EEC 277 Project 2

Phil Monroe and Kramer Straube

Feb. 6, 2012

1 Introduction

In this report, we will describe the series of tests we performed to characterize various aspects of the graphics card, namely the Evga NVidia 9500 GT. Some basic statistics about this card are that the GPU contains 4 multiprocessors with 8 cores each. Since this GPU supports NVidia CUDA compute capability version 1.1, it can support up to 768 threads per multiprocessor. In section 2, we will cover how the graphical load of the GPU is affected by different variables. In section 3, we will examine the statistics of the compute capabilities of this particular graphics card.

2 Part A

In this section, we will discuss the way in which different input data sets and settings can affect the graphical capabilities of the graphics card using OpenGL and the instructional version of WesBench.

2.1 Triangle Area

This test iterates through various triangle areas to determine how the fill rate and vertex rate change. The data, as shown in Figure 1, shows how this graphics card handles the change in triangle area size. The fill rate flattens out as it gets larger, which is a sign that the rasterizer is getting maxed out. The vertex rate stays steady since primitive assembly is maxed out. It then declines as the area increases because there are fewer vertices overall. The crossover where the rasterizer becomes the bottleneck over primitive assembly is at 64px.
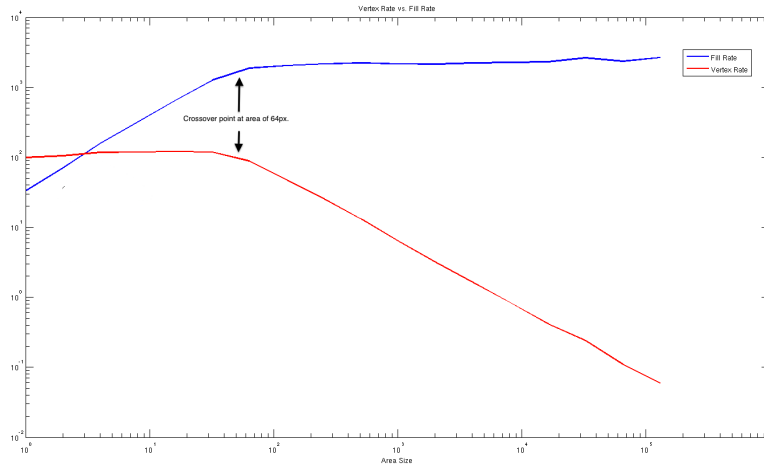

Figure 1: Vertex rate and fill rate for various triangle areas.

2.2 Lighting and Texture

This test shows the affects of enabling lighting and texturing the triangles. Figure 2 shows that the lighting has relatively little affect on the graphics performance of this card. This is likely due to the specialized lighting hardware on the GPU. Texturing, as shown in Figure 3, drops the graphics performance a bit, but still has a minor impact. Figure 4 shows the affects of enabling both texture and lighting, which is similar to the texture test. This is due to the fact that the effect of lighting is negligible to the effect of texturing. Therefore, adding texture and lighting have no effect on the rasterization vs. primitive assembly crossover point.
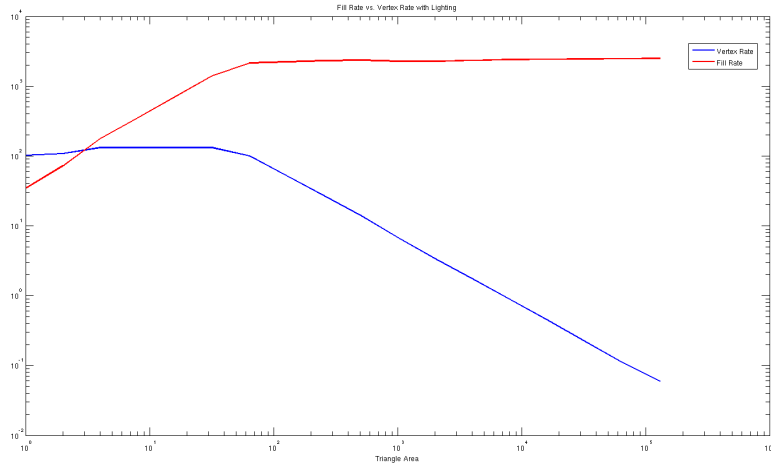
Figure 2: Vertex and fill rate for various triangle areas with lighting.
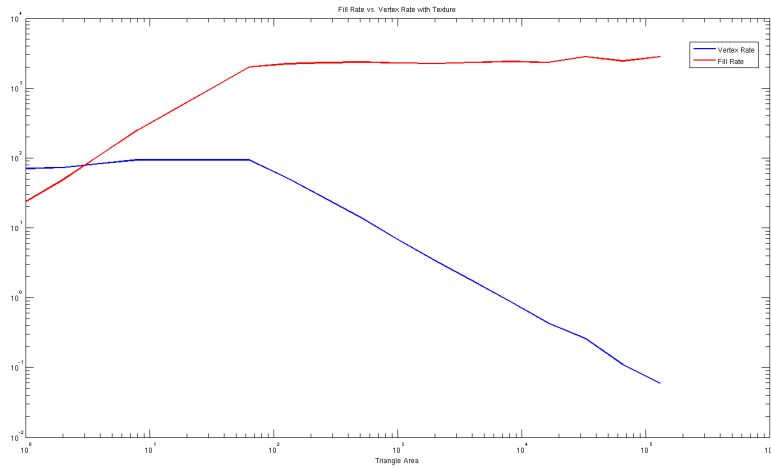
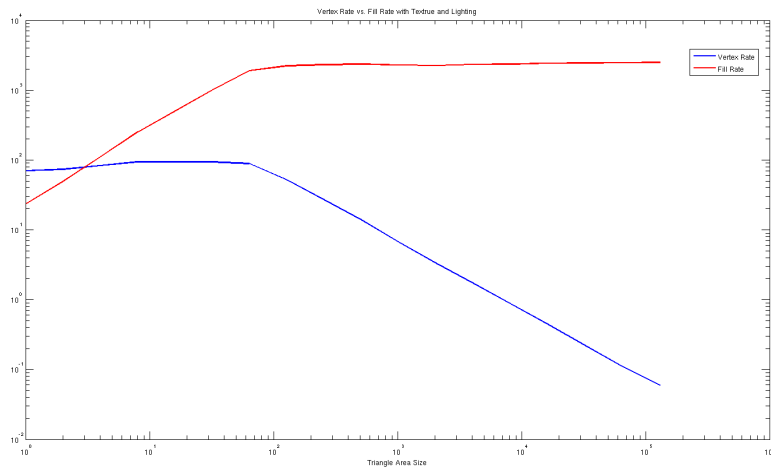Figure 3: Vertex and fill rate for various triangle areas with texture.

Figure 4: Vertex and fill rate for various triangle areas with lighting and texture.

2.3   Triangle Types

This tests the effect of sending triangles to the GPU by using disjoint or triangle strips. Figure 5 shows the vertex rate vs. the fill rate using disjoint triangles and appears exactly the same as all the other graphs, since disjoint triangles are the default for the tests. The vertex rate vs the fill rate for triangle strips are shown in Figure 6. Note that the crossover point shifts down to a triangle area of 16px due to the efficiency of triangle strips.
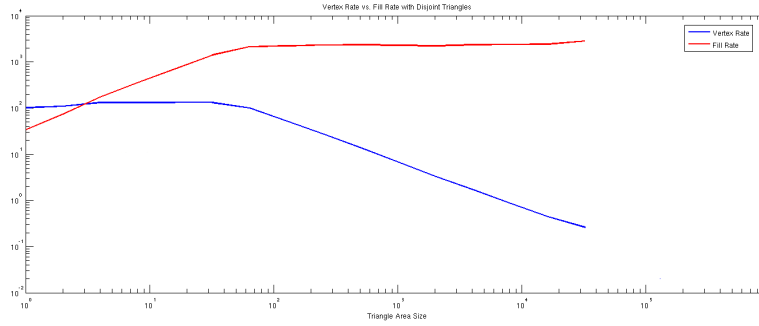

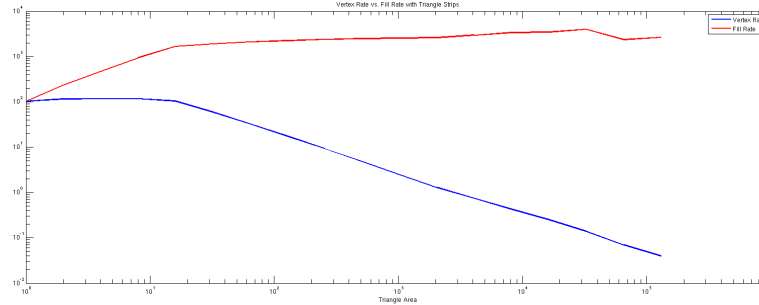Figure 5: Vertex and fill rate for various triangle areas with disjoint triangles.

Figure 6: Vertex and fill rate for various triangle areas with triangle strips.

2.4   Input Bandwidth

In this test, the program varies the number of vertices per bucket and records the associated vertex rate. The vertex rate increases as the vertices per bucket increases until it flattens at FJDSK, as seen in Figure 7. This indicates that, at first, the interface between the CPU and the GPU is the bottleneck and then it transfers to the geometry stage of the GPU. The second jump occurs because the area of the triangles was reduced to prevent memory overflow. However, the trend is still constant which indicates that the GPU geometry stage is still the bottleneck.
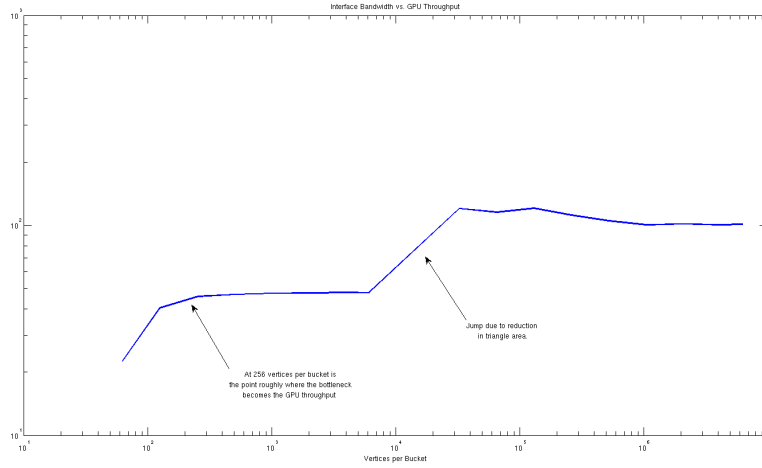

Figure 7: Vertex rate plotted vs the number of vertices per bucket.

6

2.5　Texture Size

This test sweeps the size of the texture used and shows how the fill rate changes, as seen in Figure 8. In particular, the fill rate suffers as the texture size increases. The tipping point is when the texture size reaches about 256 px. At that point, the fill rate begins to reduce more rapidly as the texture lookups hinder the rasterizer.
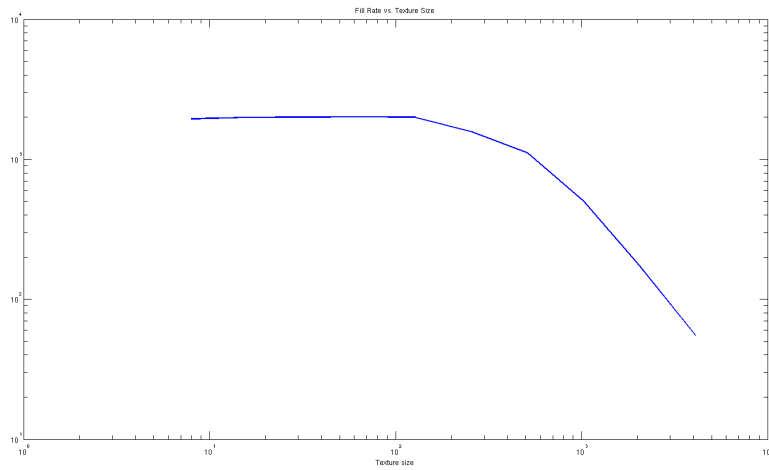

Figure 8: Fill rate for texture sizes.

3   Part B

In Part B, we examine the general purpose graphics processing unit(GPGPU) capabilities of the graphics card with a particular focus on the compute performance as characterized by the floating point operations per second (FLOPS), often represented in GigaFLOPS (GFLOPS). This part of the assignment primarily used CUDA to generate GPGPU code.

3.1   Maximum FLOPS

The maximum number of FLOPS capable by the graphics card is 87 GFLOPS. This data was found by creating a kernel to operate on the GPU that performed many register arithmetic operations and wrote those to a location in memory at the end.

3.2   Memory Bandwidth

The maximum memory bandwidth of the graphics card is 9.4 GB/s. We tested this by creating a kernel which read and modified many memory locations in a coalescing-friendly way amongst threads.

3.3   Effect of Blocks on FLOPS

This test examined how the number of blocks used affected the maximum FLOPS. To perform this test, the same kernel as the maximum FLOPS test was used but the number of blocks were altered for each complete run to obtain a full data set. As the number of blocks increases, the maximum FLOPS oscillate but increase, as seen in Figure 9. The oscillations occur as the number of block get closer or further from maximum occupancy. The overall increase occurs because more blocks mean more threads and more threads correspond to a higher minimum utilization.
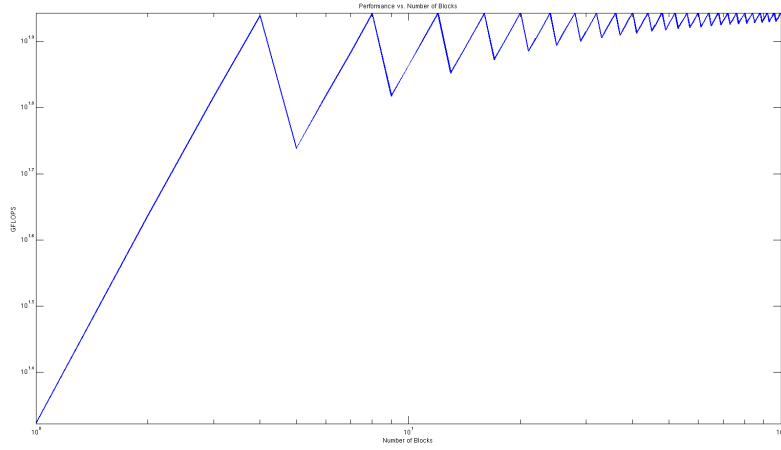

Figure 9: Performance (GFLOPS) vs. Number of Blocks.

3.4 Effect of Warps on FLOPS

This test altered the number of warps (groups of 32 threads) and recorded the maximum FLOPS at each of these points. The number of warps affected the FLOPS by increasing the number of threads which allows more data to run on the multiprocessors (until it reaches the GPU max FLOPS ceiling), as shown in Figure 10. The small oscillations occur because the warps cannot be mapped in a way that causes full occupancy for all possible values.
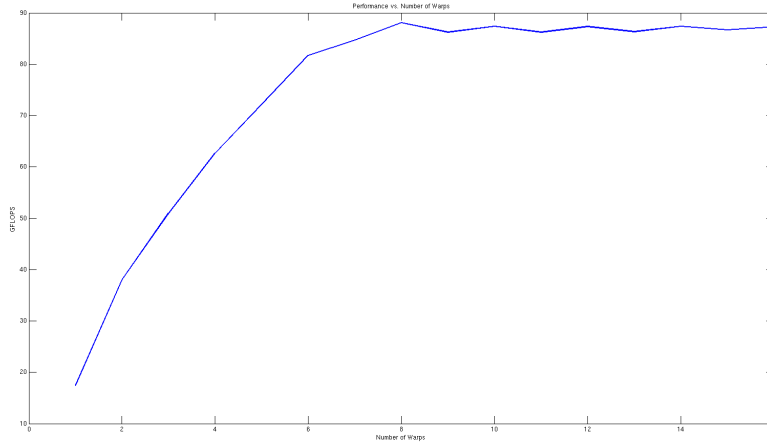

Figure 10: Performance (GFLOPS) vs. Number of Warps

3.5 Effect of Branch Granularity on FLOPS

This test changed the branch granularity within a defined kernel (that supported up to 32 branches) and measured the number of FLOPS that resulted from it. The specialized kernel used part of the FLOPS test from before but added a part that would be changed based on which branch was taken. As seen in Figure 11, the higher the branch granularity is, the worse the FLOPS become. This occurs because a higher branch granularity requires the other threads in the warp to stall while the branching threads complete their execution. This applies for each potential branch path that the threads in a warp take so more

10

paths (called granularity) results in more stalling while the different paths are computed. The curve of the data in Figure 11 occurs because as more branch directions are added, the overall effect of each direction on the average thread execution time is marginally reduced.
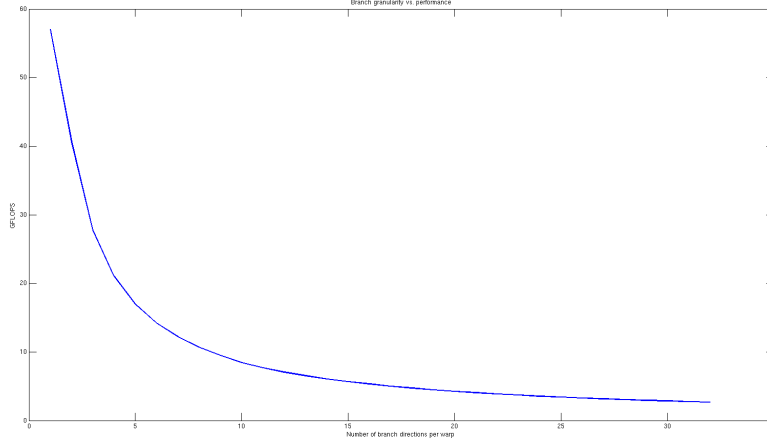

Figure 11: Performance (GFLOPS) vs. Branch Granularity.

3.6   Effect of Persistent Threads on FLOPS

The final test compared the use of persistent threads, using threads that matched to the hardware rather than to the work needed to be done, with the standard CUDA approach, which maps one thread to one piece of work that needs to be completed. As a result of this test, we saw a speedup using persistent threads from 87 GFLOPS to 89 GFLOPS, a 2.3% speedup.

11

4 Code

The code can be found on our GitHub repository:
`https://github.com/phil-monroe/EEC-277---Project-2`