OptiX vs Embree using Vertex Connection and Merging

André Bico[1]

Abstract

The OptiX and Embree frameworks were developed by NVIDIA and Intel to implement physically based rendering methods. These frameworks have been adopted by several commercial applications to accelerate rendering on modern parallel hardware. To test the merits of these frameworks, we implemented and tested two rendering engines with them, that implement Vertex Connection and Merging (VCM) a state of the art global illumination algorithm. In our experience OptiX delivered the best application performance and Embree was the easiest framework to use.

Keywords: Embree, Optix, Rendering, Global illumination, VCM, Parallel development

1. Introduction

Over the years there has been a growing need for the realistic rendering of three dimensional scenes. Due to their inherent parallelism, rendering algorithms have naturally been adapted from CPUs to GPUs, yet these implementations barely resemble their original simple and elegant versions.

Specific hardware architecture issues and differences in the development environment make programming for GPUs a hard task. To alleviate these issues, NVIDIA developed OptiX, a framework that takes advantage of the high performance of modern day GPUs. GPU hardware delivers tantalising performance results when rendering complex scenes [DKHS14]. CPU development did not fall behind, it has had continuous hardware and software improvements. This is manifest in the Intel Embree framework, that allows the development of high performance ray tracing kernels, as a competitive framework on CPU hardware.

NVIDIA and Intel state that their frameworks and hardware provide the best solution for the rendering of realistic three dimensional scenes. These statements are backed with performance results in [PBD*10] and [WWB*14]. In practice, commercial applications use both solutions, as an example, Mental Ray and Adobe After Effects use OptiX, while V-Ray and Corona Renderer use Embree. It is therefore important to analyse these frameworks to understand which type of hardware and framework are worth investing time into.

2. Vertex Connection and Merging

To understand the reasons behind the performance claims of the vendors it is important to see which algorithms were used to test these frameworks. In [WWB*14] it is shown that Embree comes ahead performance wise over OptiX with the path tracing algorithm. Even though this method is unbiased and straightforward to implement, it does not render newer and more representative light effects. [PBD*10] uses a technique similar to photon mapping [HOJ08] to present its performance evaluation. This method relies on the number of photons, instead of samples, to increase render quality which can cause problems when it comes to reducing artefacts.

Even though both performance analyses are valid recent algorithmic developments warrant a new analysis. Looking at the state of the art in photorealistic rendering it can be surmised that Vertex Connection and Merging (VCM) developed by [GKDS12] currently provides the best representation as it merges Bidirectional Path Tracing [Vea98] and Photon Mapping [HOJ08] into a combined formulation. OptiX and Embree were developed to provide fast performance boosts to previously existing methods that depend on ray traversal. VCM was originally a CPU based implementation so taking this formulation and adapting it to OptiX and Embree is a significant challenge to overcome. So we can test the capabilities of the frameworks by looking at how they are able to provide performance boosts to this more recent algorithm.

3. Solution Architecture

We can now present our simple unified parallel architecture. As shown in Figure 1 our solution has been divided into four main components, *LightTracing*, *PhotonGridConstruction*, *CameraTracing* and *CumulativeSampling*.

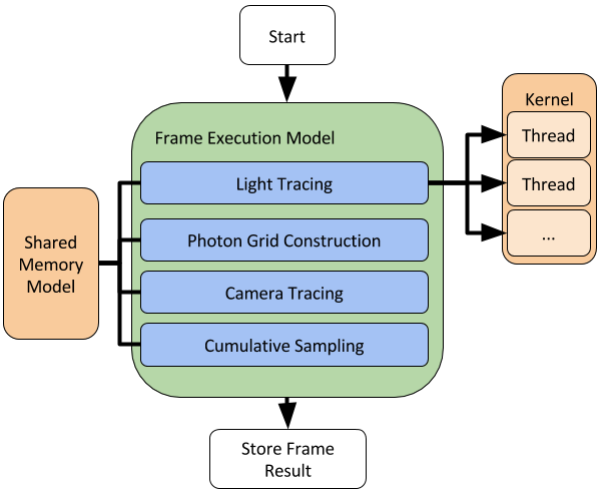


Figure 1: Unified execution model.

VCM creates paths from light sources and the camera eye so we split rendering into the *LightTracing* and *CameraTracing* phases. This formulation is similar to the way Bidirectional Path Tracing works. The base algorithm also requires a photon map to be constructed, based on the *LightTracing* results, this is generated in the *PhotonGridConstruction* phase. Furthermore to see a significant improvement in image quality we must cumulate samples at the end of each frames execution this is done in the *CumulativeSampling* phase.

Our formulation focuses on the execution of each individual component, when one component finishes threads start executing the next component. This requires a shared memory model as threads concurrently update values. That can be accommodated by the OptiX framework, and in Embree by using the Threading Building Blocks library (TBB).

3.1. OptiX

Our solution in OptiX, Figure 2 consists of a regular C++ application, using OpenGL, together with several OptiX CUDA programs running on the GPU. In this solution colour calculations and ray traversal are done on the GPU with ray generation programs, written in CUDA, and then the calculated result is returned to the host code with storage on a buffer or texture.

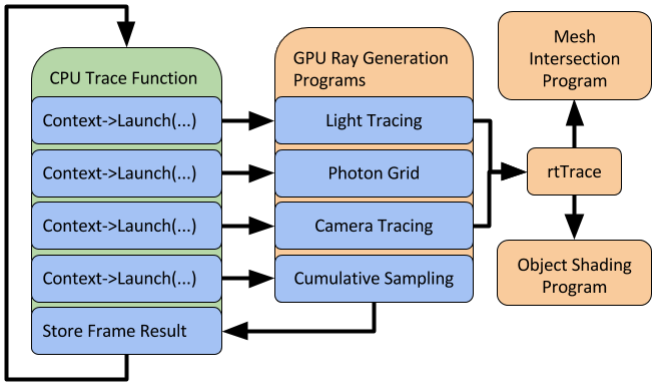


Figure 2: OptiX execution model.

3.2. Embree

Instead of compiling different CUDA programs and using them from host code, as we do in OptiX, here the process is much simpler. There is a set of required scene objects and structures that the implementations have to follow, but in the end everything can be changed and programmed in any way the developer wants. In the same way as in OptiX, our Embree solution in Figure 3 works together with a regular OpenGL C++ application. We also divide it into host and device code, the first controls initialization and parallel launch, while device code contains ray traversal and colour calculations. This is just a design adherence to the examples provided by Embree as any application can have different functions and design principles into it.

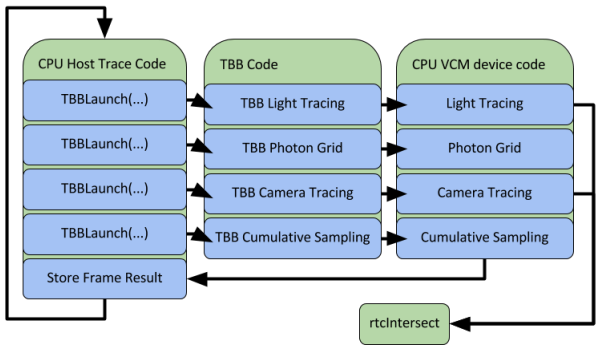


Figure 3: Embree execution model.

4. Acceleration Structures

The main features provided by OptiX and Embree, are state of the art acceleration structures. As such our Embree solution uses a quad branching bounding volume hierarchy, while the OptiX implementation uses a standard *bvh* structure.

5. Metrics

Before moving into the testing section we go through a brief overview over the image quality and performance metrics we use.

5.1. Image Quality Metrics

There is a high number of available metrics to qualify images but they do not have a generalised use [vHM*12]. Image quality can be quantified by mathematical metrics that are divided into two major categories, those which depend on reference images and those which do not.

Reference images represent the perfect quality achievable by an algorithm. By comparing the results of two implementations of the same algorithm, with the reference images, one can determine if a render result is better or worse. However, when algorithms generate different graphical effects, it is unfeasible to compare them with reference images. Non-image reference metrics evaluate, individually, the quality of an image, for example, some of them quantify the amount of noise or contrast present on a single image.

5.1.1. SSIM

Structure Similarity Index Metric (SSIM) is a state of the art image quality metric [WSB03] used to perceive the degree in which a distorted image is similar to its undistorted reference.

$$SSIM(x,y) = [l(x,y)]^{\alpha}[c(x,y)]^{\beta}[s(x,y)]^{\gamma}$$

We consider it a good metric to compare two different render results, because it analyses luminance $l(x,y)$, contrast $c(x,y)$ and the structural component $s(x,y)$ over two image areas x and y. Luminance evaluates the intensity of an image, contrast determines the brightness and colour differences, while the structure component takes into account the tendency of x and y to vary together, showing how similar both images are.

Independently if we consider x or y the image reference, the metric result is equal. This allows us to compare OptiX and Embree image results without a reference image.

5.1.2. SNS

Contrast enhancement techniques are methods that augment an image by increasing its sharpness. From these we can derive Speckle Noise Strength (SNS) as a metric to evaluate speckle noise [IH14]. This type of noise is represented by a granular look present on digital images.

Since noise is the biggest drawback of global illumination algorithms, because of estimator bias, quantifying the amount of noise present in an image allows conclusions about its quality to be made.

This metric works by applying a median filter of size 25×25 pixels to the original image. A median filter iterates all pixel values on a image and replaces their value with a median of nearby values. This metric then compares the luminance of the filtered image with its original version. Regardless of its main area of application, this metric is easy to implement, and quantifies speckle noise in a range of $[0, 100]$, where higher values mean higher noise.

5.1.3. EME

Called Measure of Enhancement (EME) [ALG00] estimates the contrast of an image. In this context contrast is considered as the perceived brightness of an image. Higher values means that the generated image contains more contrast.

This metric divides an image into blocks of a specified size. In our testing scenario we divided the image with blocks of one pixel. The next process is comparing the minimum and maximum pixel colour values between each block.

By evaluating contrast one can determine the quality of a render result. High contrast results can mean that a result achieved more clear and defined results. However high contrast can be obtained for images that present high noise. This is the main reason why we evaluate contrast and noise to determine quality.

5.2. Performance Metrics

The OptiX framework presentation article [PBD*10] takes an implementation of the photon mapping algorithm and evaluates its performance in terms of total execution times. [WWB*14] tests and compares Embree and OptiX with the path tracing algorithm. Both implementations are then evaluated in terms of rays per second. Naturally, we took these metrics and added others to our testing scenarios. We present the full list of performance metrics used together with a small explanation as to why they were used.

5.2.1. Runtime in Seconds

In our testing scenarios, total runtime is set as the maximum time we let the solutions run. This is done to see which solution achieved the best results given the same rendering time.

5.2.2. Iterations

In our solutions an iteration corresponds to the complete execution of all components of the VCM algorithm. Additionally, all iterations are rendered with the same parametrization, it does not change as render time elapses. The purpose of this metric is to see which solution rendered the most iterations given the same rendering time.

5.2.3. Rays per Second

Rays per second is meant to determine which solution emitted most rays. High amounts of traced rays means that more calculations were done.

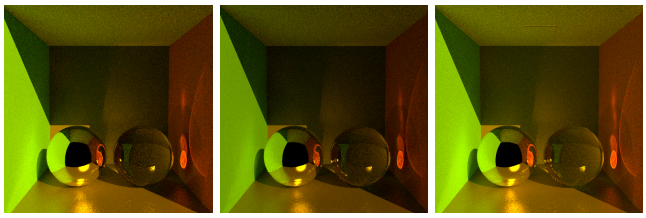
5.2.4. Individual Execution Times

We measure the execution time of each component in our parallel architecture to see its absolute performance cost.

6. Validation Results

Before comparing our rendering engines we decided to validate them with the implementation supplied by the authors of the VCM algorithm (SmallVCM). This allows us to guarantee that our implementations are proper independent variants. In this test phase we will show the results of two replicated scenes, a scene with mirror and glossy materials, together with their variants with directional, and point light sources. In terms of hardware, this validation ran the Embree and SmallVCM implementations on a Intel Core I7-4710HQ 2.50GHz, while the OptiX implementation ran on a Nvidia 860m. We believe that using this hardware is enough at this stage, as we ran SmallVCM in the same machine as our implementations.

6.1. Directional Light with Mirror and Glossy Materials



OptiX Embree SmallVCM

Figure 4

The performance results in Table 2 for the images in Figure 4, show Embree and OptiX performing faster while rendering the same, or more, iterations in less time than SmallVCM. Looking at SSIM in Table 1 it can be concluded that our implementations are similar to SmallVCM. Furthermore, in terms of SNS and EME present in Table 1, the values obtained show that our implementations achieve equal or higher contrast, albeit with increased noise.

Parametrization:
Geometry: 1930 triangles.
Path Depth: 8.
Render Time Aim (Embree and OptiX): 13 seconds.
Iteration Target Aim (SmallVCM): 40.
Resolution: 512x512 pixels.

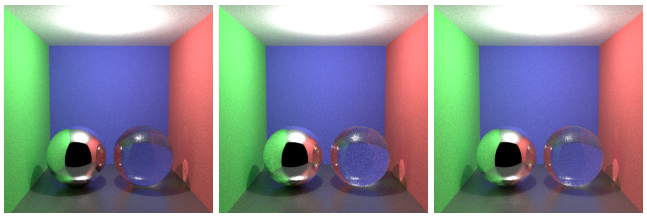
Implementation	SNS	EME	SSIM
Embree	2.97	285.05	0.84
OptiX	3.45	287.95	0.95
SmallVCM	2.97	180.04	1

Table 1: Image quality comparison.

Implementation	Runtime (s)	# Iterations
Embree	13.14	40
OptiX	13.03	44
SmallVCM	16.66	40
Implementation	Average Iterations	MRays/s
Embree	3.04	5.40
OptiX	3.38	5.11
SmallVCM	*not measured*	*not measured*

Table 2: Performance comparison.

6.2. Point Light with Mirror and Glossy Materials



OptiX Embree SmallVCM

Figure 5

Results in Table 3 show a performance advantage of both OptiX and Embree versus SmallVCM, for the point light scene shown in Figure 5. OptiX rendered the highest number of iterations, and all image quality results in Table 4 are similar.

Parametrization:
Geometry: 1930 triangles.
Path Depth: 8.
Render Time Aim (Embree and OptiX): 13 seconds.
Iteration Target Aim (SmallVCM): 29.
Resolution: 512x512.

Implementation	Runtime (s)	# Iterations
Embree	13.56	29
OptiX	13.22	36
SmallVCM	15.56	29
Implementation	Average Iterations	MRays/s
Embree	2.14	5.35
OptiX	2.72	4.32
SmallVCM	*not measured*	*not measured*

Table 3: Performance comparison.

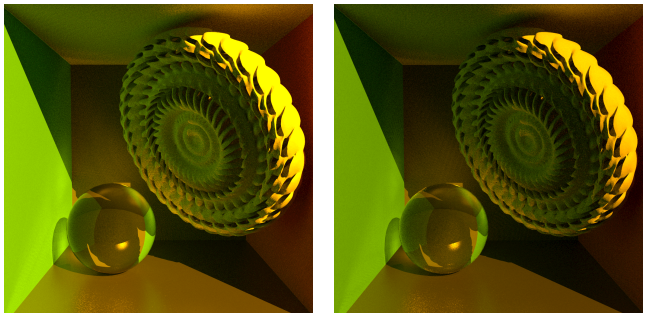
Implementation	SNS	EME	SSIM
Embree	2.29	145.33	0.97
OptiX	2.24	145.45	0.98
SmallVCM	2.30	143.88	1

Table 4: Image quality comparison.

7. OptiX vs Embree Results

After validating the rendering quality of our solutions we proceed to compare the Embree and OptiX implementations. The test procedure is similar to that of the validation stage except it is conducted on higher performance hardware with a higher resolution rendering parametrization. The hardware used for this test phase was a Intel Core I7-4770K 3.50 GHz and a NVIDIA GeForce GTX 780 Ti. Results include the execution time for each component in our solution.

7.1. Complex Directional Light with Glossy and Diffuse Materials



OptiX Embree

Figure 6

The scene presented in Figure 6 consists of a loaded fractal geometry with a diffuse material associated to it, next to a refractive sphere. We observed in Table 5 that OptiX renders more than twice the iterations than Embree. Additionally, Table 7 shows a surprising bottleneck in the camera tracing component of Embree. Despite the significantly different number of rendered iterations, it did not have a significant impact in terms of final image quality, as shown in Table 6.

Parametrization:
Geometry: 370 406 triangles.
Maximum Path Depth: 8.
Render Time Aim: 120 seconds.
Resolution: 1024x1024 pixels.

Implementation	Runtime (s)	# Iterations
Embree	120.62	80
OptiX	120.02	170
Implementation	Average Iterations	MRays/s
Embree	0.66	4.88
OptiX	1.42	7.85

Table 5: Performance comparison.

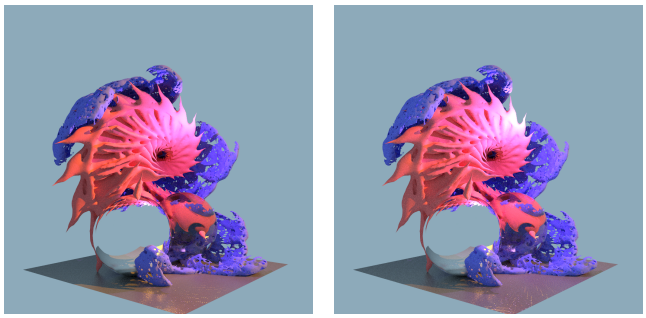
Implementation	SNS	EME	SSIM
Embree	2.85	291.19	0.81
OptiX	3.16	292.89	0.81

Table 6: Image quality comparison.

Implementation	Grid Construction	Light Tracing
Embree	11.60 s	17.54 s
OptiX	39.60 s	23.70 s
Implementation	Camera Tracing	Cumulate Samples
Embree	84.58 s	0.26 s
OptiX	54.08 s	0.13 s

Table 7: Total execution times in seconds.

7.2. Complex Directional, Point and Environment Lights with Reflective and Glossy Materials



OptiX Embree

Figure 7

The rendered scene in Figure 7 features a more compact and complex set of objects and light sources making it a harder scene for rays to traverse. It can be seen in Table 9 that the OptiX implementation emits more rays than the one with Embree. Furthermore, this scene, even though more complex, renders faster than the less complex scenes present in the previous sections. This is because the area surrounding the scenes main geometry is not occupied with an object. So for those particular pixels no light or camera paths are created and the colour is equal to the environment light source.

The biggest performance hazard on OptiX is the grid construction present on Table 10, mainly because it takes multiple kernel launches and variable allocations. While Embree grid construction is much faster it performs worse while executing the light and camera tracing components. In terms of image quality, the results were very similar, with OptiX featuring slightly more contrast as shown in Table 8.

Parametrization:
Geometry: 1 094 022 triangles.
Maximum Path Depth: 8.
Render Time Aim: 120 seconds.
Resolution: 1024x1024 pixels.

Implementation	SNS	EME	SSIM
Embree	1.68	145.81	0.98
OptiX	1.74	173.09	0.98

Table 8: Image quality comparison.

Implementation	Runtime (s)	# Iterations
Embree	120.25	152
OptiX	120.33	243
Implementation	Average Iterations	MRays/s
Embree	1.26	4.76
OptiX	2.02	10.46

Table 9: Performance comparison.

Implementation	Grid Construction	Light Tracing
Embree	19.62 s	42.45 s
OptiX	54.89 s	37.14 s
Implementation	Camera Tracing	Cumulate Samples
Embree	46.01 s	0.50 s
OptiX	23.10 s	0.18 s

Table 10: Total execution times in seconds.

8. Conclusions

OptiX and Embree were designed to boost the performance of state of the art algorithms which are dependent on ray traversal. We pursued this design goal with the implementation of two rendering solutions based on VCM. Tests on these implementations allow us to comment on the suitability of each framework for the implementation of global illumination algorithms.

Looking at the comparative results, between Embree and OptiX, the OptiX implementation showed a clear performance advantage across the test scenarios beating the Embree implementation on most performance metrics. We consider these results as reliable since the quality of our implementations was validated with SmallVCM. The implementations reached a higher performance without the sacrifice of image quality.

On the programming models it is clear that OptiX has a more complex architecture increasing the barrier of entry to the framework. Furthermore when it comes to debugging and executing CUDA code, driver errors and crashes can occur, making the framework hard to use. Embree bypasses all of these problems by using the same code base as any C/C++ application. Its advantage lies in how expedient it is to plug its features to an already existing CPU renderer and obtain good performance code.

While compiling and using these frameworks, Embree also fares ahead in how easy it is to setup in comparison to OptiX. This is mainly because there is no requirement to use a domain specific language like CUDA or to compile to another hardware architecture. When running tests on different machines OptiX required constant recompilation to fit the hardware being used.

Despite its programming complexity OptiX has proved itself as the most suitable framework to implement global illumination algorithms. However we can conclude that both OptiX and Embree have a place in high end rendering. On one hand Embree is the best solution if the goal is to achieve the fastest performance without many development considerations. On the other hand OptiX allows better performance in the long run but requires time, dedication, and careful programming.

References

[ALG00] Agaian S. S., Lentz K. P., Grigoryan A. M.: A new measure of image enhancement. In *IASTED International Conference on Signal Processing and Communication* (2000). 3

[DKHS14] Davidovič T., Křivánek J., Hašan M., Slusallek P.: Progressive light transport simulation on the gpu: Survey and improvements. *ACM Trans. Graph. 33*, 3 (June 2014), 29:1–29:19. URL: http://doi.acm.org/10.1145/2602144, doi:10.1145/2602144. 1

[GKDS12] Georgiev I., Křivánek J., Davidovič T., Slusallek P.: Light transport simulation with vertex connection and merging. *ACM Trans. Graph. 31*, 6 (Nov. 2012), 192:1–192:10. URL: http://doi.acm.org/10.1145/2366145.2366211, doi:10.1145/2366145.2366211. 1

[HOJ08] Hachisuka T., Ogaki S., Jensen H. W.: Progressive photon mapping. *ACM Trans. Graph. 27*, 5 (Dec. 2008), 130:1–130:8. URL: http://doi.acm.org/10.1145/1409060.1409083, doi:10.1145/1409060.1409083. 1

[IH14] Ibrahim H., Hoo S. C.: Local contrast enhancement utilizing bidirectional switching equalization of separated and clipped subhistograms. *Mathematical Problems in Engineering* (2014). 3

[PBD*10] Parker S. G., Bigler J., Dietrich A., Friedrich H., Hoberock J., Luebke D., McAllister D., McGuire M., Morley K., Robison A., Stich M.: Optix: A general purpose ray tracing engine. *ACM Transactions on Graphics* (August 2010). 1, 3

[Vea98] Veach E.: *Robust Monte Carlo Methods for Light Transport Simulation*. PhD thesis, Stanford University, Stanford, CA, USA, 1998. AAI9837162. 1

[vHM*12] Čadík M., Herzog R., Mantiuk R., Myszkowski K., Seidel H.-P.: New measurements reveal weaknesses of image quality metrics in evaluating graphics artifacts. *ACM Trans. Graph. 31*, 6 (Nov. 2012), 147:1–147:10. URL: http://doi.acm.org/10.1145/2366145.2366166, doi:10.1145/2366145.2366166. 3

[WSB03] Wang Z., Simoncelli E. P., Bovik A. C.: Multiscale structural similarity for image quality assessment. In *in Proc. IEEE Asilomar Conf. on Signals, Systems, and Computers, (Asilomar* (2003), pp. 1398–1402. 3

[WWB*14] Wald I., Woop S., Benthin C., Johnson G. S., Ernst M.: Embree: A kernel framework for efficient cpu ray tracing. *ACM Trans. Graph. 33*, 4 (July 2014), 143:1–143:8. URL: http://doi.acm.org/10.1145/2601097.2601199, doi:10.1145/2601097.2601199. 1, 3