EE 271 - Project Report
Hardware Improvements for Rasterizer Design

Krishna Teja Malladi (0558 2221)
Piyush Keshri (0559 4497)
Stanford University, Department of Electrical Engineering

November 29, 2009

1 Problem Statement

The goal of the project is to get a render speed of 500 Million μPolys/sec while consuming as little power and area as possible. The baseline rasterizer provided runs with a clock period of 1.05 ns and has a throughput of 0.083 μPolys/cycle(or 79 Million μPolys/sec).

2 Approach

The figure of merit we choose in the project to evaluate various options which satisfy the required throughput, is $FOM = \frac{Throughput}{Power}$. This figure of merit has been chosen, as it is independent of clock frequency. Our design methodology is to maximize FOM and to resolve the critical paths after the synthesis, to sacrifice more power (within power budget) if we want higher throughput by choosing higher frequency, f.

Of the suggested approaches, simple duplication of rasterizer for parallelization of micropolygon processing is the fundamental and easiest way to achieve the required throughput. However, it has the issues of extra overheads because of area and power. As the required throughput is 500 Million μPolys/sec, we would like to make the throughput of each parallel stage close to an integer fraction of 500. (i.e. either 500 or 250 or 166.67 so on). So, our approach is to choose design improvements so as to,

Optimize, $T(design, f) = \frac{500}{N}$ subject to constraints of minimum area & power.

where, T= throughput per stage & N = no. of parallel stages.

In our design, we try to implement the design improvements in sequential manner to achieve T close to one of the integer fractions suggested above without altering the clock frequency. If we realize that we cannot reach the next fraction, we try to squash a few of the improvements till we reach the current fraction of 500 so as to save power and area. This will maximize the figure of merit. Then, by resolving the most critical paths, we can further increase throughput by increasing f.

3 Improvements

We ran 200,000 test vectors on the baseline design and the base design results obtained are as follows: 2189483 clock cycles at t=1.05 ns. This is equivalent to 0.083 μpoly/cycle or 79 million μpoly/sec.

Cycles: 2189483 uPoly: 200001 sampleTests: 1989480 sampleHits: 252364

3.1 Backface Culling

The idea of backface culling is to recognize the directionality of the polygon vectors early in the design and disallow counter-clockwise vectored- polys as we know apriory that such polys will not be rendered anyway, since they point away from the display. Currently, they are eliminated in 'sampletest' i.e. at a deeper stage within the pipeline with halt for all the invalid sample points resulting in higher no. of clock cycles and lower sample hit to miss ratio.

3.1.1 Triangles

For triangle backface culling, we get the normal vector direction by taking cross product of any two vector edges of the triangle and make such triangles invalid which are back-facing (resulting normal out of screen). The Improvement results are as follows:

Cycles: 1764706 uPoly: 200001 sampleTests: 1564694 sampleHits: 252364

3.1.2 Quadrilaterals

The next logical idea is to extend the idea of backface culling to simple quadrilaterals (rhombus like) by using the auxilliary vector (diagonal like) and calculating four normal vectors to determine the directionality before backface culling. Since, the idea of backface culling to quadrilaterals result in overall improvement by around 15% further, the design is extended to take into account the backface culling for the complex quadrilaterals as well, taking into account the various possible directionality combinations. The improvement results are as follows:

Cycles: 1358541 uPoly: 200001 sampleTests: 1158529 sampleHits: 252362.

Next, we implemented backface culling for 'bow-tie cases' to make the design robust. The improved results obtained are as follows:

Cycles: 1358541 uPoly: 200001 sampleTests: 1158529 sampleHits: 252362

Remark: There is no cycle change for 'bow-tie' cases as either they have been removed considering backfacing complex quadrilaterals or are not present. Also, the discrepancy resulted in the number of sample hits (2 misses from baseline) obtained by the improved design has been investigated. The test case for the invalid result is:

1 4 fffe0c fffd15 000895 0000fc 00043c 000116 00056f 0003ec

The invalid sample has been incorrectly studied by the baseline design. Sampletest script is not able to eliminate the invalid test case (backfacing complex quadrilateral) and considers it as a 'bow-tie' case(since, the condition for 'bow-tie' is not exhaustive).

3.2 Bubble Smashing

Currently, we notice that it is possible for NOPs to appear in the pipeline when a poly is backface culled as it takes atleast one cycle of work to determine the facing. However, if we could advance a waiting polygon, we can save on the cycle. So we modified the design such that we read 2 polys in a given cycle (Though we currently read in the driver, it is fair to assume that in real-life the polys could be coming from a memory location and we can always extend the memory block-size to 2 from 1). Now, if we see that one of the fetched polys is invalid (either due to backface or invalid to begin with) we advance the other. If we see both are valid, we squash the program counter to serialize the polys and send one by one. By doing this, we have eliminated the cycle delay bubble that could have been due to the backface culling. The improved results obtained are as follows:

Cycles: 1315978 uPoly: 200001 sampleTests: 1158529 sampleHits: 252362

3.3 Idle FSM

We noted from the timing diagrams that there is an idle cycle in the FSM of the test_iterator. This arises because the evaluation of the signal validsamp_R13S waits for current_state to go to TEST stage while it could have right away made validsamp_R13S HIGH and passed a valid sample test. So we implemented this change and also modified the FSM to pass on of ll_x+MSAA (next right sample) or ll_y+MSAA (next up sample) in this cycle as ll_x is tested in the last cycle anyway.
Remark: Note that this mux to choose between ll_x or ll_y is because there could be cases where all sample points are on a vertical line.

Cycles: 1175706 uPoly: 200001 sampleTests: 1133845 sampleHits: 252362

3.4 Better Bounding box

[1] suggests a new method of traversing over the bounding-box : the increments are not along X or Y axis but along the edges of the polygon to reduce the hit to miss ratio. However the paper is addressed for bigger polygons before the concept of small micropolygons has been proposed for which the improvements would be not as much as desired. [2] mentions in Chapter 3 that the best method to reduce the hit to miss ratio for μpoly is obtained by merging triangles to quadrilaterals when possible, as for almost the same amount of traversal of bounding box, we can cover the sample points of both triangles. So, instead of method in [2], we implemented triangle merging as described in the next section.

3.5 Triangle Merging

The motive behind triangle merging to form a quadrilateral is that the bounding box is a tighter bound on the quadrilateral than on the triangle, which results in higher sample hits to misses ratio. This higher ratio is desirable as it increases the pipeline's micropolygon throughput. During triangle rasterization, we first backface cull the 2 fetched micropolygons and we observe if both micropolygons are valid triangles (after back culling) and share a common edge (the directionality of the common edge has to be opposite in the two triangles, otherwise we will obtain a five-sided polygon), we merge them to form a quadrilateral and pass the merged quadrilateral micropolygon to the pipeline stages.

Cycles: 1008565 uPoly: 200001 sampleTests: 966659 sampleHits: 252362

3.6 BBox

The idea is to round off lower bound of the bounding box to the upper right sample to reduce the number of sample tests. We tried ceiling the lower bound instead of floor and while we did observe a marginal performance benefit, it resulted in few sample misses. We think that this could be a problem with the sample counter as we were able to match the sample hit points to baseline. Added to that, this modification needed the modification of the logic of at_right_edg and at_top_edg from = (simple XOR) to > (24 bit comparator) which could be expensive in hardware. Due to the above reasons, the decision is that this could be the first improvement that will be dropped if we realize we cannot go to the next fraction of 500 Million μPolys/sec.

3.7 Multi test

We did not implement 'Multi samples test' improvement design as the verification collateral change and power penalty could be high. The other issue is that since, the micropolygons are very small in size, the number of samples per micropolygon are nearly around 12 samples/poly and investing much on hardware without large improvements and adding more overhead to control flow of sample points fetching is not worthy enough and hence not implemented. Neverthless, the hardware can be easily duplicated if required for multiple tests per cycle.

3.8 Duplication/Parallelism

By using all possible improvements, we were able to make T=197 million μPolys/cycle. So, the no. of duplication units needed is $\frac{500}{197} = 2.53$. which could have been made 2 if we make a few more optimizations like pipelining. However, we note that the margin to attain 500 Million μPolys/sec would have been small in that case as a very nasty test vector with no triangle merges, no backfaces can be possible in a real life situation. So, we dropped the Bbox ceiling idea to reduce T and P. Now, the hardware will have to be 3-way parallelized.

3.9 Cycle time reduction

The throughput per stage, T by making the discussed optimizations is about 197 million μPolys/cycle and FOM= 12.628 millionμpolys/sec-mW. We noted that the current critical path is from the 14 bit sample multiplication from sampletest. So, we reused the booth algorithm we designed for Assignment 2 to reduce the critical path and make the design synthesizable at t=0.72ns. This would give a final overall throughput of **275 Million μPolys/sec per stage** with (P, A) = (10.7 mW, 0.017736 mm^2) . So, now need **only 2 duplication** parallel units which give a total **Throughput of 550 Million μPolys/sec and (P, A)= (21.4 mW and 0.035472 mm^2)**.

4 Verification and Synthesis

We ran the 200,000 test vector on the modified design and observed the following results:

Cycles: 1008565 uPoly: 200001 sampleTests: 966659 sampleHits: 252362

The 2 samplehit misses are the bad sample hits which we caught (discussed in Section 3.1.2) and the no. of cycles has reduced from 2189483 to 1008565 showing an improvement of about 54%. We then synthesized the design in Synopsys by reducing the cycle time to 0.75ns and obtained no timing violations.

5 Results and Analysis

Though the approach we discussed above, we were able to obtain a throughput of **550 Million μPolys/sec**. Note that we squashed the ceiling optimization to save power. By choosing **2** multiple instances of rast, the final power and area can be computed as $T_{overall}$**=550 Million μPolys/sec, P= 21.4 mW and A= 0.035472** mm^2

Hence, our design meets the throughput requirements, has an optimized FOM value and also has power and area dissipation within the specified budget of 300mW and $1mm^2$ respectively.

The throughput improvements are tabulated below. Note that the improvements are incremental, i.e. each subsequent modification includes all previous improvements too.

Table 1: Throughput Improvements with optimizations

Improvement	**No.of cycles** #	**Clk period** (ns)	**Throughput** (Mils of μpoly/s)	**Cycle Impr** % over base
Baseline design	2,189,483	1.05	79	0
Triangle Backface Cull	1,764,706	1.05	108	19.4
Complex/Simple Quad Backface Cull	1,358,541	1.05	140	37.95
Bow-tie Quad- Backface Cull	1,358,541	1.05	140	37.95
Bubble Smashing	1,315,978	1.05	145	39.89
Idle FSM	1,175,706	1.05	162	46.30
Triangle Merging	1,008,565	1.05	189	53.93
Clock period reduction	1,008,565	0.72	275	53.93
Hardware Duplication	1,008,565	0.72	550	53.93

6 Future Work

There are three more optimizations we have thought and tried to implement but faced difficulty due to some issues:

6.1 Dynamic μpoly Fetch

In this optimization, we have modified the rast-driver to first fetch a μpoly and then dynamically keep fetching next μpoly until we get a frontface μpoly. By this, we will never even fetch an invalid μpoly into the second word and we have seen a throughput improvement to **574 Million μPolys/sec** as shown below:

```
Cycles: 968233 uPoly: 83356 sampleTests: 967575 sampleHits: 252362
```

However, the issue is that this dynamic process might present a critical path as the next frontface could be far apart. To eliminate this, we have put a limit of 100 in the above experiment. So, in future if we have control on the up-stream of rasterizer, we can surely induct this idea for better throughput.

6.2 Sample Test Reduction

The idea which we implemented is that in bounding box traversal, whenever we notice that the current sample is a miss and the previous sample is a hit, we no longer need to move to next_rt_sample (only in case of Triangles/ simple Quads). Instead we can go to next_up_sample. Unfortunately, we did not see any improvement by implementing this as the μpolys are very small and this condition could occur with a low probablity. Improvements:

Cycles: 1008565 uPoly: 200001 sampleTests: 966659 sampleHits: 252362.

6.3 Dynamic μpoly Fetch using Probability

If we have a dynamic probability distribution table of backface μpolys, we can fetch dynamically from the sampletests (instead of contiguous fetches: as the chance of back-facing among consecutive μpolys has a high correlation). This sure will have some sample misses but on rendering on a big screen, it might not effect the accuracy.

References

[1] J. Pineda, "A parallel algorithm for polygon rasterization," *Computer Graphics*, 1988.

[2] E. Luong, "Data parallel rasterization of micropolygons," Ph.D. dissertation, Stanford University, 2009.