Assignment 1 report

Aurora Li Min de Freitas Wang - s3529137
Pedro Henrique Morais Pereira - s3495839

1 INTRODUCTION

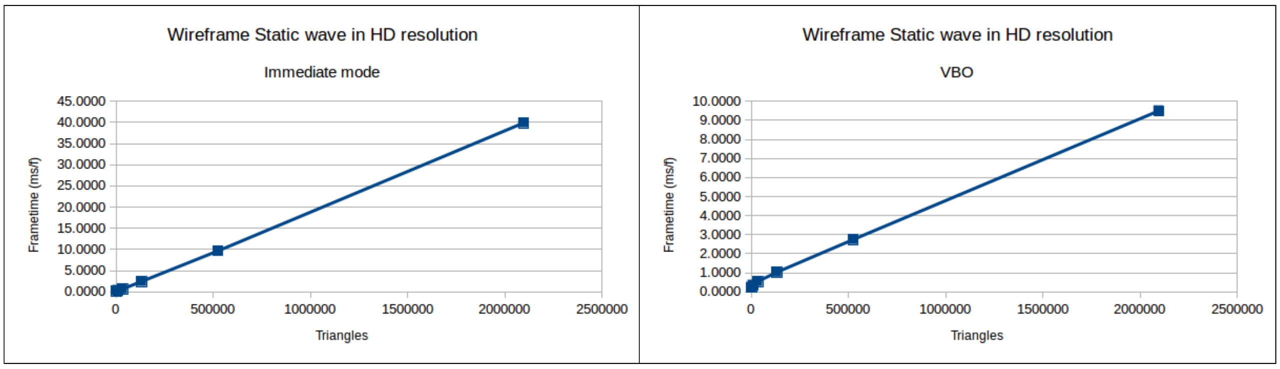
The key to obtain success on Computer Graphics area is performance. Users will always want faster applications with better images. Thus, one of the main objectives of Real-Time Rendering and 3D Games Programming course is to teach students how to measure performance e how they can improve it using better rendering techniques.

2 OBJECTIVES

To discuss and analyse the performance data extracted from the modified and extended code that renders a sine wave using SDL and VBOs along with various interactive controls made by this group.

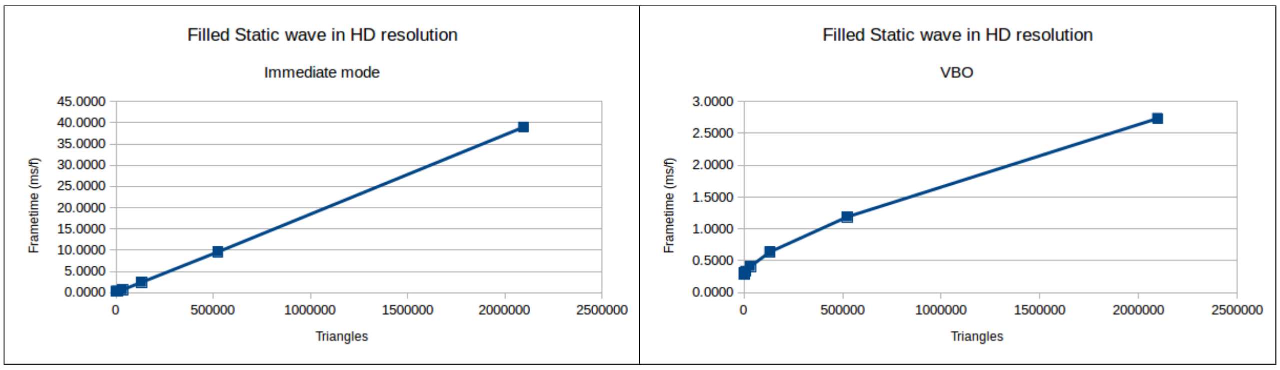
3 RESULTS

The results below were made with information collected directly from our program while running on Sutherland machines. To make the comparison between the data more clear we choose to focus on Triangles X Frametime graphs. However, the collected information also includes Tesselation (that can be inferred from Triangles) and Framerate (that can be inferred from Frametime). Therefore, this complementary data can be found on this assignment folder.



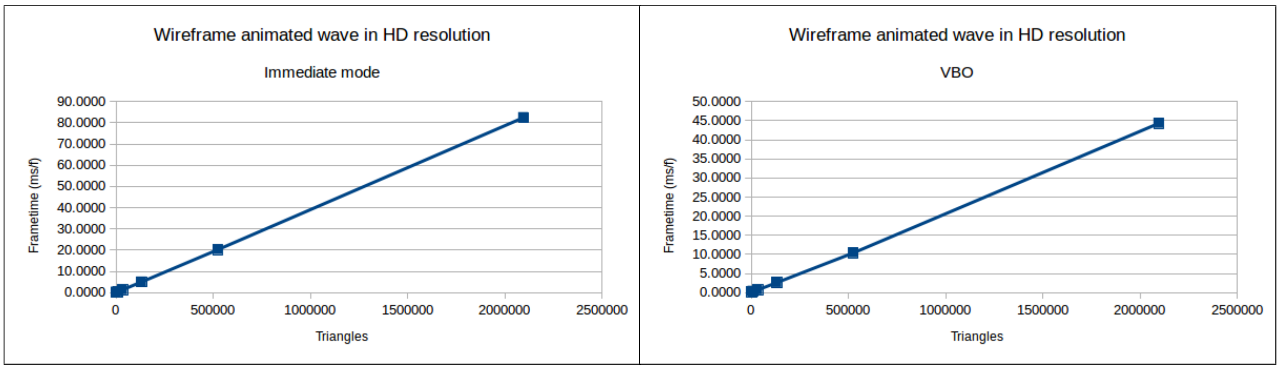
(a) (b)

Figure 1



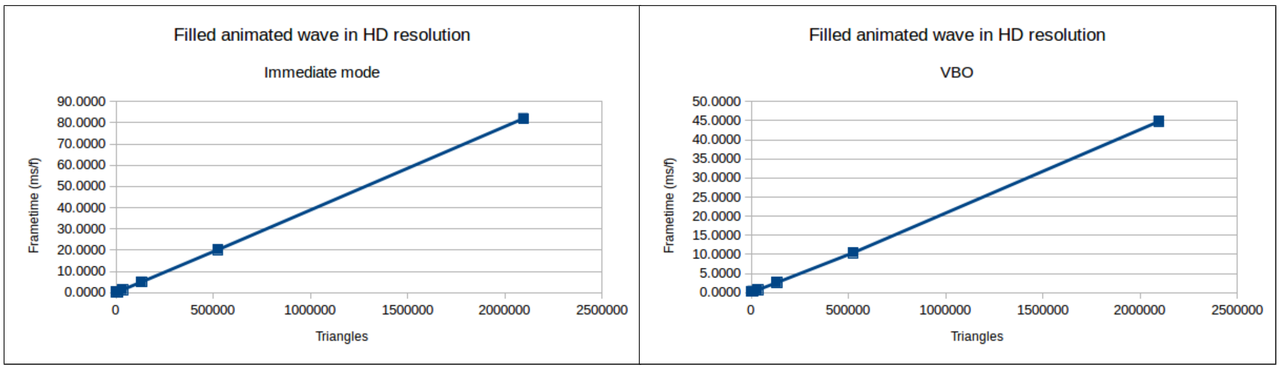
(a) (b)

Figure 2



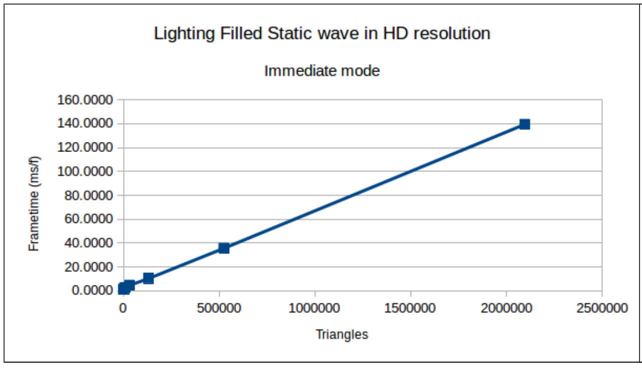
(a) (b)

Figure 3

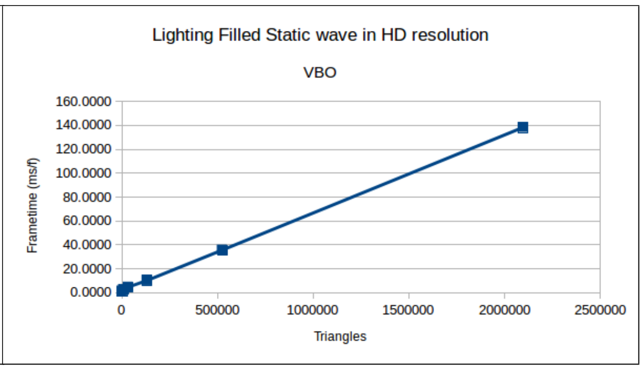


(a) (b)

Figure 4

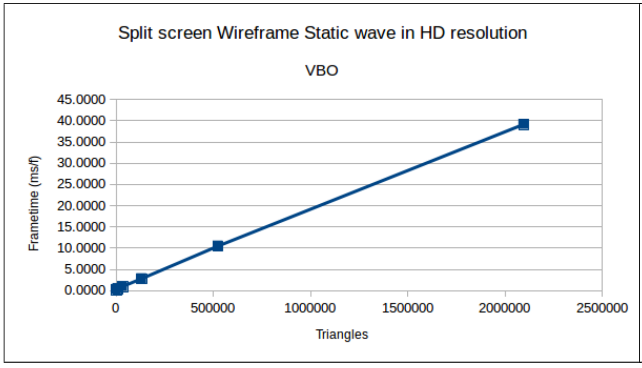


(a)

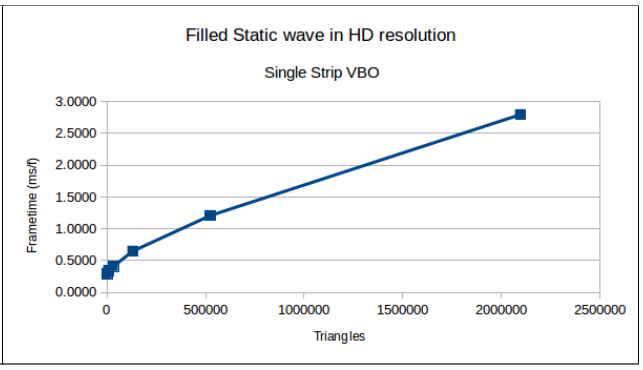


(b)

Figure 5



(a)



(b)

Figure 6

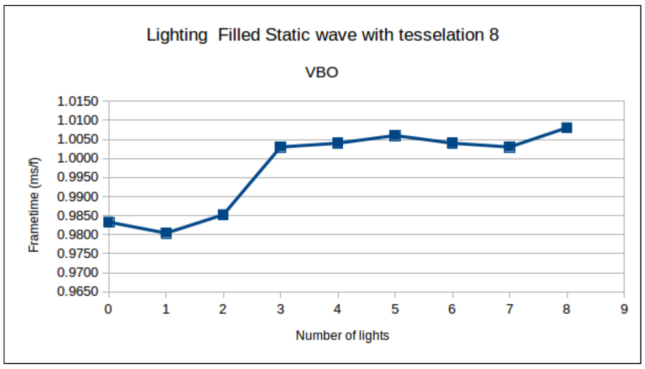


Figure 7

4 Analysis

[!htb] By analysing the above data is possible to notice a direct correlation between Triangles and Frameratime: If the program is rendering a wave with more Triangles more Frametime will be needed. This relation tend to be linear, as shown on Figures 1-6. In other words, the GPU takes about twice the time to render two triangles as it would take to render one.

However, when it comes to Frametime it's not only a direct relation of the analysed change. As the Frametime changes due to the display function, the rest of the code remains the same. Therefore it will not achieve the perfect linear scenarios, especially in cases that display is fast, i.e. Tesselation 8.

Besides that, it’s also possible to infer that the relation between Tesselation and Frametime time is quadratic as: $number of triangles = 2 * tesselation^2$.

Moreover, the performance is highly affected by Vertex Buffer Objects (VBOs) as can be seen comparing graphs ’a’ and ’b’ in each image from Figure 1 to Figure 4. The explanation for this is that sending vertices immediately to the GPU every time creates a bottleneck between client (program) and server (graphic card). Thus, sending all the information together to VBO is faster and prevents the bottleneck by saving the information needed on a high-performance graphics memory on the server side.

It is also possible to notice differences on the performance using VBO by changing rendering mode from Wireframe to Filled. To render filled polygons is less costly than to render Wireframe polygons. Then, most of the NVIDIA Geforce GPUs prioritise filled rendering over wireframe rendering as it’s simpler to do and used more commonly. However, when it comes to changing from Wireframe to Filled on immediate mode the performance remain the same. This happens because the time needed to render both wireframe and filled waves is lower than the time spent by the bottleneck explained on the previous paragraph.

When it comes to animated waves, the cost of rendering a filled one and a wireframe one is the same due to the fact that the cost difference between rendering them is lower than the cost of having to constantly update and render all vertices. As it was expected, all animated waves performed worst than static ones. In addition, it is noteworthy that a animated wave using VBO took half of the cost spent by a animated wave using immediate mode. On this case, as the VBO was constantly updated it acted like a Vertex Array. Then, still less costly to send vertex by vertex than to send all the vertices at a time.

Another aspect that highly influences the performance is turning one light on. Both of Figure 2 and Figure 5 data was collected while rendering a Filled Static Wave. But lighting Figure 5 wave caused a drastic performance fall comparing to the other one performance. The cause of it is that to calculate normals and apply light to the scene is really costly. Actually, turning the light mechanism on is so costly that overcome the client-server bottleneck cost, making both immediate and VBO modes perform the same in this case.

However, by looking to Figure 7 is possible to notice that the cost to turn the lighting mechanism on remains averagely the same regardless of the number of lights used. As said on the previous paragraph, most of the time is used to calculate normal vectors and apply the light. Adding or removing lights will not change this.

Another way to comparate the data is to check the slope of the linear equation, between the last two point, to roughly predict the cost of adding extra triangle:

Slope (* 1000)					
	Static			Animated	
	Light filled	Filled	Wireframe	Filled	Wireframe
Immediate	6.61	1.87	1.92	3.93	3.95
VBO	6.53	0.10	0.43	2.18	2.16
VBO is X% faster	1.23	1770.00	348.83	80.28	82.87

As it was taken from the graph $Frametime \times Triangles$, it is possible to compare two cases and say that extras triangles costs X% more, where cost is the Frametime and $X(A,B) = (slope(A)/slope(B) - 1) * 100$.

Also, the cost to render a split screen with different visions of the same static wireframe wave is the same as rendering just the standard view of a static wireframe wave. In addition, around 524288 lit shaded triangles can be drawn at a ’real-time frame rate’ of around 30 fps. And finally, it takes 1.304 ms to render a million triangles on the filled static VBO mode.