Real Time Rendering & 3D Games Programming Assignment 1

s3405176 Ke Yi Ren
s3434180 William Field

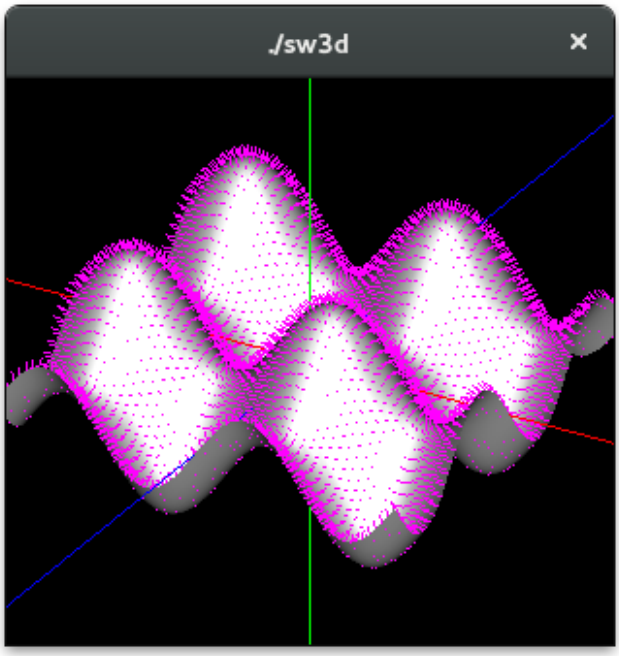


Table of Content

Introduction:

Aim: This assignment is intended to introduce students to graphics performance, measurement, benchmarking and optimisation, primarily through the use of vertex buffer objects (VBOs) and associated APIs. SDL is to be used for the UI and event handling instead of GLUT, although not for performance reasons.

Method: In order to achieve a consistent benchmark, default settings to compare frame rate and frame time were needed. The following options were used;

Drawing Mode: Filled
Animation: Off
Number of Sine waves: 1
Screen Dimension: 1920 * 1080
Multi-view: False
Lights: Disabled

From this point we would change one option, check the frame time & frame rate, up the tessellation, and swap between immediate mode and Vertex Buffer Objects (VBOs). E.g. Frame time and rate would be recorded for, 8, 16, 32, 64, 128, 256, 512, 1024 tessellations and repeat with VBO for comparable results

Since Lighting is quite taxing on the system and performance, we did four different experiments to cover most of the outcomes, Disabled light, one light, 5 lights and all nine lights and recorded the result.

Observation: What we observed was that for most cases VBO out performed immediate mode. We found an anomaly while testing animation, and found that VBO performed significantly worse than normal, similar to immediate mode.

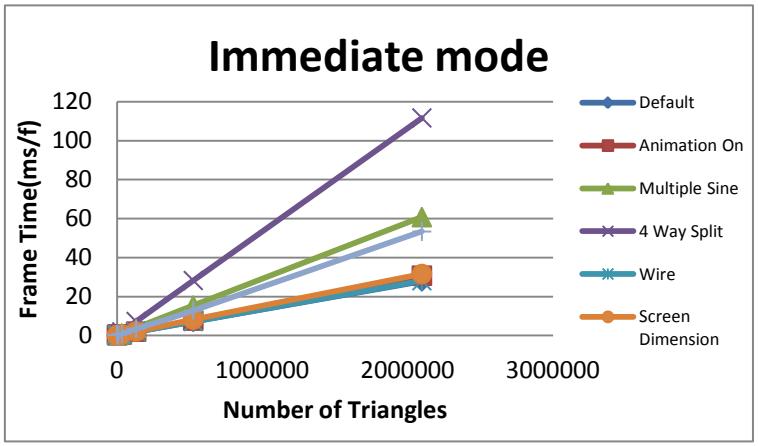
Results: The first lot of results show, in general, that VBO performs better overall (Appendix A.1). The following discussion covers the range of results we attained through different options; see Appendix B for all results.

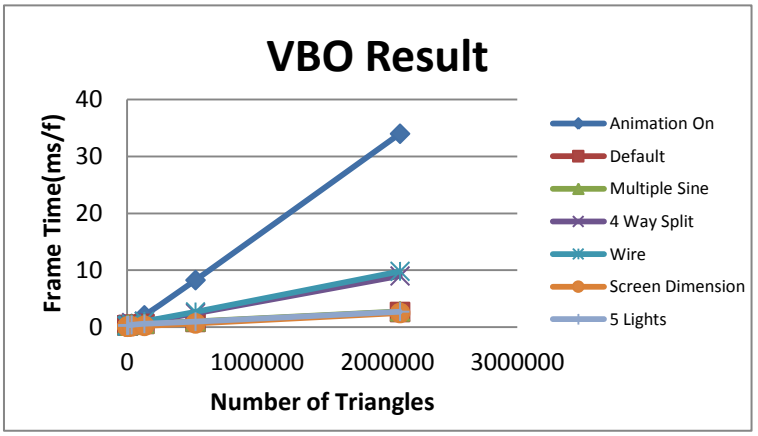
Discussion:

In order to achieve the best possible results for frame time and rate, programmers aim to minimise bottlenecking. Some of the potential bottlenecks are CPU performance, Bus bandwidth, Vertex performance and fragment performance. To increase performance we implemented Vertex Buffer Objects which allows processing of data on the GPU and frees up the CPU. This is reflective through our results as we found a dramatic performance improvement using VBOs especially while rendering grids of 256 tessellations or greater; for lighting and 1024 tessellation, VBOs performed twenty times better than the Immediate mode.

Default Settings

As shown by our generalised results VBO performs significantly better since, immediate mode needs to calculate and draw the wave every frame, while VBO calculates and stores the data, and redraw every frame. As expected with the increasing tessellation, the performance for both rendering modes drop due to more pixels being processed and drawn. The tessellation is the most hindering factor for graphical performance and our results show that the 128/256 mark the two modes times start to separate.





Filled vs. Wireframe:

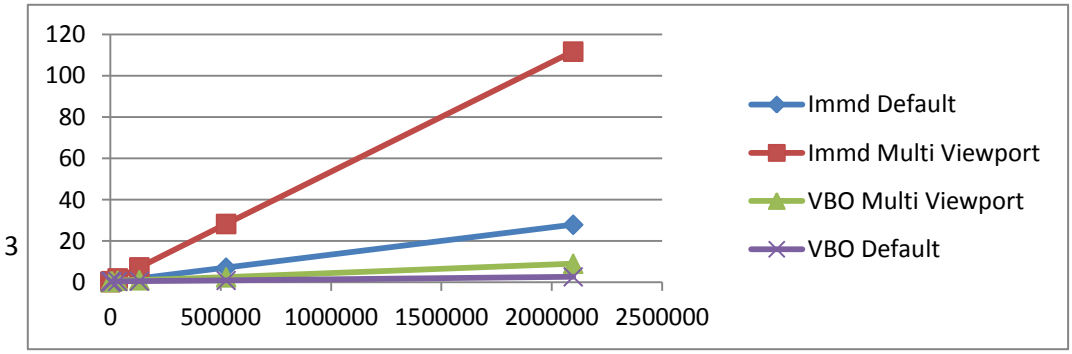
We found that for Wireframe mode out performs Filled mode when rendering less than 32,000 triangles, and for immediate mode they take similar time for 1024 tessellation, ~28ms/f. This is expected since, for smaller tessellations there is visual difference between wired and filled, but higher tessellations they look identical from afar since the polygons are dense.

However for VBO rendering took 3 times longer to draw Wireframe for 1024 tessellation, 2.66ms/f compared to 9.816ms/f (Appendix B)

Multi-view port vs. Single

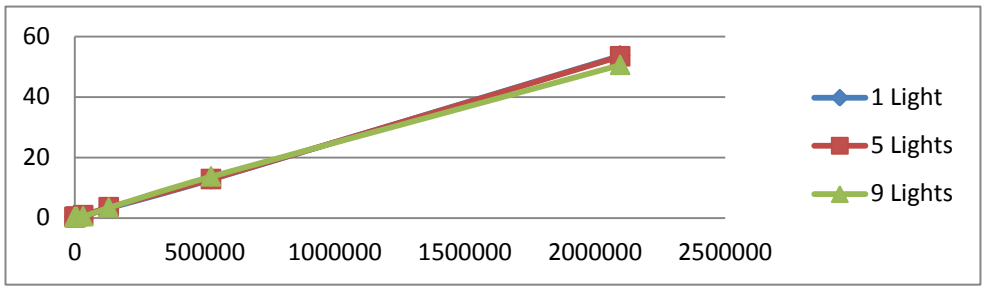
We predicted that the multi-view mode would take around four times longer to render; since there are 4 graphs, but found VBO takes approximately 3.4 longer (Appendix B) to draw the graphs since VBO only renders everything on the screen whilst immediate mode takes four times longer since it draws everything. We deduced this through playing with zoom, where we found that the further you zoom into the page with VBO, the better the performance: For immediate mode the result was steady, meaning that the immediate mode draws everything in the frame, then crops out what isn't seen, while VBO renders everything in the scene.

Multi-view was the option that affected immediate mode the most, and resulted in 20 times worse compared to VBO. This is due to how it is drawn; we used two for loops, nested, to draw the immediate mode sine wave and one loop for VBO. The cost of this relation is n^2 vs. n; and when we need to draw 4 times the number of graphs, it becomes $4n^2$ vs. 4n.



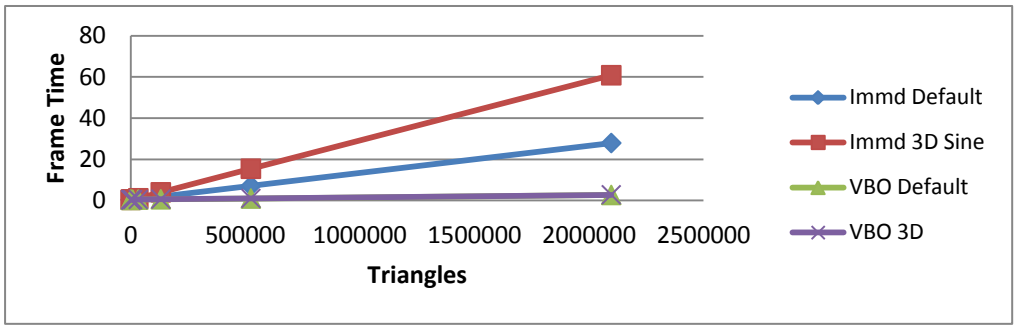
Lighting Enabled

As previously mention, we initially believed that VBO would perform better with Lighting enable (Appendix B).We found that disabling GL_LIGHT_MODEL_TWO_SIDE drastically improved performance since half the number of side would be lit, compensated with single side lighting. Unexpectedly, as seen in our results (Appendix B & Below), as the number of lights increased the performance remained the same. We didn't predict that one light, five lights and nine lights, the frame time would be relatively similar for the respective mode for all tessellation. We thought that since there are more light sources and rays being cast and more calculations for each vertex, the frame time would increase but found that they were the same.



2D Sine vs. 3D Sine

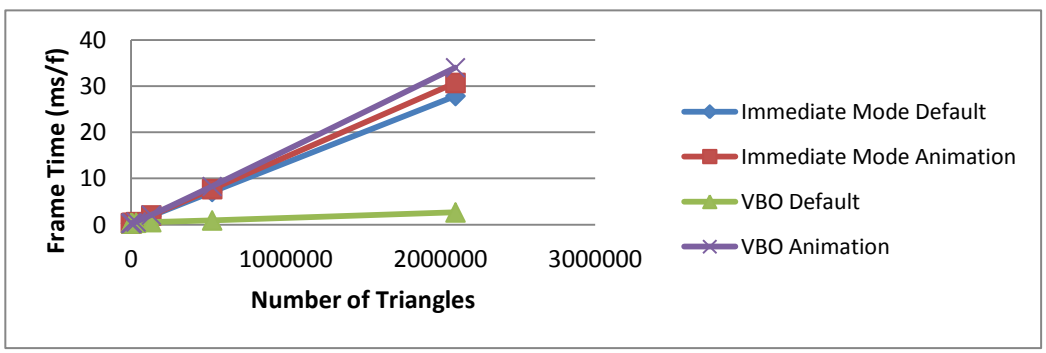
The difference in the two graphs is the 3D sine requires twice as many sinf() and cosf() calculations, hence we thought that the graphs would take around twice as long to draw, due to sinf() and cosf() being expensive calculations. Looking below you can see that the immediate mode took approximately twice as long to draw the 3D sine the calculation are expensive to do ever frame. The VBO mode took the same time to do both modes since we do the calculation once, and store it, then continue to redraw it.



Animation

We found from our results that animation is the most expensive operation for VBO and that it performed similar to immediate mode. The animation is expensive for both modes since all points need to be calculated and redrawn every frame (Appendix B & Below), preventing storing for VBO since next frame the values would need to be calculated.

However we concluded that the code may not be working correctly. After polling other students we found their result show VBO out performs immediate.



Screen Dimension: 640 * 400

One interesting result was the performance difference between the two rendering mode when the camera was at maximum zoom (out) we found for low tessellations that both modes where getting 10k+ frames per second. We guessed this is due to the drawing the lines would be small and quick; for the large tessellations the time taken to draw comes from the number of point and the time taken to draw a line is minuscule, hence both screen sizes are similar in time to default screen dimensions (Appendix B)

As part of the investigation you are specifically required to find how many lit shaded triangles (polygons) can be drawn at a 'real-time frame rate' of around 30 fps, i.e. in 1/30th of a second using the fastest rendering approach.

The results attained allow us to predict that in order to maintain 30fps we would draw
VBO, Lights = 1;
365fps = 1024 tess, 2097152 triangles
2.74 (ms/f) / 2097152 = (1/30) / x ~= 1/ x > 2.09m,
1.307 x 10-6 = (33.3 ms/f)/x
X = 25487285 triangles //In theory this should be it, but not so much in prac
Approximately tessellations 3269

You are also to specifically find how long a wave of a million polygons takes to render.

As shown by our results (Appendix B), drawing 1,000,000 polygons lie between 512 tessellation and 1024;
1000000 = ((tess)*(tess)) /2, which tess comes out at 707~

For Immediate Mode, Default settings
27.861/2097152 = x/1000000 ~= 13.285 ms/f

For VBO, Default settings
2.66/ 2097152 = x/1000000 ~=1.268 ms/f

Conclusion:

As proven through the discussion, we were able to achieve our aim to measure, benchmark and optimise our code through the use of Vertex Buffer Objects and Vertex Arrays. Furthermore, we could apply what we found to accurately predict how long it would take to draw one million polygons and how many polygons would we need to draw to maintain 30 fps.

Immediate mode

		Default		Animation On		Lights 1		Lights 5		Lights 9		Multiple Sine		Screen Dimension		4 Way Split		Wire	
Grid Tess	Triangle	Frame rate (f/s)	Frame time(ms/f)	Frame rate (f/s)	Frame time(ms/f)	Frame rate (f/s)	Frame time(ms/f)	Frame rate (f/s)	Frame time(ms/f)	Frame rate (f/s)	Frame time(ms/f)	Frame rate (f/s)	Frame time(ms/f)	Frame rate (f/s)	Frame time(ms/f)	Frame rate (f/s)	Frame time(ms/f)	Frame rate (f/s)	Frame time(ms/f)
8	128	3105	0.322	3106	0.322	3525	0.284	3520	0.285	3285	0.304	2904	0.344	**14216**	**0.0703**	4063	0.245	4529	0.221
16	512	3091	0.323	3058	0.327	3395	0.294	3429	0.291	3280	0.305	2850	0.351	13186	0.0758	3970	0.252	4168	0.24
32	2048	2975	0.336	2976	0.336	3256	0.307	3273	0.305	3056	0.327	2693	0.371	9734	0.103	3769	0.265	3801	0.263
64	8192	2762	0.362	2767	0.361	2954	0.339	3033	0.33	2700	0.37	2501	0.399	5308	0.188	1804	0.554	2985	0.335
128	32768	1953	0.511	1825	0.548	1864	0.536	1137	0.879	1812	0.552	959	1.043	1789	0.559	528	1.892	1915	0.522
256	131072	538	1.856	500	1.998	305	3.278	284	3.51	291	3.436	255	3.922	480	2.081	138	7.23	527	1.894
512	524288	141	7.092	129	7.7	78.4	12.759	78	12.821	72.9	13.712	64	15.39	121	8.23	35.4	28.25	140	7.135
1024	2097152	35.9	27.861	32.6	30.697	18.6	53.632	18.738	53.368	19.78	50.55	16.5	60.765	31.68	31.56	8.96	111.667	35.6	28.111

VBO

		Default		Animation On		Lights 1		Lights 5		Lights 9		Multiple Sine		Screen Dimension		4 Way Split		Wire	
Grid Tess	Triangle	Frame rate (f/s)	Frame time(ms/f)	Frame rate (f/s)	Frame time(ms/f)	Frame rate (f/s)	Frame time(ms/f)	Frame rate (f/s)	Frame time(ms/f)	Frame rate (f/s)	Frame time(ms/f)	Frame rate (f/s)	Frame time(ms/f)	Frame rate (f/s)	Frame time(ms/f)	Frame rate (f/s)	Frame time(ms/f)	Frame rate (f/s)	Frame time(ms/f)
8	128	3108	0.322	3456	0.289	3497	0.286	3503	0.285	3365	0.297	2898	0.345	**13715**	**0.073**	4040	0.248	4531	0.221
16	512	3089	0.324	3324	0.301	3399	0.294	3476	0.287	3284	0.304	2841	0.352	12632	0.079	3974	0.252	4151	0.241
32	2048	3020	0.331	3193	0.313	3356	0.298	3272	0.306	3172	0.315	2744	0.364	12536	0.08	3314	0.302	3779	0.265
64	8192	2752	0.363	2911	0.343	2976	0.336	2977	0.336	2930	0.341	2543	0.393	10823	0.0923	2330	0.429	3033	0.33
128	32768	2369	0.422	1723	0.580	2529	0.395	2589	0.386	2520	0.397	2155	0.464	7205	0.139	1236	0.809	2016	0.496
256	131072	1772	0.564	482	2.072	1624	0.616	1667	0.599	1745	0.573	1567	0.638	4378	0.228	965	1.036	963	1.038
512	524288	1087	0.919	121	8.267	1029	0.972	1028	0.972	1109	0.901	982	1.018	1507	0.663	410	2.435	373	2.679
1024	2097152	375	2.66	29.7	34	365	2.74	367	2.72	376	2.659	360	2.773	401	2.49	111	9	101	9.813