate decreases is much faster than the GPU animated scene.

Scene 5 hits a minimum frame rate at three subdivisions where as scene 6 sits comfortably at 60 frames per second at subdivision level 5 with 38400000 faces.