Interactive Hyper Spectral Image Rendering on GPU

Romain Hoarau[*], Eric Coiro[*], Sébastien Thon[‡] and Romain Raffin[‡]

[*]Onera - The French Aerospace Lab
Salon-de-Provence, F-13661 France
phone, email: romain.hoarau14@gmail.com, eric.coiro@onera.fr
www: https://www.onera.fr/en

[‡]Aix Marseille University / Toulon University / CNRS / ENSAM / LIF
Marseille, France
phone, email: sebastien.thon@univ-amu.fr, romain.raffin@univ-amu.fr
www: http://www.lif.univ-mrs.fr/

Abstract

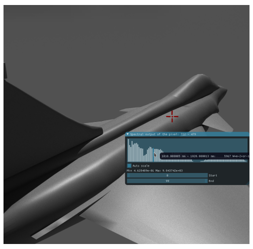
In this paper, we describe a framework focused on spectral images rendering. The rendering of a such image leads us to three major issues: the computation time, the footprint of the spectral image, and the memory consumption of the algorithm. The computation time can be drastically reduced by the use of GPUs, however, their memory capacity and bandwidth are limited. When the spectral dimension of the image will raise, the straightforward approach of the Path Tracing will lead us to high memory consumption and latency problems. To overcome these problems, we propose the DPEPT (Deferred Path Evaluation Path Tracing) which consists in decoupling the path evaluation from the path generation. This technique reduces the memory latency and consumption of the Path Tracing. It allows us to use an efficient wavelength samples batches parallelization pattern to optimize the path evaluation step and outperforms the straightforward approach.

First presented at International Conference on Computer Graphics Theory and Applications 2018, extended and revised for JVRB

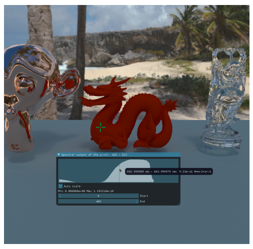
Keywords: Spectral Image Rendering, Global illumination, GPU Computing, Predictive rendering.

1 Introduction

Aircraft detection in the IR (Infrared) and Visible bands is an active research field in the aerospace community. Dimensioning the sensors for this purpose requires the spectral radiance of a scene at their input, that can be expensive and difficult to obtain by airborne measurement campaigns. A fast and accurate simulation (Fig. 1) of these data is thus needed.



(a) 100 channels in the SWIR range.



(b) 400 channels in the Visible range.

Figure 1: The spectral output of a pixel (red point on both images) and its displayable color can be visualized at the same time.

In the IR range, local illumination methods (e.g. [Whi79]) are used to simulate the light transport in most cases for former aircraft generation or supersonic flight where IR signature is overwhelmed by the radiation of hot parts. Nevertheless these methods have

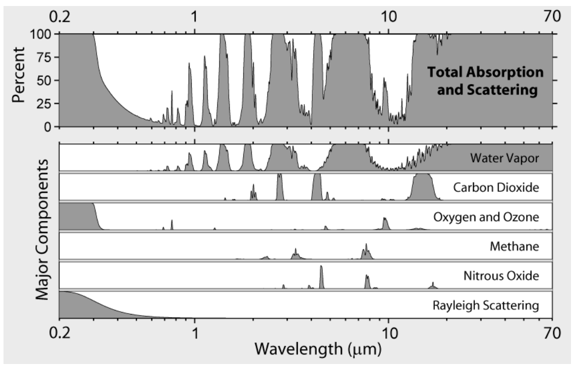


Figure 2: The absorption spectrum of the atmosphere [Roh07] is very spiky.

proved to be insufficient since the new generation of aircraft relies on their shape (e.g. curved nozzle) to hide their hot parts to improve their stealth characteristic. Hereafter, the inter-reflections have to be taken into account. For the aircraft IR signature simulation, the aerospace community has begun to use global illumination methods (e.g. [Coi12]). The typical scene is an aircraft surrounded by an environment map which contains the radiation of the sky and the ground. The scene can also contain the volume of the atmosphere and the plume (aircraft exhaust combustion gases).

The spectral output of this simulation is usually stored in a spectral image composed of a predefined number of fixed wavelengths. At the end of the simulation, the wavelengths of the spectral image are integrated to fit the spectral channels of the sensor.

This method is spectrally biased since the wavelengths are fixed. Therefore it requires a high number of wavelengths to mitigate this bias and especially if the scene contains complex spectral data such as the gases of the atmosphere (Fig. 2) and the aircraft's plume in our case which leads us to important waste of computation time and memory consumption.

In this paper we describe a framework to render directly the channels which removes this spectral bias and drops the spectral dimension of the simulation. Nevertheless, the evolution of technology leads us to explore new concepts of sensors from 380 to 20,000 nm, at high resolution (a few nm of accuracy in visible, and larger in IR). Hence, the number of channels to simulate can be still very high (100 - 400 channels) and problematic since it leads us to three major issues:

- The computation time: For each channel, we need to compute at least one wavelength sample per path.
- The footprint of the spectral image: 4 bytes per channel per pixel in single precision.
- The memory consumption of the algorithm: For instance, the Path Tracing [Kaj86] needs to consume 8 bytes at least (to accumulate the spectral radiance and the path throughput) per wavelength sample per path.

The computation time can be drastically reduced by the use of GPUs, but their memory capacity and bandwidth are limited. The straightforward approach of the Path Tracing would be to evaluate at each bounce the path extension for each wavelength sample.

To be efficient, a path has to be reused by at least one wavelength sample per channel. When the number of channels will raise, the wavelength sample states won't be able to fit into the registers or the local memory. Therefore, they would have to be stored in the global memory which would imply high memory consumption and latency problems.

To overcome these problems, we propose the DPEPT (Deferred Path Evaluation Path Tracing) which consists in decoupling the path evaluation from the path generation. At each bounce, the bare minimum information is stored in a path vertex. Once the path is completed, the wavelength sample of each channel is evaluated. Since the path vertices are saved, the channels can be evaluated per batch of wavelength sample to keep in check the memory consumption. The state of this batch can be stored in the local memory to reduce latency. This method allows us to optimize the path evaluation and outperforms the straightforward approach when the spectral resolution of the simulated image raises. Our approach enables to render efficiently multi, hyper and even ultra (more than 1,000 channels) spectral image.

Our research are beneficial to some Computer Graphics fields such as the predictive rendering and the colorimetric validation. The main contributions of this paper are:

- A framework to render directly a spectral image composed with K number of channels (Sec. 3).
- A suitable path tracing method (DPEPT) to render a high spectral dimension image on GPU (Sec. 5.1).
- An efficient wavelength parallelization pattern of the path evaluation step (Sec. 5.2).

2 Previous works

2.1 GPU implementation of the Path tracing

The Path Tracing [Kaj86] is the simplest global illumination algorithm. Its implementation on GPU was first studied by [PBMH02] which proposed a multi pass approach due to the lack of programmability of the early hardware.

One of the main problems at this time was the various lengths of the path which implied an inefficient workload and code divergence. Few threads process the long paths while other threads are idle. To solve this problem, [NHD10] introduced the Regenerative Path Tracing which decouples path from pixel by using the thread persistent technique. If a thread is idle it generates a new path instead of waiting.

To improve the Regenerative Path Tracing, [vA11] proposed the Streaming Path Tracing which compacts the regenerated threads to increase their coherence for their primary closest intersection tests.

Later, [LKA13] highlighted the register pressure problem of the mega kernel approach (one single kernel devoted to the algorithm), especially when complex materials are used. They introduced the Wavefront Path Tracing, which splits the algorithm in different kernels. In order to reduce the register spilling and to improve the coherence, a kernel was dedicated to each material. This implementation showed a performance gain with scene made of complex materials. However, for a scene containing simple materials, the sorting step (to dispatch the workload to each kernel) and the kernel launching overhead neutralized the performance gain.

[DKHS14] made an exhaustive GPU implementation survey of the Path Tracing. They proposed new approaches and benchmarked the different implementations. The Multi Kernel Streaming Path Tracing was the most performance portable implementation.

2.2 Spectral Rendering

To correctly simulate the light transport, Spectral Rendering is mandatory. There are two classes of approach:

- The Predefined Wavelengths Methods.
- The Spectral Monte-Carlo Methods.

2.2.1 Predefined Wavelengths Methods

The principle of this approach is to render the scene with a pre-fixed wavelength set $\{\lambda_1, \cdots, \lambda_k\}$:

$$P = \int_{\omega\in\Omega} \begin{bmatrix} f(\omega,\, \lambda_1) \\ \vdots \\ f(\omega,\, \lambda_k) \end{bmatrix} \mathrm{d}\omega \tag{1}$$

A low spectral dimension simulation would miss important wavelengths which would lead to large systematic errors. Several authors [ZCB98][WTP00][CW05] tried to mitigate this bias by carefully choosing the set of wavelengths. However, this set of wavelengths is very scene dependant and difficult to be known a priori. Moreover, it is still inefficient when the scene contains complex spectral input data. In that case, the number of wavelength would still have to be very high.

Therefore, the spectral dimension of the simulation with this kind of method is usually high to mitigate the spectral bias which would leads to high memory consumption and computation time.

Even with this downside, this method is still commonly used in the Optic community since it is the only one which allows to keep the spectral output of the simulation.

2.2.2 Spectral Monte-Carlo Methods

To overcome this drawback, the Computer Graphics community has developed a spectral sampling approach. Instead of setting the wavelengths, a spectral dimension is added to the Monte-Carlo integration to compute the tristimulus XYZ:

$$XYZ = \int_{\omega\in\Omega} \int_{380nm}^{780nm} \begin{bmatrix} f_X(\omega,\, \lambda) \\ f_Y(\omega,\, \lambda) \\ f_Z(\omega,\, \lambda) \end{bmatrix} \mathrm{d}\lambda\, \mathrm{d}\omega \tag{2}$$

Thanks to this approach, a high number of wavelengths per path is not required anymore which reduces the memory consumption and the computation time.

To improve the convergence rate, [EM99] proposed to reuse a path for a cluster of wavelengths sampled by the lights of the scene.

[RBA09] improved this technique by introducing a spectral sampling during the path generation. In order to reduce the variance, the authors suggested to use a MIS (Multi Importance Sampling) [VG95] to take into account the different probability density functions of the wavelength carried by a path.

[WND+14] clarified the use of multiple wavelengths per path and the MIS method. They proposed a simple but optimized approach. The cluster of wavelengths is generated by one wavelength (named “hero wavelength” by the authors) by applying a regular offset in order to cover the visible range.

The major problem of this kind of method for the Optic community is that the spectral output of the simulation is lost since the XYZ tristimulus is directly computed.

3 Spectral Image Rendering Framework

3.1 The inputs

The inputs of simulation are spectral data (reflectance, complex index of refraction, environment map, ...). They can be stored as a tabulated spectrum, but the bounds retrieval of a query (wavelength sample) to compute the interpolated value requires the use of a binary search. At the moment, we use regular spaced spectrum which gives us a good performance since we can easily retrieve the bounds. The number of samples of each spectrum is entirely dynamic which gives us a good accuracy if needed.

3.2 The outputs

The spectral output of the simulation must be stored in a suitable data structure. An efficient and robust data structure for these data is an open problem, further research has to be done. The most straightforward approach is the use of a spectral image composed with K numbers of channels, nevertheless it is memory-wise (3d image) and computation-wise (each channel needs to be processed) inefficient. A channel is not a wavelength but the integration of the spectral radiance over a spectral interval. The spectral radiance is usually weighted by a sensor response curve before the integration. By the way, the XYZ triplet can be seen as three overlapping channels over the Visible range which integrate the same wavelength samples with different sensor response curve (CIE XYZ curves).

Depending upon the user needs, we propose two pipeline for the simulation:

- Pre-integration sensor curve weighting: The spectral radiance is weighted by a sensor response curve before the integration. In order to produce a color image (XYZ, RGB, false color, grayscale...), the wavelength samples can be reused to integrate the color channels at nearly zero cost. In this case, if we change the sensor curve, the simulation have to be reseted.

- Post-integration sensor curves weighting: If a large number of sensors have to be evaluated for the same scene, it would be inefficient to recompute the spectral image for each sensor.In order to reuse the spectral output later, the spectral radiance of the channels are not integrated with a response curve of a sensor. In this case, the response curves have to be considered constant in the spectral interval of the simulated channel to be able later to integrate these data correctly. Otherwise, we need smaller channels to integrate them with a piece-wise constant function which corresponds to the response curve in the sensor channel. If we change the sensor curve, the simulation don’t have to be reseted, since the spectral radiation of a scene at the input of a sensor is refined in a spectral image. To display a color image, the spectral image can be used at any moment to integrate the color channels.

An interesting feature of this kind of rendering is to be able to visualize in real time the spectrum output of a pixel of the image (Fig. 1).

3.3 The light transport equation reformulation

In order to render a spectral image of K number of channels, the spectral domain of the light transport equation has to be discretized. We need to compute a pixel of a such image according to the equation below:

$$P = \begin{bmatrix} \int_{\lambda min_1}^{\lambda max_1} \int_{\omega \in \Omega} f_1(\omega,\, \lambda)\, \mathrm{d}\omega\, \mathrm{d}\lambda \\ \vdots \\ \int_{\lambda min_k}^{\lambda max_k} \int_{\omega \in \Omega} f_k(\omega,\, \lambda)\, \mathrm{d}\omega\, \mathrm{d}\lambda \end{bmatrix} \tag{3}$$

This approach is however inefficient especially if an image has to be computed with a very large number of channels, since a path can carry multiple wavelengths for the most of materials. Therefore, to be efficient, a same path has to carry at least one wavelength sample per channel. Otherwise, more paths would be required to refine the spectral image.

Moreover, the spectral intervals are not necessarily adjacent and their size are not always equal, hence we cannot use the hero wavelength cluster generation of [WND+14] to avoid the generation cost and the storage of a wavelength sample per channel per path. Instead of that, we generate a unique uniform sample μ between 0 and 1 per path and we use this linear interpolation mapping function between the bounds of each channel to retrieve their wavelength sample:

$$\Lambda_k(\mu) = (1-\mu)\,\lambda min_k + \mu\,\lambda max_k \tag{4}$$

Using the previous statements, we can rewrite the Eq. 3:

$$\begin{aligned} P &= \int_{\omega\in\Omega} \begin{bmatrix} \int_{\Lambda_1(0)}^{\Lambda_1(1)} f_1(\omega,\,\lambda)\,\mathrm{d}\lambda \\ \vdots \\ \int_{\Lambda_k(0)}^{\Lambda_k(1)} f_k(\omega,\,\lambda)\,\mathrm{d}\lambda \end{bmatrix} \mathrm{d}\omega \\ &= \int_0^1 \int_{\omega\in\Omega} \begin{bmatrix} f_1(\omega,\,\Lambda_1(\mu))\,\Lambda_1'(\mu) \\ \vdots \\ f_k(\omega,\,\Lambda_k(\mu))\,\Lambda_k'(\mu) \end{bmatrix} \mathrm{d}\omega\,\mathrm{d}\mu \end{aligned} \tag{5}$$

This equation can be solved by the Monte-Carlo method:

$$\langle\, P\,\rangle \approx \frac{1}{N}\sum_{i=1}^{N} \begin{bmatrix} \frac{f_1(\omega_i,\Lambda_1(\mu_i))\,\Lambda_1'(\mu_i)}{p_1(\omega_i,\mu_i)} \\ \vdots \\ \frac{f_k(\omega_i,\Lambda_k(\mu_i))\,\Lambda_k'(\mu_i)}{p_k(\omega_i,\mu_i)} \end{bmatrix} \tag{6}$$

3.4 The path samples generation

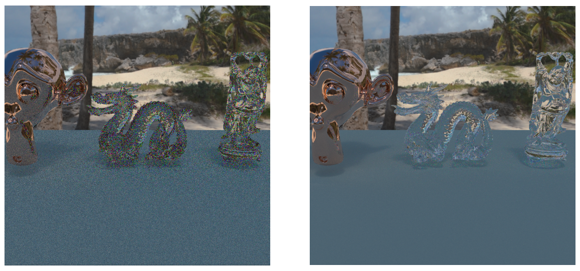
The viability of the spectral sampling methods [WND+14] for paths carrying a lot of wavelength samples needs to be investigated. Therefore, at the moment, our path decisions (extension or termination) are based on either the importance sampling of the mean value of the input spectral data (materials and the light) or an uniform sampling. Since it's not wavelength dependent, the probability density function $p_k(\omega_i,\,\mu_i)$ can be reduced to:

$$\begin{aligned} p_k(\omega_i,\,\mu_i) &= p(\mu_i)\,p(\omega_i|\Lambda_k(\mu_i)) \\ &= p(\omega_i) \end{aligned} \tag{7}$$

We can now rewrite the Eq. 6 as:

$$\langle\, P\,\rangle \approx \frac{1}{N}\sum_{i=1}^{N}\frac{1}{p(\omega_i)} \begin{bmatrix} f_1(\omega_i,\,\Lambda_1(\mu_i))\,\Lambda_1'(\mu_i) \\ \vdots \\ f_k(\omega_i,\,\Lambda_k(\mu_i))\,\Lambda_k'(\mu_i) \end{bmatrix} \tag{8}$$

If an ideal refraction is encountered during the propagation (Fig. 3), we use the degradation strategy [EM99]. This strategy chooses randomly one wavelength to continue the propagation and stops the others. In this case, the throughput of the chosen wavelength sample has to be scaled by the number of wavelength samples carried by the path.



(a) 16 spp. (b) 1,024 spp.

Figure 3: Refraction with degradation of paths which carry 64 wavelength samples at different spp (number of sample per pixel).

4 The straightforward approach of the Path Tracing on GPU

Our GPU methods are based on the Streaming Path Tracing [vA11]. The Multi Kernel implementation is chosen because the performances are more portable amongst the different GPU Vendors [DKHS14]. Moreover, most of the intersection libraries on GPU (Optix Prime [NVI], RadeonRays [AMD] ...) do not provide an intersection device function which would allow us to integrate it in a Mega Kernel. So we need to split at least some parts of the algorithm to communicate with these libraries.

4.1 Algorithm

The straightforward approach would be to extend and evaluate the path at each bounce. In this case, the states of every wavelength sample of each path have to be kept alive, since the full path is not known and some values (spectral radiance and path throughput) have to be accumulated.

To be efficient, a path has to be reused by at least one wavelength sample per channel. When the number of channels will raise, this approach would lead us to high memory consumption and latency problems, since the states of each wavelength sample have to be updated from the global memory at each bounce.

It should be noted that a Mega Kernel implementation does not solve these problems since the wavelength sample states won't be able to fit into the registers or the local memory. The only possibility is to store them in the global memory.

4.2 The memory consumption

The memory consumption of this method is proportional to the number of channels (Fig. 4) since we need to store for each thread:

- The path state which weights 32 bytes composed by :
  - The idle state.
  - The current depth.
  - The pixel index.
  - The ray index.
  - The random number generator.
  - The spectral sample μ.
  - The last material probability density function (For MIS).
  - The selected channel index in case of refraction.
- For each channel, the wavelength sample state which weights 16 bytes (4 padding bytes):
  - The throughput.
  - The spectral radiance.
  - The direct light contribution (For a Multi Kernel implementation, we need to add it later to the spectral radiance if the shadow ray is not occluded).

This is problematic because it limits the number of channels and the size of the pool threads. We saw during our experiments that a size between $1,024^2$ and $2,048^2$ threads is required to reach the peak of performance. This performance gain of a larger pool of

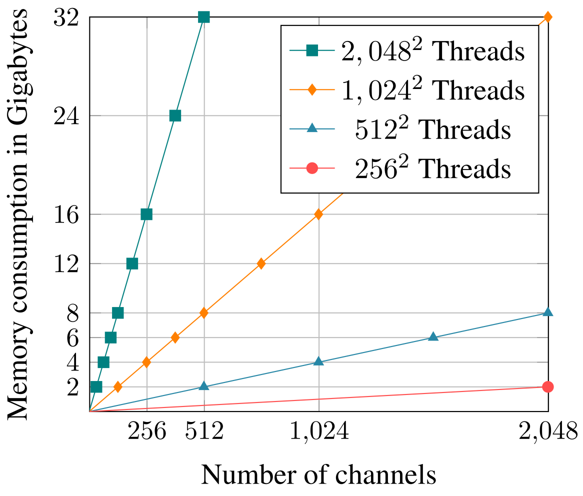


Figure 4: Memory consumption of the Path Tracing for different threads pool sizes.

thread is due to the mitigation of the host kernel launch overhead and the ability to compute a packet of ray per pixel (which increase the coherency of intersection computation and shading). As we can see with the Fig. 4, a pool of $1,024^2$ threads would quickly limit the number of channels (e.g. 512 channels for around 8.2 GB of memory footprint).

5 DPEPT: Deferred Path Evaluation Path Tracing

5.1 Algorithm

The DPEPT consists in two following steps:

1. The building and the saving of the full path.
2. The evaluation of the wavelength samples of the saved paths.

To reduce the memory consumption, the wavelength samples are evaluated per batch by reloading their path (Fig. 5). Since registers are scarce and precious resources (and they have to be spared for the path evaluation), the state of a batch are stored in the local memory to reduce latency (Algorithm 1).

The batch size is determined by the amount of available local memory for the work-group and the number of channels. The local memory size depends of the work-group size, the hardware capability and the register pressure of the path evaluation step (since we don't want to further decrease the occupancy by increasing the local memory size).

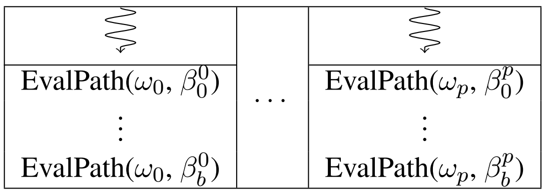

EvalPath(ω_0, β_0^0)
⋮
EvalPath(ω_0, β_b^0)
. . .
EvalPath(ω_p, β_0^p)
⋮
EvalPath(ω_p, β_b^p)

Figure 5: Path parallelization pattern of a work-group (one path per work-item) for the paths evaluation. ⥯ is a work-item (thread of the work-group). EvalPath(ω_p, β_b^p) is the evaluation of the path sample p for its wavelength sample batch b (Algorithm 1).

5.2 Spectral Parallelism

Although the DPEPT was better than the previous method, the performance was not enough sufficient. To improve the path evaluation step of the DPEPT, a new parallelization pattern (. refWavelengthParallelizationPattern) has been developed.

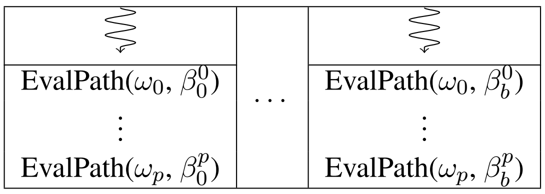

EvalPath(ω_0, β_0^0)
⋮
EvalPath(ω_p, β_0^p)
. . .
EvalPath(ω_0, β_b^0)
⋮
EvalPath(ω_p, β_b^p)

Figure 6: Spectral parallelization pattern of a work-group (wavelength samples batches of different paths per work-item) for the paths evaluation. ⥯ is a work-item (thread of the work-group). EvalPath(ω_p, β_b^p) is the evaluation of the path sample p for its wavelength sample batch b (Algorithm 1).

Instead of processing a path per work-item (Fig. 5), the whole work-group processes different wavelength samples batch of the same path at the same time. This way, the work-group instruction coherence is nearly perfect. And since the work-items read exactly the same path, the cache miss decreases and we can benefit from the broadcast feature if it's available.

However, in order to be fully utilized, the work-group needs to have enough batch. On AMD hardware (16-wide SIMD units), when the number of channels is less than 16, the version 1 (Fig. 5) is better than the version 2 (Fig. 6).

An hybrid scheme is a tempting solution, but in practice it's tricky to implement. For instance, in order to accumulate (reduction operation) the XYZ color across the work-items at the end of the path evaluation kernel, we will need to manage the case where there are wavelength batches with different paths inside the same work-group.

5.3 The memory consumption

The memory consumption of this implementation is dependent on the maximum number of bounces of a path (Fig. 7) since we need to store for each thread:

- The path state which weights 32 bytes like the previous method.
- A number of vertex which each weights 80 bytes (8 padding bytes) composed by:
  - An integer to store some flags (if a shadow ray was not occluded...).
  - The material ID.
  - The light ID.
  - The direction to the light.
  - The light sample's probability density function (The cosinus term and the MIS weight are packed in the PDF).
  - The outgoing direction to the light (BSDF space).
  - The ingoing direction (BSDF space).
  - The outgoing direction (BSDF space).
  - The surface self emission MIS weight.
  - The material sample's probability density function (The cosinus term and the MIS weight are packed in the PDF).

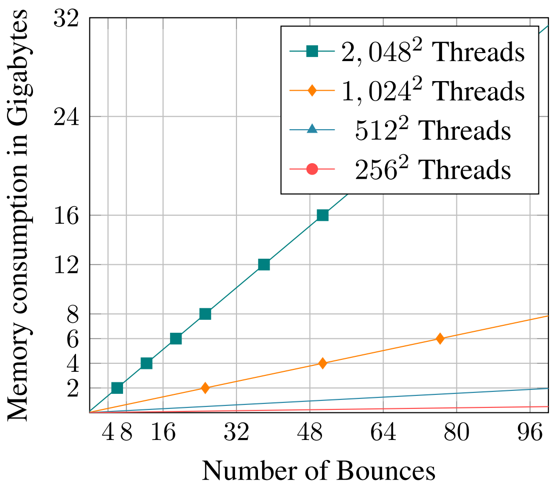


Figure 7: Memory consumption of the DPEPT implementation for different thread pool sizes.

GPUs don't mostly support dynamic allocation on the device side. And when they do, it's strongly advised to not use this feature. Therefore for each thread, a maximum of path vertex has to be preallocated on the host side. Hopefully we do not need as much bounce as channel. Therefore, regardless the number of channels, a pool of $1,024^2$ threads with 32 bounces will consume for around 2.6 GB. Although the size of a path vertex will increase when complex materials (texture, volume...) will be added, there is still plenty of room before reaching the level of memory consumption of the straightforward approach.

6 Results

To compare the PT (conventional Path Tracing) and the DPEPT methods, a benchmark (Fig. 8) has been made. It consists in measuring the number of paths completed per second (raw performance) and the FPS (interactivity) at different thread pool sizes and number of channels.

The benchmark (Fig. 8) has been run on the following scenes:

- Cornell box: The maximum depth of this easy indoor scene has been fixed at 5 bounces.
- Conference: The maximum depth of this medium-complex indoor scene has been fixed at 8 bounces.
- Aircraft: An easy outdoor scene which is typically used to dimension aircraft detection sensors. The scene contains a spectral environment light generated by MATISSE [SCF+06]. The maximum depth has been fixed at 5 bounces.
- Statues: A medium-complex outdoor scene which contains a refractive object and a spectral environment light converted from a rgb hdr image. The maximum depth has been fixed at 16 bounces.

Except Suzanne (Blender) and the Aircraft (ONERA), the other meshes come from [McG17].

For a $1,024^2$ pool of thread with 16 bounces on a Fury X (4GB of VRAM), a full HD image would limit the number of channels to around 220 for the DPEPT and to around 127 for the PT. Therefore, in order to be not limited by the memory consumption of the spectral image, the spatial image resolution was fixed to 512x512.

When the number of channels is under 8, the performance of our method is usually a bit subpar compared to the Path Tracing. However, when the number of channels raises, the Path Tracing is considerably slower than our method.

Due to its lower memory consumption, the DPEPT can render more channels than the straightforward Path Tracing approach which limits the size of the thread pool.

For instance if we render a image with 512 spectral channels, the Path Tracing won't be able to allocate a thread pool size of $1,024^2$ on the Fury X (4GB of VRAM). Our technique can easily reach the size of $1,024^2$ threads and it is thus faster by around 38 times on average (Fig. 9).

The results of the benchmark show us that our method allows us to render at interactive frame rate (around 10 FPS) a hyper spectral image (between 100 and 200 channels) at low spatial resolution.

7 Technical details

We use SYCL to implement our rendering engine. SYCL is the new royalty-free Khronos Group standard that allows to code in OpenCL in a single C++ source fashion. It lowers the burden of programmer by managing the data movement transaction between the host and device. Thus, the programmer can focus on the optimization of his kernels. At the moment there are three implementations of the standard: ComputeCpp [Cod19], TriSYCL [Xil19] and SYCL-GTX [Žuž19]. We use the beta of the Community edition of ComputeCpp, since it is the most advanced implementation at the time of this publication.

To compute the ray and shadow ray intersections, we use RadeonRays. It is an AMD open source library for ray tracing. Like Optix Prime, we need to provide a ray buffer, and the library returns a hit point buffer. The library can use different back-ends: Vulkan, OpenCL and Embree. We have chosen the OpenCL back-end since we can use it via the interoperability feature of SYCL.

8 Conclusion

In this article, we endeavour to simulate efficiently and accurately spectral images (Fig. 1). A new framework

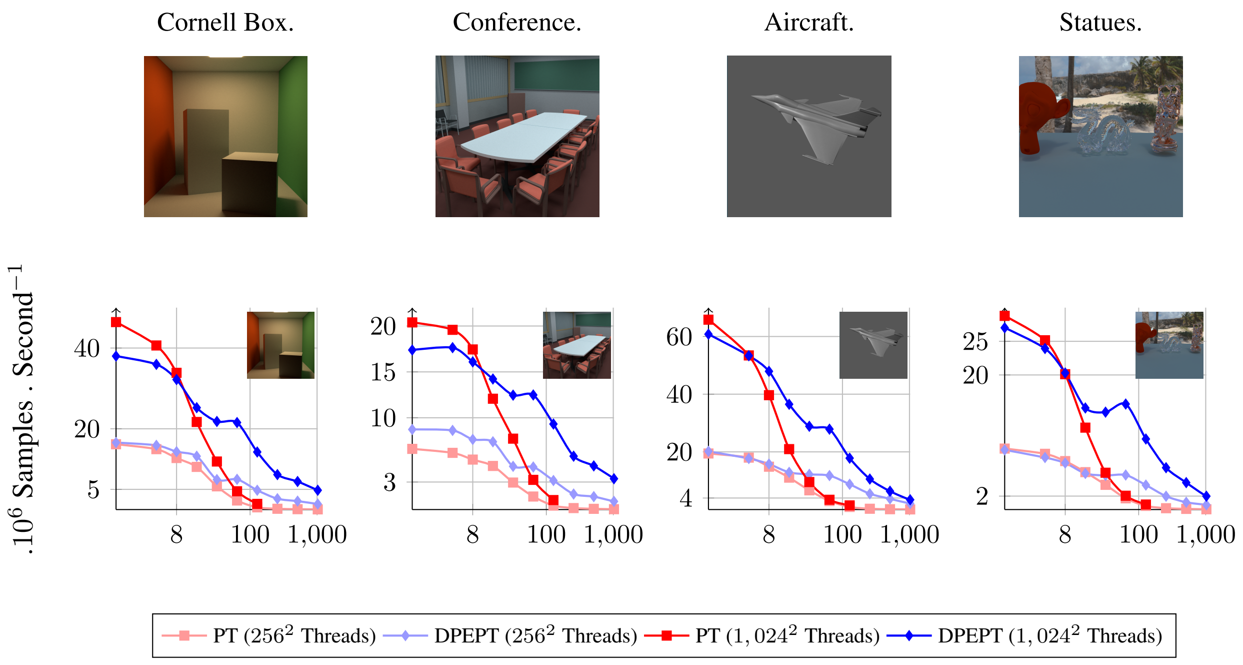


Figure 8: Benchmark with an AMD Fury X (VRAM: 4GB). The scaling of the x-axis is a logarithm base 2.

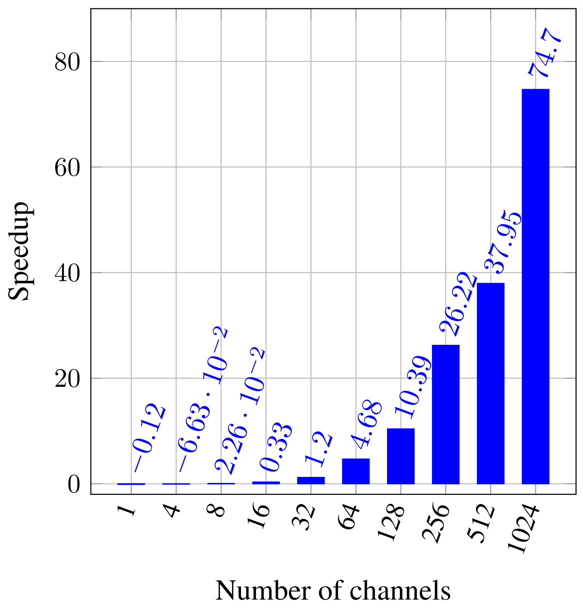


Figure 9: Average DPEPT speedup of the performance over the PT for the benchmark (Fig. 8).

(Sec. 3) has been described to render a spectral image composed with K number of channels.

The rendering of a such spectral image will lead us to three major issues: the computation time, the footprint of the spectral image, and the memory consumption of the algorithm. The computation time can be drastically reduced by the use of GPUs, however, their memory capacity and bandwidth (compared to their compute power) are limited. When the number of channels will raise, the straightforward method (Sec. 4) will lead us to high memory consumption and latency problems.

To overcome these problems, we propose the DPEPT (Sec. 5.1) which consists in decoupling the path evaluation from the path generation. The memory consumption of this approach is not dependent on the number of channels. Its path evaluation step can be efficiently parallelized (Sec. 5.2). Our method outperforms the straightforward approach when the spectral resolution of the simulated image raises. Our contributions enable to render multi, hyper and even ultra (more than 1,000 channels) spectral image. Interactive frame rate of hyper spectral image rendering can be achieved for easy and medium complex scene at low spatial resolution.

However we think there are still room to improve the compute time and the convergence of the simulation. Furthermore, the footprint of the spectral image problem is not yet solved, therefore it can limit the simulation if the spectral image is too big to fit in the global memory of a GPU.

9 Future work

The possible future work would consist in:

- Improving the compute time: we have made the assumption that it's needed to compute one wavelength sample per channels per path, maybe we can find a better tradeoff between the number of wavelength samples to evaluate per path and the number of path to trace.
- Investigating the feasibility of the spectral MIS [WND+14] for paths which carry a large number of wavelength samples on GPU: it would improve the convergence of the algorithm when using high wavelength dependent materials.
- Mitigating the spectral image footprint problem: The footprint of the spectral image can limit the simulation if it is too big to fit in the global memory of a GPU. We are currently prospecting the two following solutions:
    - A Multi buffering technique to refine the spectral image in out-of-core fashion by alternatively overlapping the transfers and the computations of two small buffer.
    - A Multi GPU approach where the spectral image and its computation are split spatially between GPUs. The color image is merged at each iteration of the simulation but the spectral image is merged only by the user demand (to save the spectral image for instance).

  Another approach would be to work on an efficient and compact data structure to refine the spectral output (wavelet-like, polynomial-like, ...). But as stated above it is not an easy problem.

Acknowledgements

This work has been funded by the Provence-Alpes-Côte d'Azur (PACA) French region and ONERA.

Algorithm 1: Evaluation of the path ω for the wavelength batch β.

```
Local Memory:
$L$ : Spectral radiances of the batch $\beta$.
$\tau$ : Path throughputs of the batch $\beta$.
Global Memory:
$V$ : Saved Paths vertices.
$I$ : Spectral Image.
Function (At the vertex $v$ for the wavelength $\lambda$):
$Env(v, \lambda)$ : Environment light contribution.
$NEE(v, \lambda)$ : Direct light contribution.
$Le(v, \lambda)$ : Self emission of the surface.
$\mathcal{T}(v, \lambda)$ : Path throughput.
Function EvalPath($\omega$, $\beta$):
    // Initialize the batch
    foreach $\lambda \in \beta$ do
        $L[\lambda] = 0$
        $\tau[\lambda] = 1$
    end

    // Evaluation of the batch
    foreach $v \in V[\omega]$ do
        if $v$ is a miss vertex then
            // Environment light
            foreach $\lambda \in \beta$ do
                $L[\lambda]$ += $\tau[\lambda]$ * $Env(v, \lambda)$
            end
        else
            foreach $\lambda \in \beta$ do
                // Next event estimation
                $L[\lambda]$ += $\tau[\lambda]$ * $NEE(v, \lambda)$
                // Surface self emission
                $L[\lambda]$ += $\tau[\lambda]$ * $Le(v, \lambda)$
                // Update the throughput
                $\tau[\lambda]$ *= $\mathcal{T}(v, \lambda)$
            end
        end
    end

    // Refine the spectral image
    foreach $\lambda \in \beta$ do
        Refine($I[\omega.pixelID]$, $L[\lambda]$)
    end
return
```

References

[AMD] AMD, *Radeonrays*, https://github.com/GPUOpen-LibrariesAndSDKs/RadeonRays$_SDK$, Last visited January 21th, 2019.

[Cod19] Codeplay, *Computecpp*, https://www.codeplay.com/products/computesuite/computecpp, 2019, Last visited January 21th, 2019.

[Coi12] E. Coiro, *Global Illumination Technique for Aircraft Infrared Signature Calculations*, Journal of Aircraft **50** (2012), no. 1, 103–113, ISSN 0021-8669, DOI 10.2514/1.C031787.

[CW05] Jin R. Chern and Chung Ming Wang, *A novel progressive refinement algorithm for full spectral rendering*, Real-Time Imaging **11** (2005), no. 2, 117–127, ISSN 1077-2014, DOI 10.1016/j.rti.2005.01.004.

[DKHS14] Tomáš Davidovič, Jaroslav Křivánek, Miloš Hašan, and Philipp Slusallek, *Progressive Light Transport Simulation on the GPU: Survey and Improvements*, ACM Transactions on Graphics **33** (2014), no. 3, 1–19, ISSN 0730-0301, DOI 10.1145/2602144.

[EM99] Glenn F. Evans and Micheal D. McCool, *Stratified wavelength clusters for efficient spectral Monte Carlo rendering*, Proceedings of Graphics Interface '99, Canadian Human-Computer Communications Society, 1999, DOI 10.20380/GI1999.07, pp. 42–49, ISBN 0-9695338-8-8.

[Kaj86] James T. Kajiya, *The Rendering Equation*, ACM SIGGRAPH Computer Graphics **20** (1986), no. 4, 143–150, ISSN 0097-8930, DOI 10.1145/15886.15902.

[LKA13] Samuli Laine, Tero Karras, and Timo Aila, *Megakernels Considered Harmful: Wavefront Path Tracing on GPUs*, High Performance Graphics, Association for Computing Machinery, 2013, ISBN 9781450321358.

[McG17] Morgan McGuire, *Computer graphics archive*, https://casual-effects.com/data, July 2017, Last visited January 21th, 2019.

[NHD10] Jan Novák, Vlastimil Havran, and Carsten Dachsbacher, *Path Regeneration for Interactive Path Tracing*, Eurographics, 2010.

[NVI] NVIDIA, *Optix prime*, https://developer.nvidia.com/optix, Last visited January 21th, 2019.

[PBMH02] Timothy J. Purcell, Ian Buck, William R. Mark, and Pat Hanrahan, *Ray tracing on programmable graphics hardware*, ACM Transactions on Graphics **21** (2002), no. 3, 703–712, ISSN 0730-0301, DOI 10.1145/566654.566640.

[RBA09] Michal Radziszewski, Krzysztof Boryczko, and Witold Alda, *An improved technique for full spectral rendering*, Journal of WSCG **17** (2009), no. 1–3, 9–16, ISSN 0014-2956.

[Roh07] Robert A. Rohde, *Atmospheric transmission*, https://en.wikipedia.org/wiki/File:Atmospheric$_Transmission.png$, 2007, Last visited January 21th, 2019.

[SCF$^+$06] Pierre Simoneau, Karine Caillault, Sandrine Fauqueux, Thierry Huet, Jean Claude Krapez, Luc Labarre, Claire Malherbe, and Christophe Miesch, *MATISSE: Version 1.4 and future developments*, Optics in Atmospheric Propagation and Adaptive Systems IX, vol. 6364, SPIE, 2006, DOI 10.1117/12.694964.

[vA11] Dieter van Antwerpen, *Improving SIMD Efficiency for Parallel Monte Carlo Light Transport on the GPU*, High Performance Graphics, Association for Computing Machinery, 2011, DOI 10.1145/2018323.2018330, pp. 41–50, ISBN 9781450308960.

[VG95] Eric Veach and Leonidas J. Guibas, *Optimally combining sampling techniques for Monte Carlo rendering*, Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '95, Association for Computing Machinery, 1995, DOI 10.1145/218380.218498, pp. 419–428.

[Whi79] Turner Whitted, *An improved illumination model for shaded display*, ACM SIGGRAPH Computer Graphics **13** (1979), no. 2, 14, ISSN 0097-8930, DOI 10.1145/965103.807419.

[WND+14] Alexander Wilkie, Sehara Nawaz, Marc Droske, Andrea Weidlich, and Johannes Hanika, *Hero Wavelength Spectral Sampling*, Computer Graphics Forum **33** (2014), no. 4, 123–131, ISSN 0167-7055, DOI 10.1111/cgf.12419.

[WTP00] Alexander Wilkie, Robert Tobler, and Werner Purgathofer, *Raytracing of Dispersion Effects in Transparent Materials*, Journal of WSCG **8** (2000), no. 1–3, ISSN 1213-6972.

[Xil19] Xilinx, *Trisycl*, https://github.com/triSYCL/triSYCL, 2019, Last visited January 21th, 2019.

[ZCB98] Eric Zeghers, Samuel Carré, and Kadi Bouatouch, *Error-bound wavelength selection for spectral rendering*, The Visual Computer **13** (1998), no. 9–10, 424–434, ISSN 1432-2315, DOI 10.1007/s003710050115.

[Žuž19] Peter Žužek, *Sycl-gtx*, https://github.com/ProGTX/sycl-gtx, 2019, Last visited January 21th, 2019.

Citation

Romain Hoarau, Eric Coiro, Sébastien Thon, Romain Raffin, *Interactive Hyper Spectral Image Rendering on GPU*, Journal of Virtual Reality and Broadcasting, 15(2018), no. 4, February 2019, DOI 10.48663/1860-2037/15.2018.4, ISSN 1860-2037.