Graphics Characterization Analysis Report

Yangzihao Wang

March 15, 2012

We run the tests on a PC with Intel Core i5 3.20GHz CPU and a NVIDIA GeForce GTX560 graphics card with 1 GB of memory. By varying the triangle size from 1.0 to 131072.0 (128K) in pixels, the geometry rate keeps droping while the fill rate keeps increasing in a much more faster pace. The reason for this can be explained as below: Following with the increasing of the triangle size, the total number of vertices to process in one frame will go down, thus the geometry rate will increase when triangle area is still small enough (In our experiment when triangle area is smaller than 32 pixels). Note the triangle area doubles every time, which in our case means that the number of fragments doubles every time. However, the geometry rate does not increase in the same pace. The reason for this is that the relationship between vertex number and triangle area is not linear, but as the following equation:

$$N_{vertices} \propto \frac{1}{\sqrt{2 \cdot A_{triangle}}}$$

When triangle area becomes large enough, the number of fragments becomes large enough to make fragment processing take most of the computing resource, thus will cause the geometry rate to drop. The fill rate increases from the beginning, however, the slope of the curve for fill rate becomes smaller as the triangle area becomes larger than . Two reasons can explain this: 1) The program is reaching the upper limit of fragment computing capability; 2) The fragment number limitation set in program prevents the increment of the fill rate. On this NVIDIA GeForce GTX560 GPU, the crossover point occurs when triangle area is between 2^7 pixels and 2^8 pixels. The crossover point represents the balance between geometry and rasterization. Before it the geometry processing has more work load and after it the fragment processing has more work load.

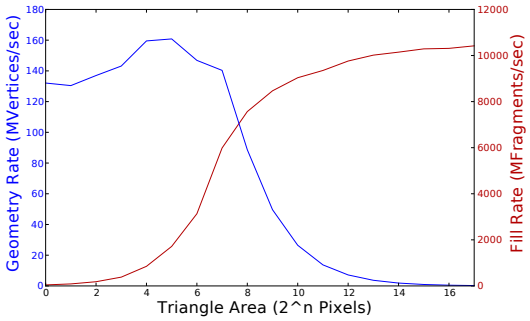
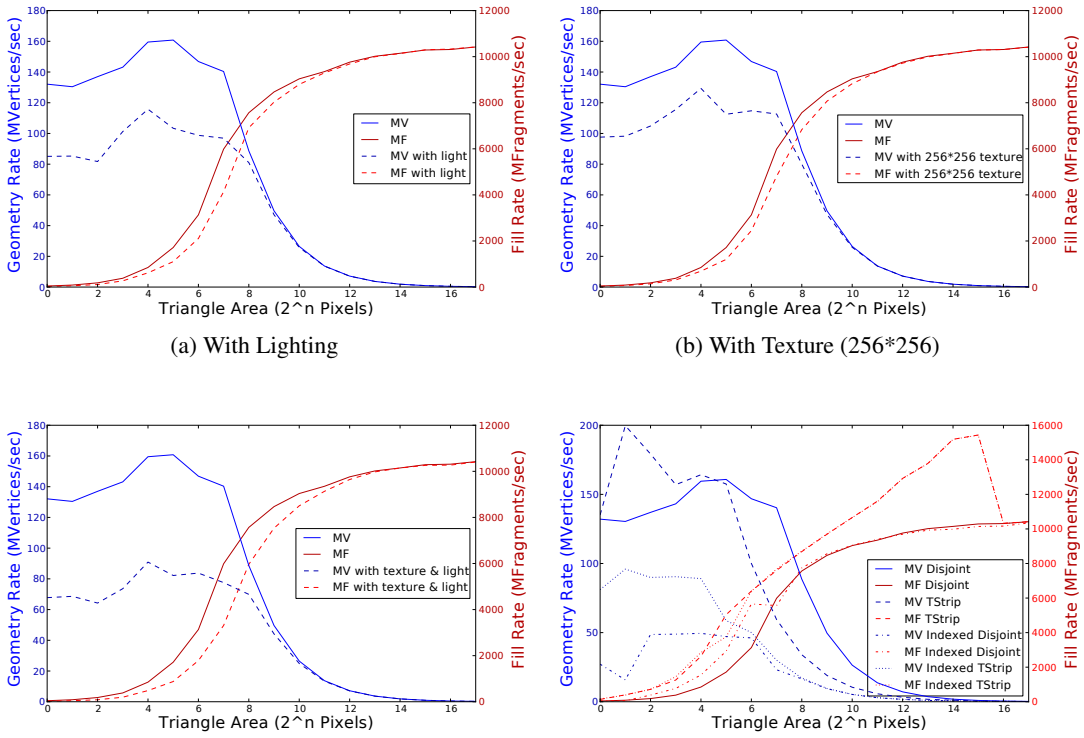


Figure 1: Relationship between geometry rate, fill rate and triangle area

Adding light and texture will affect both the geometry rate and the fill rate. It will cause the crossover point move towards the geometry side. This means the work load of geometry processing will increase faster with light, texture or both of them on, although texture fetching will affect rasterization more heavily than geometry and lighting will affect geometry more heavily than rasterization. It is because texture fecthing is a per-fragment operation while lighting calculation is a per-vertex operation. As for different primitive types, in general triangle strip mode (both indexed and non-indexed) will have larger geometry rate and fill rate than disjoint triangle mode. Also, triangle strip mode and indexed triangle mode will both add more work on geometry side so the crossover point will move towards the geometry side when using these modes to rendering.



(a) With Lighting

(b) With Texture (256*256)

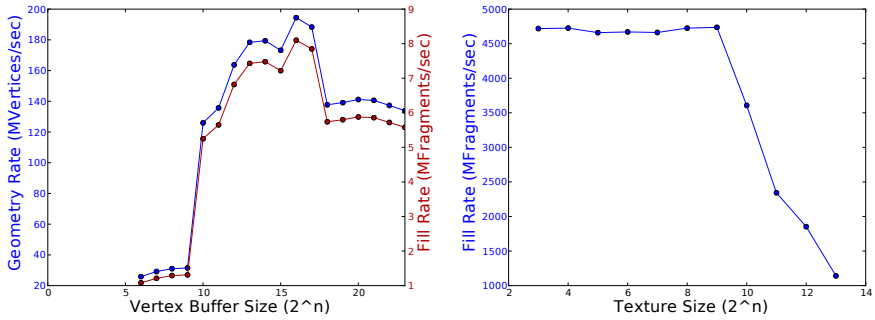
(c) With Both Lighting and Texture (256*256)

(d) Different Triangle Types

Figure 2: Relationship between geometry rate, fill rate and triangle area under different conditions

In the ideal situation, we want both the geometry rate and fill rate to increase with the increasing of vertex buffer size, which is propotional to the batch size. This is true when batch size is small. In that situation the performance is limited by the CPU's ability to send work to the GPU ("interface-limited"). When the program is still bounded by interface-limitation. The geometry/fill rate will both increase according to the increasing of the vertex buffer size. From Figure 3 we note that when the vertex buffer size is larger than 65536 (2^{16}), both the geometry and fill rate starts to drop. This means at this batch size, the system stops being interface-limited and starts being limited by the GPU. We note that on this batch size (65536), the geometry bandwidth between CPU and GPU is 194.48M Vertices/Second.

In our experiment, we can see before a threshold value, the fill rate is not affected by the texture size. However, when texture size is larger than 512*512, the fill rate starts to drop according to the increasing of texture size. The reason for this is related with texture data fetching from memory. For textures with small size, they can stay in memory cache after being fetched once and be reused afterwards. While for any



(a) Geometry rate, fill rate VS vertex buffer size (b) Fill rate VS texture size

Figure 3: Relationship between Vertex Buffer Size, Texture Size and Geometry Rate/Fill Rate.

texture with large size (larger than 512*512), memory cache cannot hold it, so every time the program does a texture fetching, it needs to fetch the texture data from memory and cost texture bandwidth. This memory access will reduce the performance, in this case, the fill rate.

On this GPU, the function of fill rate and texture size is a piecewise function, consisting of one constant function and one linear function generated using linear regression:

$$F = \begin{cases} 4.7 & T_{size} \leq 512 \\ -0.89 \log T_{size} + 12.57 & otherwise. \end{cases}$$

Where T_{size} is the side length of a square texture in pixel and F is the fill rate of the program in gigabyte fragments per second.