Real-Time Importance Sampling of Dynamic Environment Maps

H. Lu[1†] R. Pacanowski[1‡] X. Granier[1§]
Inria - Univ. Bordeaux CNRS, LP2N IOGS, LP2N
1. Inria Bordeaux Sud-Ouest - LP2N (Univ. Bordeaux, IOGS, CNRS) - LaBRI (Univ. Bordeaux, CNRS)

Figure 1: *Time-varying light samples distribution for one pixel (cyan dot) on the dragon model when lit with a dynamic environment map [STJ*04]. In the cube map images, the cyan dot corresponds to the normal direction. For each frame, among on-the-fly generated samples (red+yellow+green) on the whole environment map, our technique selects* 50 *of them (yellow and green dots). The green dots correspond to effective samples where the cosine factor is strictly positive. At nearly noon* ***(Left)****, our technique selects less samples on the sun than at sunset* ***(Right)****, since it is located at a more grazing angle. This example runs in average at 145 fps using Multiple Importance Sampling with* 50 *samples for the energy conserving Lafortune Phong BRDF with a shininess exponent set to* 150*.*

Abstract

We introduce a simple and effective technique for light-based importance sampling of dynamic environment maps based on the formalism of Multiple Importance Sampling (MIS). The core idea is to balance per pixel the number of samples selected on each cube map face according to a quick and conservative evaluation of the lighting contribution: this increases the number of effective samples. In order to be suitable for dynamically generated or captured HDR environment maps, everything is computed on-line for each frame without any global preprocessing. Our results illustrate that the low number of required samples combined with a full-GPU implementation lead to real-time performance with improved visual quality. Finally, we illustrate that our MIS formalism can be easily extended to other strategies such as BRDF importance sampling.

1. Motivation

Captured HDR environment maps participate, as natural light sources, to the realism of synthetic images. The resulting direct illumination at position p with normal $\boldsymbol{n}$ is:

$$L(p,\boldsymbol{o}) = \int_{\Omega} \mathrm{s}(p,\boldsymbol{n},\boldsymbol{o},\boldsymbol{\omega})\,\mathrm{d}\boldsymbol{\omega} \qquad \text{with}$$
$$\mathrm{s}(p,\boldsymbol{n},\boldsymbol{o},\boldsymbol{\omega}) = \rho(\boldsymbol{o},\boldsymbol{\omega})\,\langle \boldsymbol{n},\boldsymbol{\omega}\rangle\, V(p,\boldsymbol{\omega})\,L(\boldsymbol{\omega})$$

where Ω is the full unit sphere, $\rho(\boldsymbol{o},\boldsymbol{\omega})$ is the reflectance function (BRDF), $<,>$ the positive clamped dot product operator (aka *cosine factor*), $V(p,\boldsymbol{\omega})$ is the visibility function (we will omit it in this paper) and, $L(\boldsymbol{\omega})$ represents the radiance emanating from the direction $\boldsymbol{\omega}$. The size of the texture representing the environment map prohibits achieving real-time results with a brute force approach. This becomes even more obvious if $L(\boldsymbol{\omega})$ is dynamic (i.e., on-line generated or acquired [HSK*05]).

Precomputed Radiance Transfer (PRT) techniques (e.g., [WTL06]) improve rendering performance and integrate complex BRDFs and inter-reflections. However, the required memory to store the precomputed data can quickly become a bottleneck. Furthermore, the costly precomputation time prevents using PRT with dynamic environment maps [HSK*05]. Another trend is to use the general Monte-Carlo framework where the reflected radiance $L(p,\boldsymbol{o})$ is approximated by N_s samples:

$$L(p,\boldsymbol{o}) \approx \frac{1}{N_s}\sum_{i=1}^{N_s} \frac{\mathrm{s}(p,\boldsymbol{n},\boldsymbol{o},\boldsymbol{\omega}_i)}{\mathrm{pdf}(\boldsymbol{\omega}_i)} \qquad (1)$$

where $\mathrm{pdf}(\boldsymbol{\omega}_i)$ is the Probability Density Function (PDF). The closer it matches the numerator of Equation (1) the

† e-mail: heqi.lu@inria.fr
‡ e-mail: romain.pacanowski@institutoptique.fr
§ e-mail: xavier.granier@institutoptique.fr

DOI: 10.2312/conf/EG2013/short/065-068

lower the number of samples N_s will be required to converge to the solution.

In this paper, we focus on light importance sampling of dynamic environment maps (i.e., pdf($\boldsymbol{\omega}$) is proportional to $L(\boldsymbol{\omega})$). More precisely, we introduce a real-time GPU importance sampling technique that recomputes for each frame a tabulated version of the Cumulative Distribution Function (CDF) of an environment map. To generate the light samples $\{\boldsymbol{\omega}_i\}$, the CDF is inverted through a binary search before the shading pass. Our work can be applied to any kind of environment map (spherical, hemispherical, longitude-latitude) but is particularly interesting for cube maps that are widely used for real-time applications. Furthermore, to improve rendering performance, we also introduce an unbiased Monte-Carlo estimator that:

- limits the number of useless light samples that would have been null due to the cosine factor
- reduces popping artifacts that occur when using a low number of samples (this property is particularly important when using time-dependent light sources)
- integrates easily with Multiple Importance Sampling (MIS).

In the context of light importance sampling, many techniques (e.g., [ODJ04]) for static environment maps have been introduced. However, their high computational cost make them unsuitable for dynamic lighting case. One notable exception is the work from Havran et al. [HSK*05] where they generate samples on the CPU. However, contrary to their work, we do not precompute a set of visible light sources for each shaded point p. Our approach computes dynamically a set of light samples according to the light distribution of the environment map and selects some of them according to the pixel normal: it is therefore more suited for dynamic environment maps.

2. Fast and Continuous Sampling on the GPU

2.1. Monte-Carlo Estimator

When using cube maps, one needs to distribute the N_s samples between the six faces of the cube. Let μ_f be the proportion of samples used for the face f (i.e., $N_f = \mu_f N_s$), and $\text{pdf}_\text{L}(\boldsymbol{\omega}|f)$ the probability density function to select direction $\boldsymbol{\omega}$ on the face f. Then, Equation (1) can be rewritten in a MIS formalism [Vea98]:

$$L(p,\boldsymbol{o}) \approx \frac{1}{N_s}\sum_{f=1}^{6}\sum_{i=1}^{N_f}\frac{\text{s}(p,\boldsymbol{n},\boldsymbol{o},\boldsymbol{\omega}_i)}{\sum_{f=1}^{6}\mu_f\,\text{pdf}_\text{L}(\boldsymbol{\omega}_i|f)}. \tag{2}$$

In this paper, we target GPU importance sampling for dynamic environment maps. For this purpose, the cumulative distribution function of each cube face need to be computed at each frame. We thus use tabulated CDFs computed using optimized versions for GPU of the prefix sum [HSO07] on each face. The supplemental material provides a complete derivation on how to compute the associated pdf.

An obvious choice is to distribute uniformly the samples across the faces (i.e., $\mu_f = 1/6$). Unfortunately, depending on the orientation of $\boldsymbol{n}$, some samples may be behind the point and will be canceled by the cosine factor. A drastic example is when a very bright source (such as the sun) is behind the point p: many useless samples will still be generated on the face the sun belongs to. To prevent this behavior, one solution is to integrate the cosine factor $\langle\boldsymbol{n},\boldsymbol{\omega}\rangle$ into the PDF at the price of computing new CDFs that depends also on the normal $\boldsymbol{n}$. However, this becomes too costly in terms of memory and computation. Instead, we introduce a conservative, simple and dynamic strategy where μ_f depends on the normal $\boldsymbol{n}$ and is computed as follow:

$$\mu_f(\boldsymbol{n}) = \frac{F_f(\boldsymbol{n})I_f}{\sum_f F_f(\boldsymbol{n})I_f} \text{ with } F_f(\boldsymbol{n}) = \sum_{c_j\in f}\left\langle\boldsymbol{n},c_j\right\rangle \tag{3}$$

where c_j are normalized vertexes of the cube map face f. $F_f(\boldsymbol{n})$ can be seen as a pseudo form factor (point to face). Equation (3) balances dynamically for p the number of samples on each face according to its importance.

To prevent popping artefacts when the number of samples is low and changing between neighboring normals, we use a floating point N_f and introduce a new weight β_i for each sample. By combining Equation (2) and Equation (3) our estimator becomes:

$$L(p,\boldsymbol{o}) \approx \frac{1}{N_s}\sum_{f=1}^{6}\sum_{i=1}^{i<N_f(\boldsymbol{n})+1}\beta_i\frac{\text{s}(p,\boldsymbol{n},\boldsymbol{o},\boldsymbol{\omega}_i)}{\mu_f(\boldsymbol{n})\,\text{pdf}_\text{L}(\boldsymbol{\omega}_i|f)} \tag{4}$$

$$\text{with } \beta_i = \begin{cases}1 & \text{if } i \le N_f(\boldsymbol{n})\\ \text{frac}(N_f(\boldsymbol{n})) & \text{else}\end{cases} \tag{5}$$

where frac returns the fractional portion. Since $\text{pdf}_\text{L}(\boldsymbol{\omega}|f) = 0$ if $\boldsymbol{\omega} \notin f$, the sum in the denominator in Equation 2 disappears. Thanks to the use of the weight β_i, the last sample is introduced progressively.

As demonstrated in the results section, compared to the classical approach, our strategy improves the frame rate because it limits the generation of useless light samples. Based on MIS formalism, we can easily integrate other strategies. We illustrate this by using BRDF importance sampling pdf_B. For the balance heuristic, half samples are used for light importance sampling, half for BRDF. Therefore, we use $N_f(\boldsymbol{n}) = \mu_f(\boldsymbol{n})N_s/2$ for each face and $N_b = N_s/2$ for the BRDF. This leads to the following estimator:

$$L(p,\boldsymbol{o}) \approx \frac{1}{N_s}\left(\sum_{f=1}^{6}\sum_{i=1}^{i<N_f(\boldsymbol{n})+1}\beta_i\,g_{f,i} + \sum_{i=1}^{i<N_b+1}\beta_i\,g_{f,i}\right)$$
$$\text{with } g_{f,i} = \frac{\text{s}(p,\boldsymbol{n},\boldsymbol{o},\boldsymbol{\omega}_i)}{(1/2)\mu_f(\boldsymbol{n})\,\text{pdf}_\text{L}(\boldsymbol{\omega}_i|f) + (1/2)\,\text{pdf}_\text{B}(\boldsymbol{\omega}_i)} \tag{6}$$

2.2. GPU implementation

The rendering process for each frame is organized as presented in Algorithm 1. The whole process starts by an early GBuffer pass (line 4) and ends by the final tone mapping (lines 15 and 16). In-between, we compute the tabulated

Algorithm 1 Steps to render one frame. We use GC when using GPU computing and GS when using GPU shader. We indicate the for-loop that are parallelized using CUDA threads with the keyword **"do in parallel"**.

```
1:  procedure RenderFrame
2:      GEO_2D                          ▷ Buffer for vertex positions and normals
3:      LS_1D[f]                        ▷ Buffers of light samples for each face f
4:      GBuffer_Pass(GEO_2D)                                          ▷ GS
5:      for each face f of the cube environment map do
6:          compute_luminance_per_pixel                               ▷ GC
7:          compute_prefix_sum_by_inclusive_scan                      ▷ GC
8:          compute_CDF_1D_u                                          ▷ GC
9:          compute_CDF_2D_v_knowing_u                                ▷ GC
10:         LS_1D[f]=generate_light_samples(N_s/2)                    ▷ GC
11:     end for
12:     for each pixel (p,n) ∈ GEO_2D do in parallel
13:         shade(o,p,n,N_s)                                  ▷ GC - Algo. 2
14:     end for
15:     compute_average_intensity(frame)                              ▷ GC
16:     Tone_mapping(frame)                                           ▷ GS
17: end procedure
```

Algorithm 2 Shading procedure with our dynamic balancing technique. To take advantage of the CUDA architecture, we have integrated the BRDF sampling pass as a 7th pass and used a fixed maximal number of iterations (cf Line 6).

```
1:  procedure shade(o,p,n,N_s)                  ▷ in parallel for each pixel p
2:      N_f[1..6] = Samples_Per_Face(N_s/2)                  ▷ Equation 3
3:      N_f[7] = N_s/2                          ▷ number of BRDF samples
4:      for each step f=1..7 do
5:          for each i = [0,N_s/2] do
6:              break when i ≥ N_f[f]
7:              compute β_i                                  ▷ Equation 5
8:              if f < 7 then
9:                  sample = LS_1D[f][i]                     ▷ see Algo. 1
10:             else
11:                 sample = BRDF_sampling
12:             end if
13:             L(p,o)+= β_i g_{f,i}                         ▷ Equation 6
14:         end for
15:     end for
16:     L(p,o) = L(p,o)/N_s
17: end procedure
```

CDF (lines 5 to 11), then we generate the light samples (line 10) and compute the shading (line 13).

We compute the 2D CDF by using the inversion method [PH04]) and store it on GPU. More precisely, a 1D CDF (CDF(u)) and a 2D CDF (CDF($v|u$)) are computed using parallel prefix sum [HSO07] and stored as floating point buffer for each cube map face (u and v are the pixel coordinates on a face corresponding to a given light sample). The CDF computations are implemented using two GPU Computing kernels (lines 8 and 9) that are called successively because CDF($v|u$) depends on CDF(u). Based on these CDFs, we conservatively generate $N_s/2$ light samples per face before computing the shading. This ensures the generation of a sufficient amount of samples on each face for the dynamic balancing. For degenerated cases, all the $N_s/2$ samples dedicated to light sampling will be on a unique face. These samples are generated using a classical binary search.

High performance in CUDA requires thread coherency, memory coalescing and branch consistency. We have therefore designed the Algorithm 2 for the shading step (cf. line 13) to reach the best performance of our different implementations. After shading, the computation of the average luminance is done using a reduction operation [RAH07] for the tone mapping step.

Finally, since our per pixel Monte-Carlo estimator is unbiased, our approach does not introduce any bias for a given pixel. However, for efficiency reason, we use the same precomputed random sequence for all the pixels. This introduces a spatial bias between neighbor pixels but it has the advantage to limit disturbing noise in the final image. A possible extension to reduce the spatial bias would be to use interleaved sampling.

3. Results

The companion video and all results presented in this paper were computed on a 2.67 GHz PC with 6 GB of RAM and a NVIDIA GTX 680 graphics card. We implemented our system using DirectX, CUDA and the Thrust Library. The static environment map used in Figure 2 has a resolution of $256\times256\times6$ pixels. For the dynamic environment map used in Figure 1, we use the available 67 frames from the capture made during a full day by Stumpfel et al. [STJ*04]. Before using it, the only preprocess we apply on the captured images is a re-parametrization (from hemisphere to cube) to obtain $256\times256\times6$ pixels images. We also fill with an average color the pixels belonging to the roof where the acquisition device was placed. For all results, we achieve real-time framerates with dragon models, 369K (resp. 100K) polygons in Figure 1 (resp. Figure 3) as well as the model (169K polygons) used in Figure 2.

Figure 1 shows a set of images with a time-varying environment map captured from daylight to night. The central images show where the light samples are located for a given pixel. Notice how our technique avoids generating useless samples on the sun for the highlighted pixel. Moreover, the companion video demonstrates our real-time performance as well as the temporal coherence of our Monte-Carlo estimator when changing the view direction or the lighting or even the BRDF parameters.

Figure 2 presents a comparison between a classical light-based importance sampling and our approach. As shown in the picture, both images have almost the same quality when compared to a reference solution, but our technique uses three times less samples and is consequently faster. Finally, Figure 3 demonstrates the convergence speed and the quality obtained with our MIS technique. As shown by the Lab difference between our results and a reference solution, even

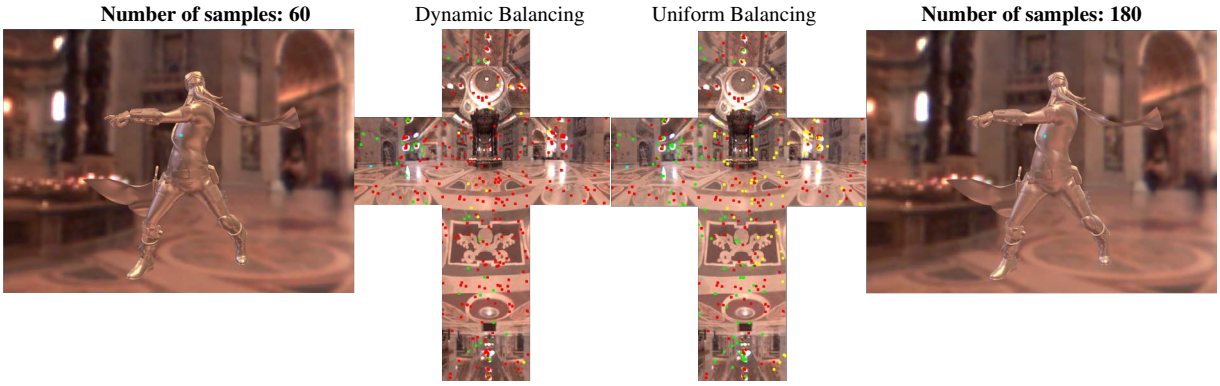

Figure 2: *Comparison of (**Left**) our dynamic sampling technique for the highlighted pixel (with a cyan dot) with (**Right**) uniform balancing of the samples per face. Among the pre-generated samples (red+yellow+green), our technique selects* 60 *of them for the current pixel (yellow and green dots) from which 47 are effective samples (green dots). However, to achieve the same quality with uniform balancing, three times more samples are required (180 vs 60) resulting in 116 effective samples. The Lab errors are compared with a reference solution computed with* $256 \times 256 \times 6$ *samples generated uniformly on the environment map.*

Figure 3: *Convergence speed of our technique when increasing the number of samples per pixel (spp). We compute the mean Lab error between our results and a reference solution (right image) computed using* 655 000 *spp generated from a cosine-based hemisphere sampling scheme. As shown by the decreasing Lab errors our MIS technique converges toward the correct solution when increasing the number of samples. Even with 240 spp, we achieve real-time frame rate (81 fps) with a very low Lab error (3.56).*

with 60 samples per pixel our method achieves visual plausible results in real-time (198 fps).

4. Conclusion and Future Work

In this paper we have introduced an improved Monte-Carlo estimator for light-based importance sampling of dynamic environment maps. Our pixel-based technique increases the number of effective samples and is faster for the same quality compared to a uniform distribution of samples on each face. Furthermore, our technique handles efficiently dynamic and time-varying environment maps. Based on Multiple Importance Sampling formalism, it can be easily combined with other sampling strategies. For future work, we would like to incorporate a more robust balancing scheme to distribute light samples and also to introduce visibility and indirect lighting effects.

Acknowledgments

Heqi Lu's PhD scholarship is funded by the Région Aquitaine. This research has been supported by the ALTA project (ANR-11-BS02-006).

References

[HSK*05] Havran V., Smyk M., Krawczyk G., Myszkowski K., Seidel H.-P.: Interactive System for Dynamic Scene Lighting using Captured Video Environment Maps. In *Eurographics Symposium on Rendering* (2005), pp. 31–42. 1, 2

[HSO07] Harris M., Sengupta S., Owens J.: Parallel prefix sum (scan) with CUDA. *GPU Gems 3*, 39 (2007), 851–876. 2, 3

[ODJ04] Ostromoukhov V., Donohue C., Jodoin P.-M.: Fast hierarchical importance sampling with blue noise properties. In *Proc. SIGGRAPH '04* (2004), ACM, pp. 488–495. 2

[PH04] Pharr M., Humphreys G.: *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann Publishers, 2004. 3

[RAH07] Roger D., Assarsson U., Holzschuch N.: Efficient Stream Reduction on the GPU. In *Workshop on General Purpose Processing on Graphics Processing Units* (Oct. 2007). 3

[STJ*04] Stumpfel J., Tchou C., Jones A., Hawkins T., Wenger A., Debevec P.: Direct HDR capture of the sun and sky. In *Proc. AFRIGRAPH '04* (2004), ACM, pp. 145–149. 1, 3

[Vea98] Veach E.: *Robust monte carlo methods for light transport simulation*. PhD thesis, 1998. 2

[WTL06] Wang R., Tran J., Luebke D.: All-frequency relighting of glossy objects. *ACM Trans. Graph. 25*, 2 (2006), 293–318. 1