Characterization and Reverse-engineering Assignment - EEC 277

Ahmed H. Mahmoud and Mohamed Bana

7 February 2017

1 Introduction

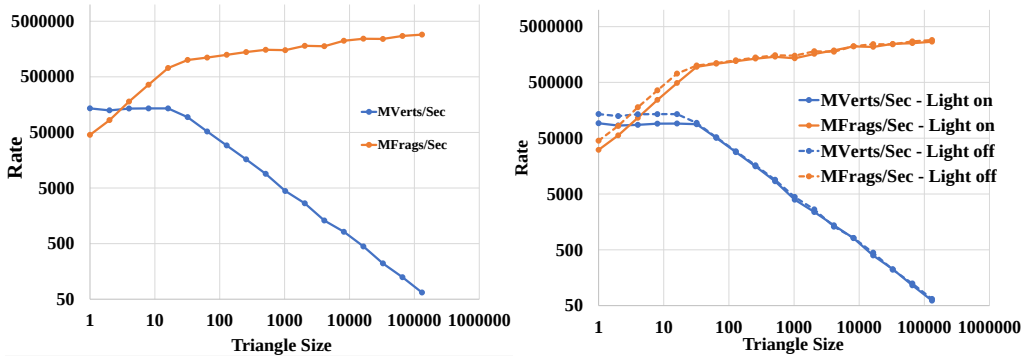
This report is divided into two parts; performance analysis of the graphics card and reverse engineering to charactrize undocumneted features of the GPU. In the first part, we are going to characterize the performance of the two GPU using `wesBench` benchmark. The target here is to find the crossover point between the geometry/vertex stage and fragment/rasterization stage for different scenarios. Broadly speaking, the graphics pipeline overall performance is a function of the slowest of these two stages. It is well known that the geometry stage favors large primitive triangles since the speed of this stage if dependent on the operations-per-triangle. In contrast, rasterization stage favors small primitive triangles since a large triangle would require more fill operations [2]. All the experiments presented in the report are done on NVIDIA GeForce GT 610 GPU on a Windows 7 machine with four-core Intel(R) Xeon(R) CPU of 3.7GHz and 32.0GB RAM.

ahmed: TODO: talk about the OpenGL calls in the code i.e., "Why do we use glDrawXXX calls rather than glBegin, glVertex, glEnd calls to draw geometry?", "What is wesBench actually drawing?" and "How does wesBench make sure that all geometry that it draws will pass through the entire pipeline?"

2 Part A

2.1 Rate Vs. Fill Rate:

The objective of first experiment is to find the crossover point for the unlit untexture triangles between the fill rate and geometry rate. This is done by testing both rates for different triangle sizes. The results are shown in Figure 1(a) on a log-log scale, where the fill rate (MFrags/Sec) increases as expected for small triangles and geometry rate (MVerts/Sec) decreases. The crossover point is between triangle area between 2 and 4 pixels. We notice that for triangle of size between 2 to 16 pixels the geometry rate is almost constant. Even though the geometry rate is the highest but this could be due to more efficient use of caching. Since the triangles are of small size, and due to spatial locality, more triangle can be fetched and put into the cache. On the other side, the rapid change on the fill rate starts to slow down at triangle size greater than 16 pixels. This could be due to the fact the geometry stage is not sending enough work for the rasterization stage. So even the overall fill rate increases as the triangle size increases, but the slope of the line is not the same overall.



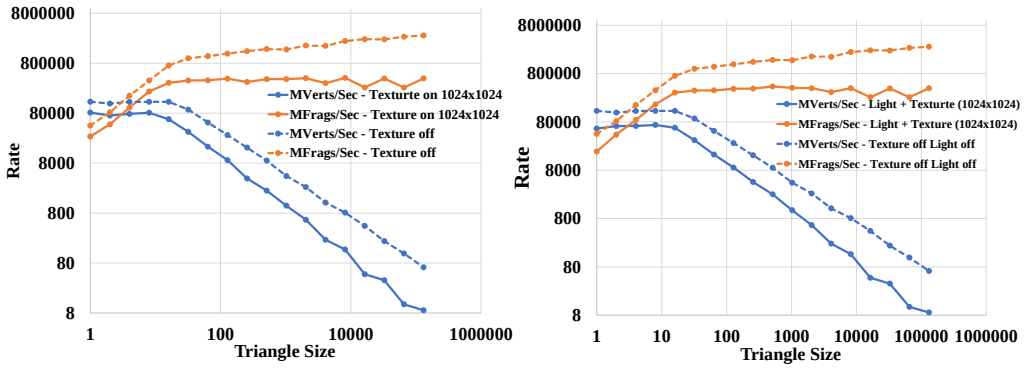
(a) Default (b) Lit Triangles

Figure 1: millions of vertices and millions of fragments

2.2 Geometry Rate Vs. Fill Rate on Lit Triangles:

The previous test was done on unlit, untexture triangles which was the default setting. Now we turn on the light on the triangles and do the same test and

compare the results with the unlit triangles. The results are shown in Figure 1(b) where the crossover point is almost the same (between triangle size of 2 and 4 pixels). Additionally, we notice that the upper portion of the graph (towards triangle of size greater than 32 pixels) is identical i.e., the dotted and the solid lines overlaps for fill and geometry rates. For the lower part of the graph, it is understood that the geometry will decrease since the lighting is being processed at this stage and thus more work need to done for computing each vertex. This will directly affect the fill rate since less work is sent to the rasterization stage and thus a decline in absolute fill rate. As the triangles get bigger in size, less number of vertices need to be processed in the geometry stage. Thus, even though more work need to be done per vertex, we notice that the unlit triangle rate is almost identical to the lit triangles. Same thing applies for the fill rate; since the geometry stage now is able to send enough work for the rasterization stage, the fill rate is identical for both lit and unlit triangle.



(a) Textured Triangles (b) Lit Textured Triangles

Figure 2: millions of vertices and millions of fragments

2.3 Geometry Rate Vs. Fill Rate on Textured Triangles:

Now we turn on testing the textured triangles. Following the same methodology as in Section 2.2, we add textures to the triangles of size 1024×1024 and vary the triangle size and record the fill and geometry. We tested first with texture of smaller size (128×128), but the graph we got was identical

to the untextured one, so we discarded. We assume that this happened due to minimal workload required with small texture size.

Figure 2(a), compare the fill and geometry rate for textured and untextured triangles. The crossover point is almost the same. Having the same crossover point for the three different scenario tested so far suggests that the graphics card is optimized to have always such a crossover points. The graph also shows an decline in the performance in the textured triangles for both the geometry and fill rate. For the geometry stage, the decline in the number of vertices processed per second is due to the necessary transformation done per vertex (applying the *projector function*[1]). The graph in Figure 1(b) and in 2(a) suggest that per-vertex-operation for the lighting is much cheaper for per-vertex-operation for texture which is indicated by the absolute value of geometry rate at certain triangle size. Additionally, the graph shows a divergence in the geometry rate between the textured and untextured. At first glance, this was surprising and we expected a similar behavior as in lit triangles. As the number of triangles decreased with size increases, then less number of vertices needed to be processed. However, we think that this divergence could be due to caching. Since we are using a relatively large texture (1024×1024) and for a big triangles we would required fetching different parts of the texture image that may not be fetched all together into the cache (less spatial locality).

For rasterization stage, the fill rate decline since the texture operation is more expensive on fragments. During the rasterization, the *corresponder function* (matrix transformation) is applied per fragment and texture value per fragment is interpolated [1]. Both could be the reason for the decline in the fill rate. We notice the same divergence happens in fill rate. We assume this is because the geometry stage does not send enough work for the ratserization as the triangle size increase.

2.4 Geometry Rate Vs. Fill Rate on Lit, Textured Triangles:

Here we use lit, textured triangles and compare the same performance metrics with the unlit, untextured triangles. Figure 2(b) shows that the crossover point did not change which confirm our hypothesis; that the graphics card is optimized for this value. For large triangles (size ≥ 32) the lighting does not have any impact on both the geometry or the fill rates, then the decline in

the performance is purely due to the texture processing. For smaller triangle size, both lighting and texturing contribute in declining

3 Part B

3.1 blah

References

[1] Tomas Akenine-Möller, Eric Haines, and Naty Hoffman. *Real-time rendering*. CRC Press, third edition, 2008.

[2] Edward W Bethel. *Using wesBench to Study the Rendering Performance of Graphics Processing Units*. Jan 2010.