NVIDIA Technical Report NVR–2012–02
Presented at HPG2012 as a poster

Understanding the Efficiency of Ray Traversal on GPUs – Kepler and Fermi Addendum

Timo Aila Samuli Laine Tero Karras

NVIDIA Research

Abstract

This technical report is an addendum to the HPG2009 paper "Understanding the Efficiency of Ray Traversal on GPUs", and provides citable performance results for Kepler and Fermi architectures. We explain how to optimize the traversal and intersection kernels for these newer platforms, and what the important architectural limiters are. We plot the relative ray tracing performance between architecture generations against the available memory bandwidth and peak FLOPS, and demonstrate that ray tracing is still, even with incoherent rays and more complex scenes, almost entirely limited by the available FLOPS. We will also discuss two esoteric instructions, present in both Fermi and Kepler, and show that they can be safely used for faster acceleration structure traversal.

Introduction

Table 1 extends "Understanding the Efficiency of Ray Traversal on GPUs" [AL09] with results for Kepler and Fermi architectures. As can be seen from the graph in Figure 3, the measured traversal and intersection performance continues to scale almost perfectly with the peak FLOPS of the GPUs, indicating that ray tracing has yet to hit the memory bandwidth wall and that instruction-level code optimizations continue to be relevant.

Implications of memory architecture

NVIDIA's Fermi architecture [NVI10] includes a conventional cache hierarchy, which reduces the average latency of memory fetches. That said, Fermi's L1 cache services requests to only one cacheline per clock, and replays the fetch instruction until all threads of a warp have been serviced, consequently introducing a new bottleneck for incoherent accesses. Highly optimized traversal and intersection kernels are in fact limited by the L1→SM communication, even in cases where the L1 hit rate is high and external memory bandwidth largely unused. Speculative traversal is less useful on Fermi than it was on Tesla primarily because it causes divergent memory accesses, but it still helps with incoherent rays and in larger scenes.

The Kepler architecture [NVI12a] brings a major upgrade to the SM's peak FLOPS and practically no improvements to the L1. As a result its L1 is useful only for coherent, low-priority accesses such as traversal stack and ray fetches. On the positive side, Kepler significantly upgrades the texture units, and since these units tolerate incoherent accesses rather well, it is beneficial to fetch all BVH node and triangle data through texture caches. Since the average latency of texture fetches is high, one should avoid dependent fetches whenever possible, even at the cost of fetching data that has a low probability of being useful. Speculative traversal is clearly beneficial with Kepler, particularly with incoherent diffuse rays and more complex scenes because address divergence does not overwhelm its texture caches.

Aila and Laine [AL09] hypothesize that it might be beneficial to replace terminated rays once the SIMD utilization drops below a threshold. In practice this is not overly useful on Fermi, probably because it makes memory accesses less coherent and amplifies the L1→SM bottleneck, but on Kepler it does improve the performance of incoherent rays as long as the data is accessed using texture fetches.

Faster ray-box test using VMIN, VMAX instructions

Ray-box intersection is the most time-consuming operation when traversing bounding volume hierarchies, and a large fraction of the instructions is spent in selecting minimum or maximum floating point values (Fig. 1). Esoteric PTX-exposed [NVI12b] instructions called VMIN and VMAX can be used for accelerating this computation on Fermi and Kepler. They can perform `max(min(a,b),c)` or `min(max(a,b),c)` in one op for signed 32-bit integers.

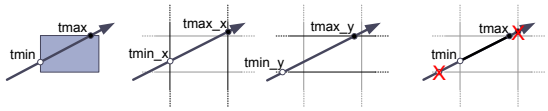


Figure 1: *This series illustrates the span intersection in ray-box test. We intersect the ray to two vertical planes to obtain (tmin_x,tmax_x), and two horizontal planes to obtain (tmin_y,tmax_y). In three dimensions there would also be (tmin_z,tmax_z). Finally, tmin is the largest of tmin_x, tmin_y and tmin_z. Similarly, tmax is the smallest of tmax_x, tmax_y and tmax_z.*

```
DEFINITIONS
B = Box (xmin,ymin,zmin,xmax,ymax,zmax);
O = ray origin (x,y,z);
D = ray direction (x,y,z);
invD = (1/D.x,1/D.y,1/D.z);
OoD  = (O.x/D.x,O.y/D.y,O.z/D.z);
tminray = ray segment's minimum t value; ≥0
tmaxray = ray segment's maximum t value; ≥0

RAY vs. AXIS-ALIGNED BOX
// Plane intersections (6 x FMA)
float x0 = B.xmin*invD[x] - OoD[x]; [−∞,∞]
float y0 = B.ymin*invD[y] - OoD[y]; [−∞,∞]
float z0 = B.zmin*invD[z] - OoD[z]; [−∞,∞]
float x1 = B.xmax*invD[x] - OoD[x]; [−∞,∞]
float y1 = B.ymax*invD[y] - OoD[y]; [−∞,∞]
float z1 = B.zmax*invD[z] - OoD[z]; [−∞,∞]

// Span intersection (12 x 2-way MIN/MAX)
float tminbox = max4(tminray, min2(x0,x1),
                min2(y0,y1), min2(z0,z1));
float tmaxbox = min4(tmaxray, max2(x0,x1),
                max2(y0,y1), max2(z0,z1));

// Overlap test (1 x SETP)
bool intersect = (tminbox<=tmaxbox);
```

Figure 2: *Pseudocode for ray-box intersection.*

The use of integer comparisons with floating point data is potentially dangerous since it gives correct results only if at least one of the arguments is non-negative. This condition is violated by our code, see Figure 2, but gladly we need only `intersect` and `tminbox` to be correctly set in the end. The latter determines the processing order of child nodes when more than one is intersected.

Since we know by construction that `tminray` is non-negative, `tminbox` will also be non-negative. Thus it does not matter that the `min2()` operations may give wrong intermediate results; when the largest of these and the `tminbox` is chosen, the potentially incorrect negative values will be ignored.

`tmaxbox` can be incorrect because the `max2()` operations may select the wrong negative numbers. Still, `max2(x0,x1)` is negative only when both of the ray's intersections with x-aligned planes are behind the ray's origin, and in that case the ray segment cannot intersect the box. Same is true for y and z. In these cases `intersect` must be false, which is what happens whenever `tmaxray` is a negative number.

So, even though most of the intermediate results can indeed be wrong, all of the end results are guaranteed to be correct. The span intersection is then simplified to:

```
// Span intersection (6 x VMIN/VMAX)
float tminbox = vmin_max(x0,x1,vmin_max(y0,y1,
                vmin_max(z0,z1,tminray)));
float tmaxbox = vmax_min(x0,x1,vmax_min(y0,y1,
                vmax_min(z0,z1,tmaxray)));
```

Summary

The most relevant optimizations on Tesla, Fermi, and Kepler architectures are summarized in Table 2.

The optimized kernels are available at http://code.google.com/p/understanding-the-efficiency-of-ray-traversal-on-gpus/

Acknowledgements

Marko Dabrovic (www.rna.hr) for the Sibenik cathedral model. University of Utah for the Fairy scene. Guillermo M. Leal Llaguno (www.evvisual.com) for the San Miguel scene.

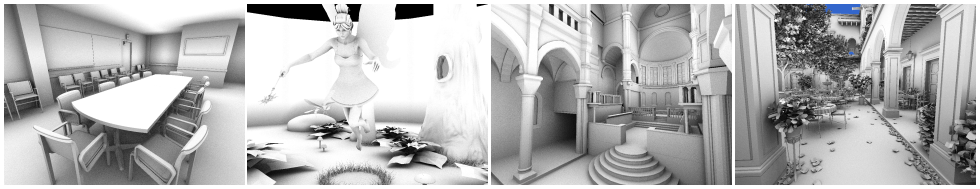
References

[AL09] AILA T., LAINE S.: Understanding the efficiency of ray traversal on GPUs. In *Proc. High-Performance Graphics 2009* (2009), pp. 145–149.

[NVI10] NVIDIA: NVIDIA's next generation CUDA compute architecture: Fermi. Whitepaper, 2010.

[NVI12a] NVIDIA: NVIDIA's next generation CUDA compute architecture: Kepler GK110. Whitepaper, 2012.

[NVI12b] NVIDIA: Parallel thread execution ISA version 3.0. Whitepaper, 2012.



		Conference, 283K tris			Fairy, 174K tris			Sibenik, 80K tris			San Miguel, 11M tris		
	Ray type	Tesla [AL09]	Fermi	Kepler	Tesla	Fermi	Kepler	Tesla	Fermi	Kepler	Tesla	Fermi	Kepler
Measured (MRays/s)	Primary	142.2	272.1	432.6	74.6	154.6	250.8	117.5	243.4	388.2	–	76.9	131.7
	AO	134.5	284.1	518.2	92.5	163.6	317.6	119.6	244.1	441.2	–	94.5	187.9
	Diffuse	60.9	126.1	245.4	40.8	73.2	156.6	46.8	94.7	192.5	–	33.3	58.8
× previous architecture	Primary		1.91	1.59		2.07	1.62		2.07	1.59			1.71
	AO		2.11	1.82		1.77	1.94		2.04	1.81			1.99
	Diffuse		2.07	1.95		1.79	2.14		2.02	2.03			1.77

Table 1: *Performance measurements in MRays/sec for Tesla (GTX285), Fermi (GTX480) and Kepler (GTX680) using the setup from Aila and Laine [AL09]. The scaling between generations is visualized in Figure 3.*

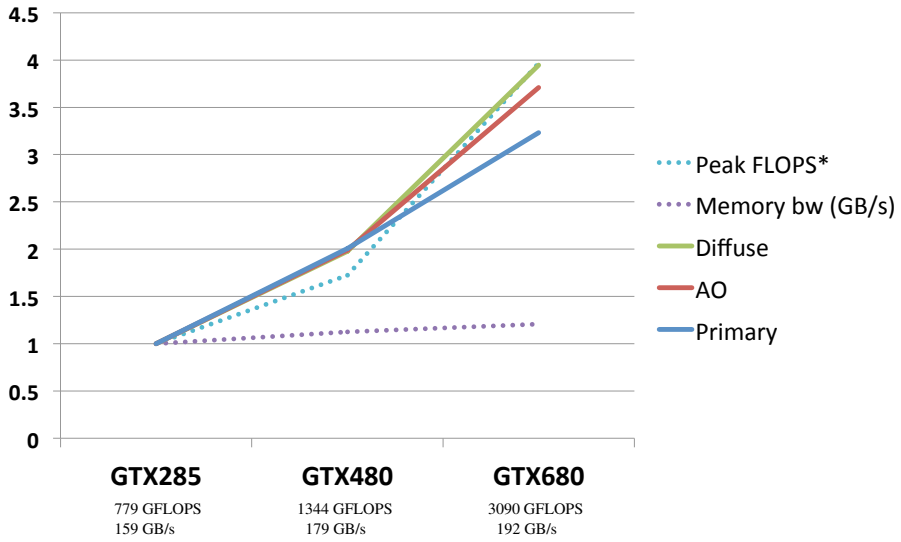


Figure 3: *Relative average performance (MRays/sec) of primary, ambient occlusion (AO), and diffuse rays in our four test scenes on GTX285, GTX480, and GTX680, plotted against the relative memory bandwidth and peak FLOPS. Ray tracing performance continues to follow peak flops very closely, while memory bandwidth has increased at a much slower rate. It seems that the primary predictor of ray tracing performance is still the achievable peak flops rather than memory bandwidth. Interestingly, diffuse rays seem to scale even better than primary rays, but that is an artifact caused by Kepler optimizations that favor incoherent rays. Since the peak flops of GTX285 can only be achieved in one very specific scenario, we adjusted its dual issue rate from 2.0 to a more realistic upper bound of 1.2 to be more directly comparable with newer architectures.*

	Tesla [AL09]	**Fermi**	**Kepler**
Data fetches	Fetch nodes through texture, triangles from (uncached) global memory.	Fetch nodes through L1, triangles through texture. L1 is a bottleneck with incoherent rays but texture is not fast enough to fetch everything.	Fetch all node and triangle data through texture, and avoid dependent fetches whenever possible. L1 is slow and beneficial only for traversal stacks and ray fetches.
Persistent threads	Doubles performance.	Not beneficial due to a better hardware work distributor.	Can replace all terminated rays once fewer than 60% of the warp's lanes have work. Favors incoherent rays, +10%.
Speculative traversal	Nearly always useful.	A small win for incoherent rays and large scenes.	One or two-slot postpone buffer is beneficial for incoherent rays.
VMIN, VMAX	Not available.	A trivial +10%.	Less useful than on Fermi because their throughput is lower, +5%.

Table 2: *Summary of differences in traversal and intersection kernels for Tesla, Fermi, and Kepler.*