GENERATIONS / VANCOUVER 12-16 AUGUST
SIGGRAPH2018

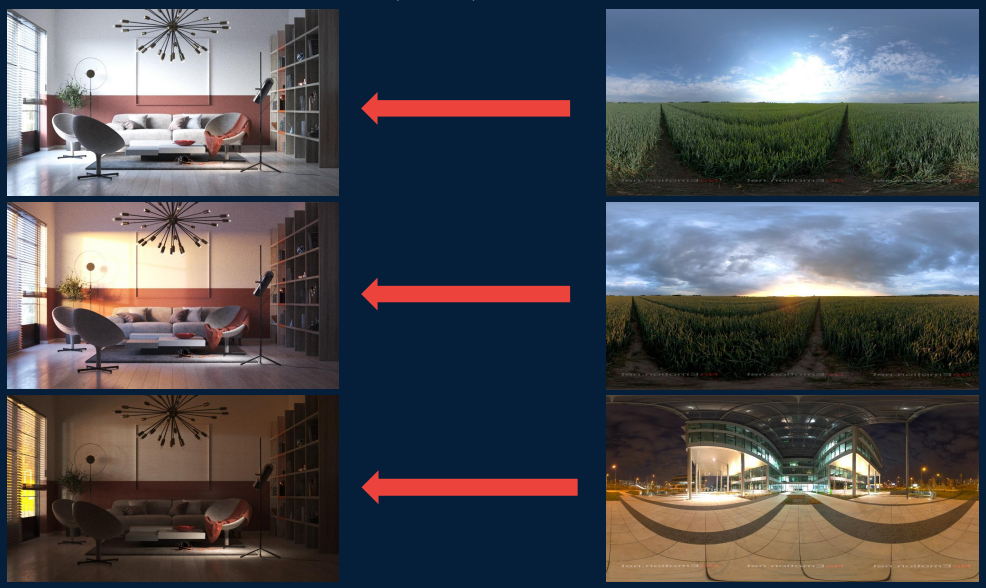
Adaptive Environment Sampling on CPU and GPU

Asen Atanasov
Alexander Soklev
Vladimir Koylazov
Vassillen Chizhov
Blagovest Taskov
Jaroslav Křivánek

Hello everybody!

Welcome to our talk “Adaptive Environment Sampling on CPU and GPU”. Today we will share about this project which was jointly developed by V-Ray R&D and V-Ray GPU teams.

Image-based lighting (IBL)



HDR images courtesy of NoEmotion

Image-based lighting is an important tool in rendering. In production, large HDR images are used to provide detailed illumination for complex scenes. Here's one of our test scenes lit by three HDRs.

IBL noise

Unfortunately, in heavily occluded scenes, like interiors, IBL is a major source of noise.

Portals

Traditionally, this problem has been solved by the introduction of portals. Portals are rectangular regions which artists are expected to manually place in their scenes around the windows to guide the shadow rays during rendering.

We used a portal based solution for years and recently efficient portals-based solutions have been developed.

This approach brings undesirable disadvantages. Artists need to perform some tedious additional work, which can be slow and tricky depending on the scene. They must learn how to use portals in order to obtain optimal results. Manual work is always prone to mistakes and badly placed portals can even slow the rendering down. For instance, artists may have different ideas how to cover circular opening with rectangular regions. For the reasons mentioned above we seek a solution that doesn’t rely on portals.

Existing solutions

- Rely on portals
- High memory consumption
- Expensive computation
- Complex data structures

Generally, existing solutions have some of the following disadvantages:

- they either rely on portals
- could require high memory consumption
- or need expensive computations
- or use complex data structures

Thus their practicality is limited, especially for GPU rendering of complex production scenes.

For instance, we prefer grid-based data structures instead of tree-based. Tree traversal results in higher code divergence which decreases the utilization of the GPU SIMD architecture.

Guidelines

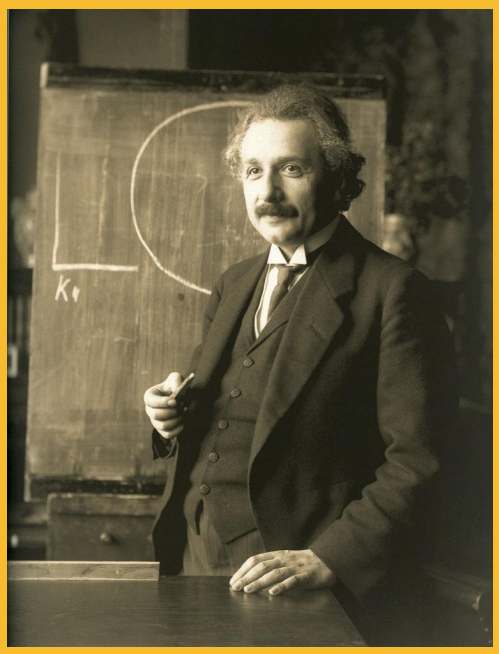
- Complex production occluded scenes
- CPU and GPU
- Account for visibility
- Lightweight sampling procedure
- No user manual work
- Low memory usage
- Simple to implement

During the development of our algorithm we set up guidelines that motivated our engineering decisions.

- First we need a solution that can handle arbitrarily complex production scenes
- We want a unified approach which will improve both our CPU and our GPU render engines
- Accounts for visibility to improve partially or heavily occluded scenes, but also benefits or at least does not slow down unoccluded scenes.
- We achieve this goal using a lightweight sampling procedure, which requires no tree traversals. Note that an approach that improves occluded scenes, but slows down other scenes imposes the requirement that the users must know when to switch on/off this solution.
- Requires no portals or any other manual scene-specific adjustments. Have few or none control parameters.
- Has modest memory requirements. This is especially important for very heavy scenes on the GPU.
- Lastly, algorithms that are simple to implement, integrate, debug and modify are always preferable, especially in our production context.

Everything should be made as **simple** as possible, but not simpler.

~Albert Einstein

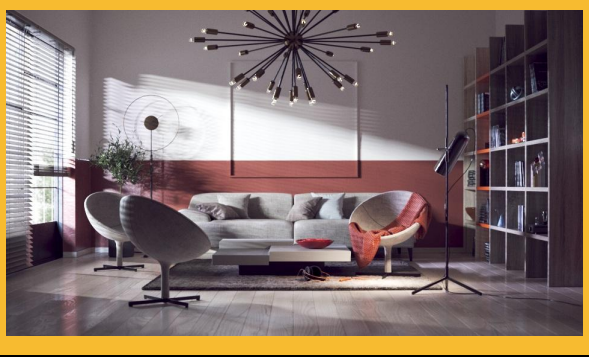


During the development process we discussed and experimented with different ideas and reached a RELATIVELY SIMPLE solution which follows the desired guidelines and works well for us.

Our Adaptive Sampling

- Partition the environment map
- The Light Grid
  - Visibility cache
  - In the camera space
- Two-phase approach
  - Learning
  - Rendering

Office scene courtesy of Evermotion



To the right are two of our test scenes. They are chosen for their heavy occlusion.

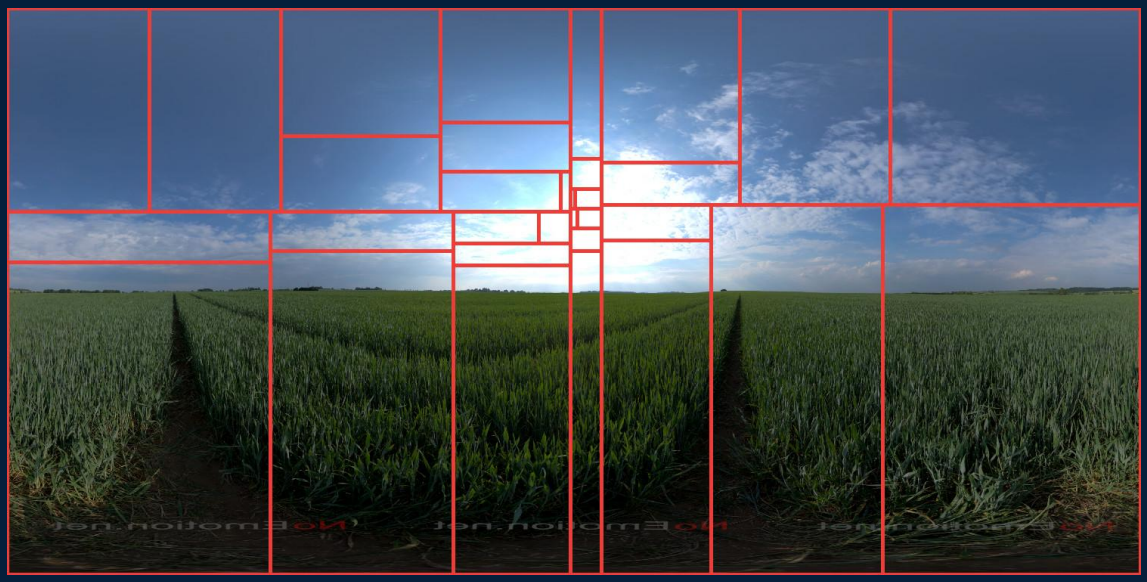
The basic idea of our adaptive sampling algorithm is to figure out which parts of the environment are important to which parts of the scene.

Thus we first partition the environment map into tiles.

We use a data structure that we call “The Light Grid” to cache visibility information in the camera space.
We take two-phase approach: during a brief learning phase we accumulate the importance of environment map tiles to different parts of the scene in the Light Grid. Then we use this cache during the rendering phase to guide the shadow rays.

Now let’s get into the details:

32 equal-energy tiles



HDR image courtesy of NoEmotion

First we talk about partitioning. We experimented with two natural partitioning strategies: equal-energy and equal-sized tiles.

Equal-energy tiles seemed a promising choice, because more small tiles are placed in the important high energy regions. Here's an example.

32 equal-energy tiles

HDR image courtesy of NoEmotion

However, more experiments reveal critical drawbacks. In the very common case of a very bright spot in the environment map the tiles become very thin and long.

32 equal-energy tiles - very **thin** tiles

HDR image courtesy of NoEmotion

They can easily degenerate for environment maps with bright day sun for example.
We seek robust and well-localized partitioning in the direction space.

4 x 8 equal-sized tiles

HDR image courtesy of NoEmotion

Here’s an example of 32 equal tiles.

Equal-energy tiles

- Thin and long tiles
- Degenerate tiles around bright spots
- Traversal or more memory for point-in-tile test

HDR images courtesy of NoEmotion

Equal-sized tiles

- Equal square tiles
- Robust and simple partitioning
- Faster point-in-tile test

In summary, here are the important features of the two strategies, relevant to our approach.

While equal-energy partitioning is prone to thin or degenerate tiles over bright environment map regions, the equal tiles are robust and trivial to compute.

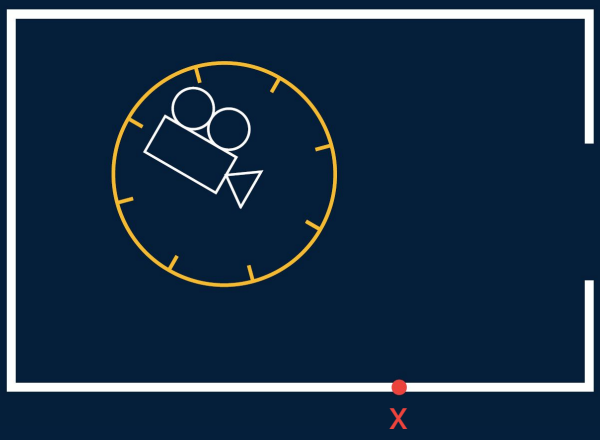
Important advantage of the equal-tile partitioning is that for any point on the map we can find the corresponding tile in constant time. This is a key to our lightweight sampling.
On the other hand, equal-energy tiles would require either a tree traversal or additional memory.
Thus, we stick to the the equal-tiles partitioning.

The Light Grid

- G_x x G_y spherical grid - $G_x = 2G_y$
- In the camera space

The Light Grid is a uniform spherical grid centered at the camera. In this diagram the camera is inside a room and there is a window in the right ride.

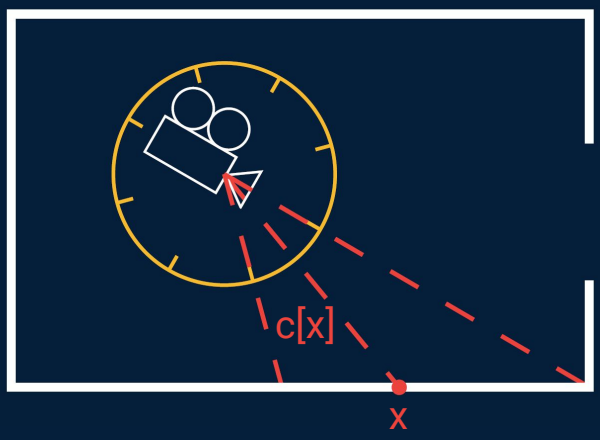
The Light Grid



- G_x x G_y spherical grid - $G_x = 2G_y$
- In the camera space
- Each scene point belongs to a Light Grid cell

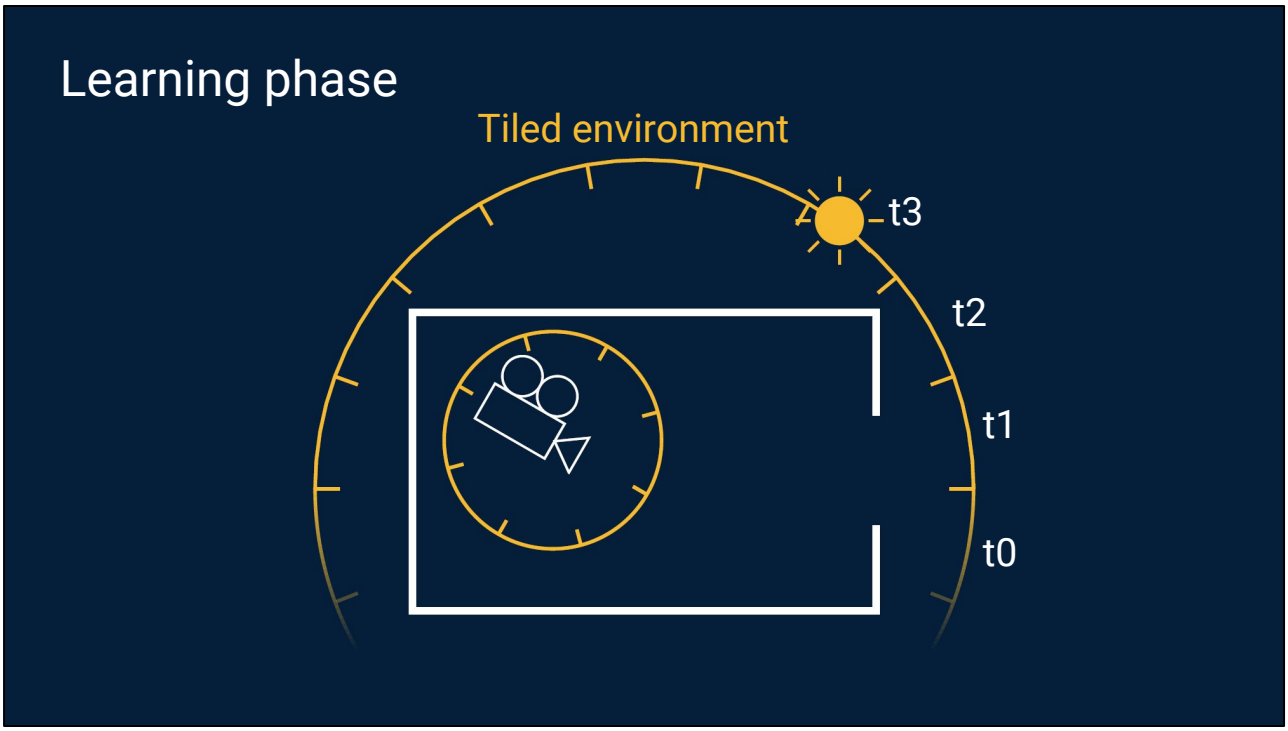
Note that each scene point belongs to exactly one Light Grid cell.
For instance, the point x can be projected to the camera center to figure out its Light Grid cell c[x].

The Light Grid



- G_x x G_y spherical grid - $G_x = 2G_y$
- In the camera space
- Each scene point belongs to a Light Grid cell

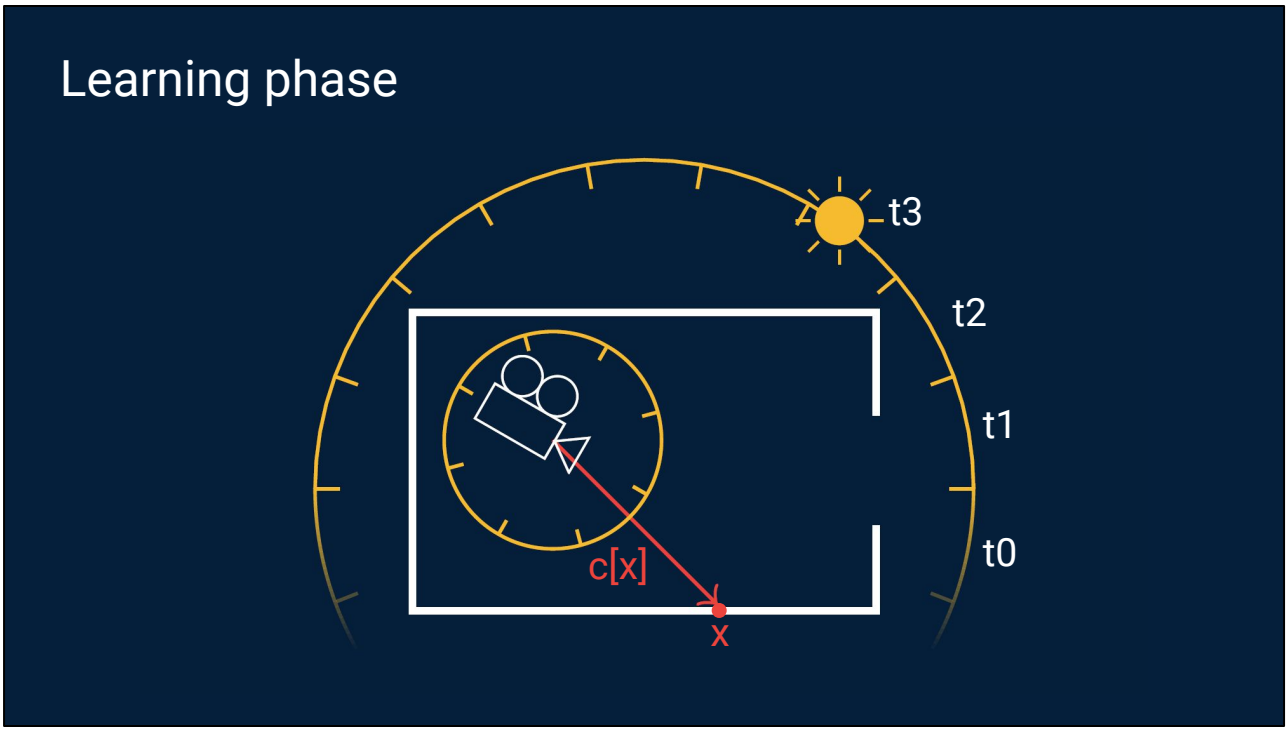
Each Light Grid cell holds one floating point weight for each environment map tile which represents the importance of the tile for this cell.



Now I will explain the learning phase:

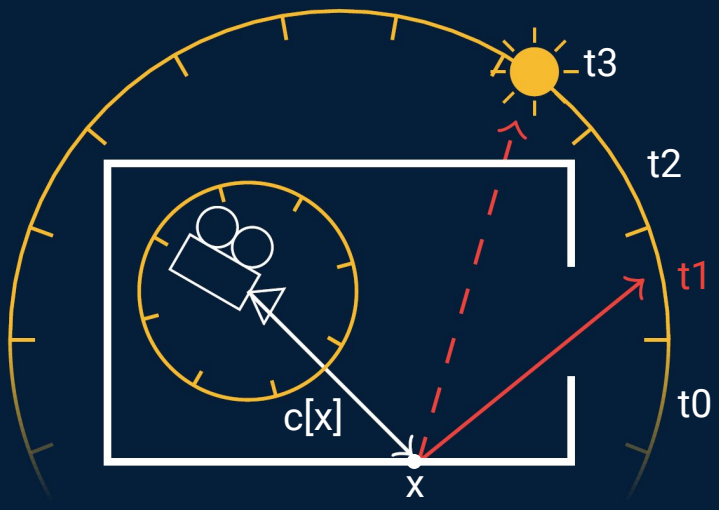
In this diagram the room is surrounded by the environment map which is partitioned into equal tiles. There is a bright sun in the map, so when we importance sample the map many samples fall there, in tile t3. The problem is that this tile is entirely occluded, so these samples are wasted.

In the beginning, all Light Grid cells are initialized with zero.

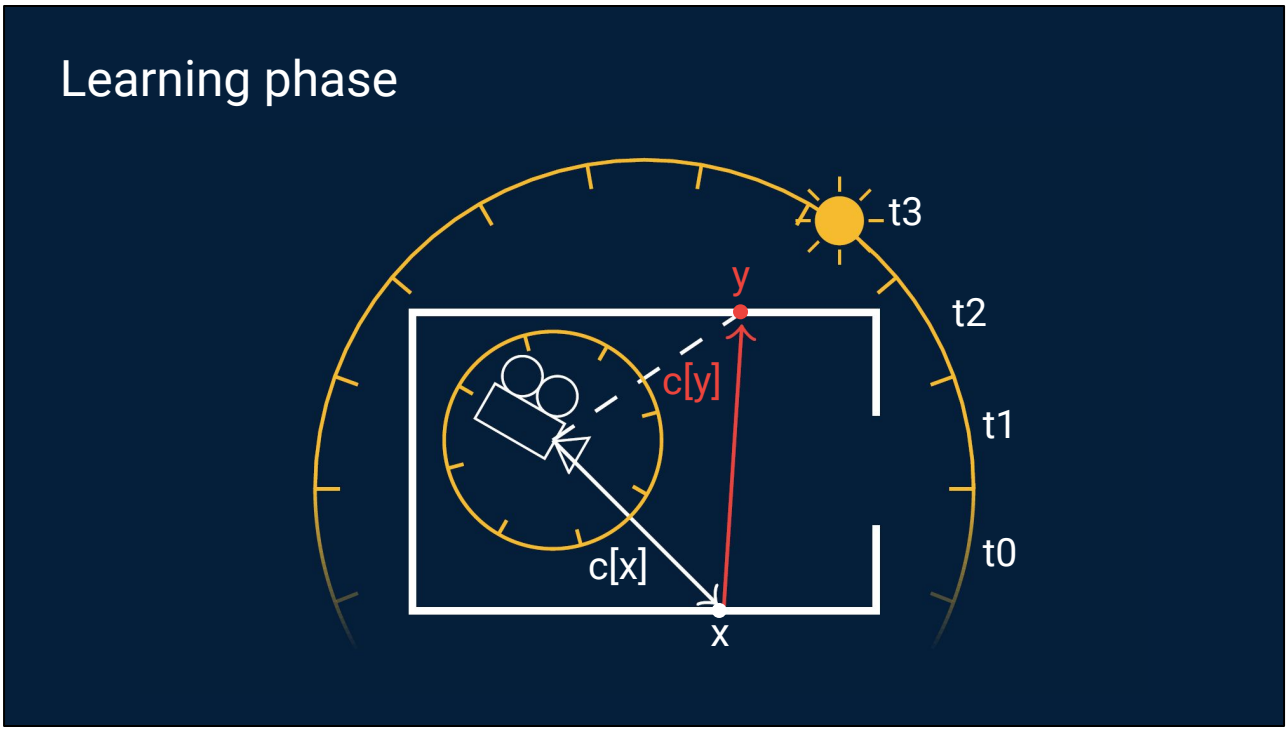


Then we start to trace camera paths into the scene, and for each path vertex point we compute the direct illumination.

Learning phase

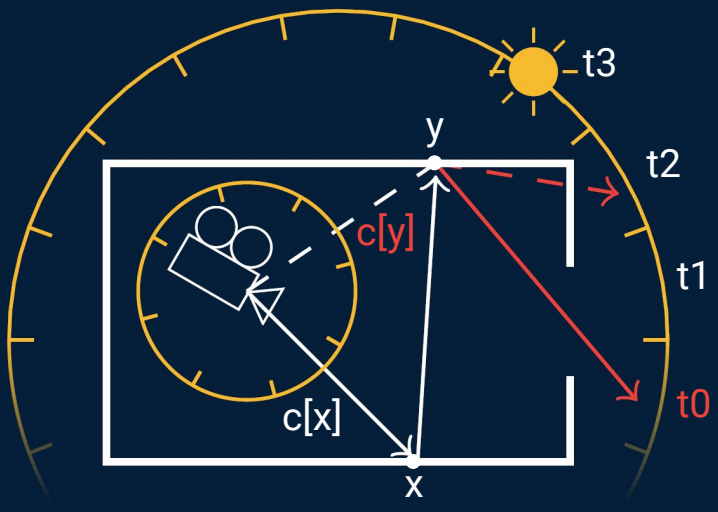


When we importance sample the environment map, we accumulate information for the visible tiles and we ignore the occluded.
In this example, the weight of the tile t1 for the Light Grid cell c[x] is updated.



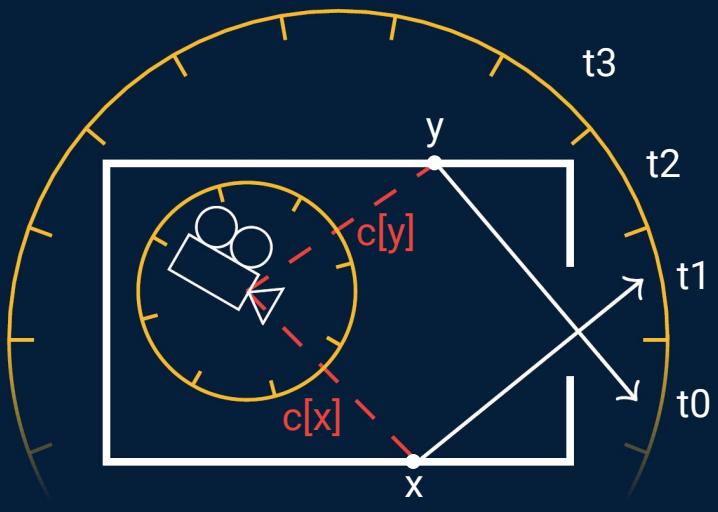
We continue the path and apply the same strategy for each path vertex. For this GI ray we find the intersection y and its corresponding cell c[y].

Learning phase



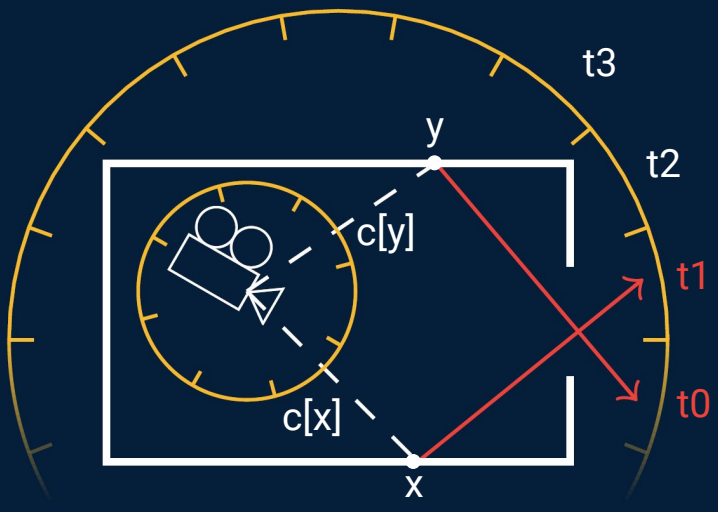
Then again we importance sample the environment map and update the visible tile weights.
In the end of the learning phase we construct a CDF in each Light Grid cell, based on the tile weights.

Rendering phase

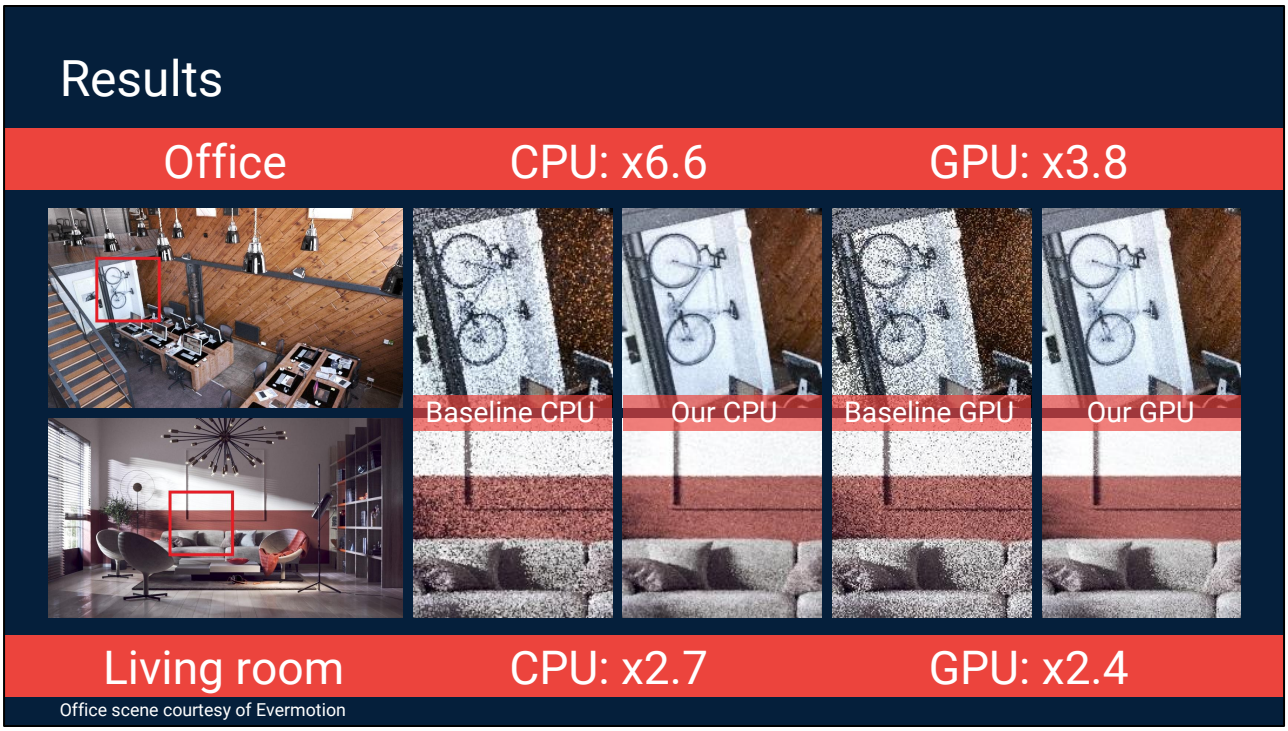


Then during the rendering phase, for each shading point in the scene, we find the corresponding Light Grid cell by back projecting it to the camera center. Then we use the precomputed probability distributions in the cells to sample tiles according to their visible intensity.

Rendering phase



When a tile is chosen, then we pick a sample inside it according to the environment map intensity.



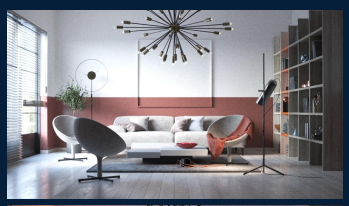
These are results for two of our test scenes: the “Office scene” in the first row and the “Living room scene” in the second.

In the insets we show both CPU and GPU results - the first two columns and the second two, respectively.

First you can observe how our adaptive sampling produces much cleaner images for the same time.
Here the baseline algorithm is BRDF sampling and environment map sampling combined with MIS.
And our approach is BRDF sampling and our adaptive sampling procedure, also combined with MIS.

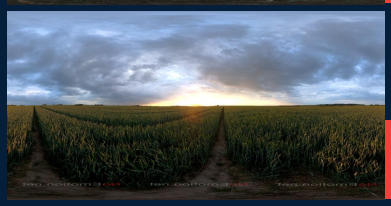
In the red band you can see the effective speedup of our algorithm for the same noise levels for both CPU and GPU.

Results




HDR “Day”

CPU: x2.2 GPU: x1.6

HDR “Sunset”

CPU: x1.9 GPU: x1.6


HDR “Night”

CPU: x3.8 GPU: x3.0

HDR images courtesy of NoEmotion

More results: the “Living room scene” lit by three different HDRs. The effective speedups are shown on the right side.

Exterior and participating medium

CPU: x2.3 GPU: x1.8 CPU: x3.4 GPU: x2.6

Two more important experiments. Partially occluded exterior with and without participating medium. You can see that our algorithm improves the rendering substantially.
The right image with the fog showcases the scenario when many distant points map to the same Light Grid cell.

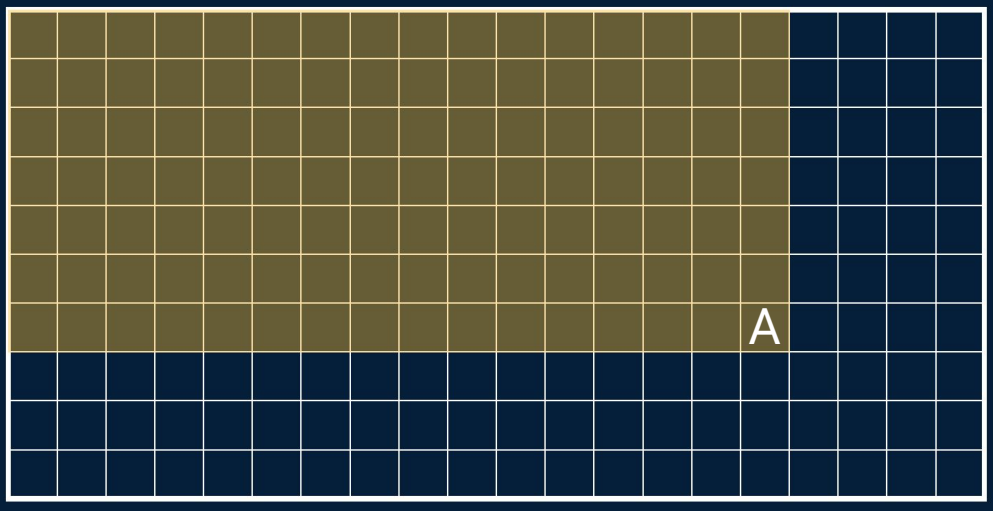
Implementation details

- CPU and GPU
- 10% - 700% speedup
- 10MB memory
- Learning:
  - 10^6 camera paths
  - ~ 1% of the render time
  - accumulation with fetch-and-add instructions
- Summed Area Table for sampling

Now I will share some implementation details:

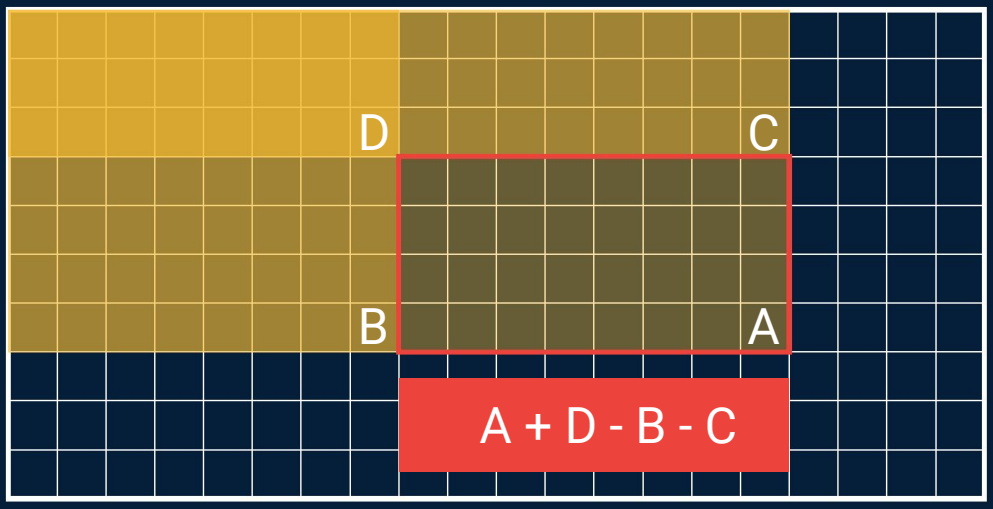
- We integrated our algorithm in two production renderers - one CPU and one GPU.
- We measured between 10 and 700 % speedup on user scenes.
- Our algorithm consumes 10MB of memory, regardless of scene complexity
    - Our default rendering pipeline starts with an irradiance caching solution which traces 1 million camera paths by default. We reuse this computation to accumulate the visibility information in the Light Grid.
    - The learning phase usually takes about 1% of the total rendering time.
    - Learning is computed efficiently in parallel using fetch-and-add instructions
- Our algorithm requires a procedure to importance sample the whole environment map during learning and the individual tiles during rendering. For this purpose we use SAT.

Summed-area table (SAT)



The Summed-area Table of the environment map is a 2D cumulative distribution function.
Each element is the sum of all previous elements in both directions.

SAT for sampling



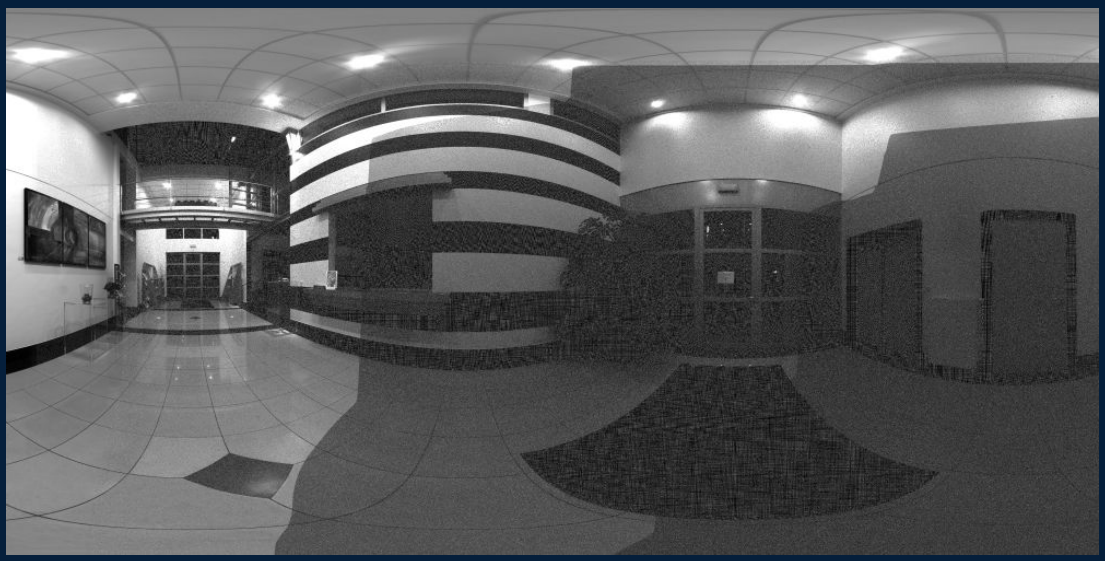
Every subregion of the SAT is monotonic, so bounds for binary search can be computed. In this way SAT can be used to importance sample arbitrary subregion of the map.

Hallway HDR image (10000x5000)

HDR image courtesy of Wouter Wynen (Aversis 3D)

Here’s one example of a very large HDR image.

Sampling reconstruction - 32-bit Float SAT



HDR image courtesy of Wouter Wynen (Aversis 3D)

Large SAT are notorious for being prone to numerical error.
In this example, we see sampling reconstruction using 32-bit float-valued SAT.
Moving to the bottom right end, the error accumulates and the quality of the sampling deteriorates, because of the increasing SAT values. This is best seen at the light sources.

One way to solve this problem is to use double precision, but this means twice more memory and twice slower queries.

Sampling reconstruction - 32-bit Integer SAT

HDR image courtesy of Wouter Wynen (Aversis 3D)

We take different approach. We build the table in double precision and then remap it to 32-bit unsigned integer. In this way the error is uniformly distributed and the sampling looks visually much better.

Integer-valued SAT vs. float-valued SAT

HDR image	Resolution	Int MSE	Float MSE
Hallway	10000x5000	1.0×10^{-5}	3.8×10^{-1}
Day	15000x7500	4.9×10^{-7}	8.6×10^{-3}
Night	3000x1500	1.4×10^{-8}	4.1×10^{-4}
Sunset	3000x1500	1.1×10^{-8}	3.6×10^{-4}

We measured the MSE for sampling reconstruction of four large HDRs.

Integer-valued SAT vs. float-valued SAT

HDR image	Resolution	Int MSE	Float MSE
Hallway	10000x5000	1.0×10^{-5}	3.8×10^{-1}
Day	15000x7500	4.9×10^{-7}	8.6×10^{-3}
Night	3000x1500	1.4×10^{-8}	4.1×10^{-4}
Sunset	3000x1500	1.1×10^{-8}	3.6×10^{-4}

You can see that our Integer-valued SAT improves the error with 3 to 4 orders of magnitude, compared to the float-valued SAT!

Q & A

Thank you for your attention and I’m ready to take questions!