Wavelet Representation and Sampling of Complex Luminaires

Asen Atanasov, Vladimir Koylazov
Chaos Software, Bulgaria

Hello,

I will present our work on efficient representation and rendering of complex luminaires.

Our method comes in a line of work for rendering complex luminaires in the context of unidirectional path tracing.

Why unidirectional path tracing?

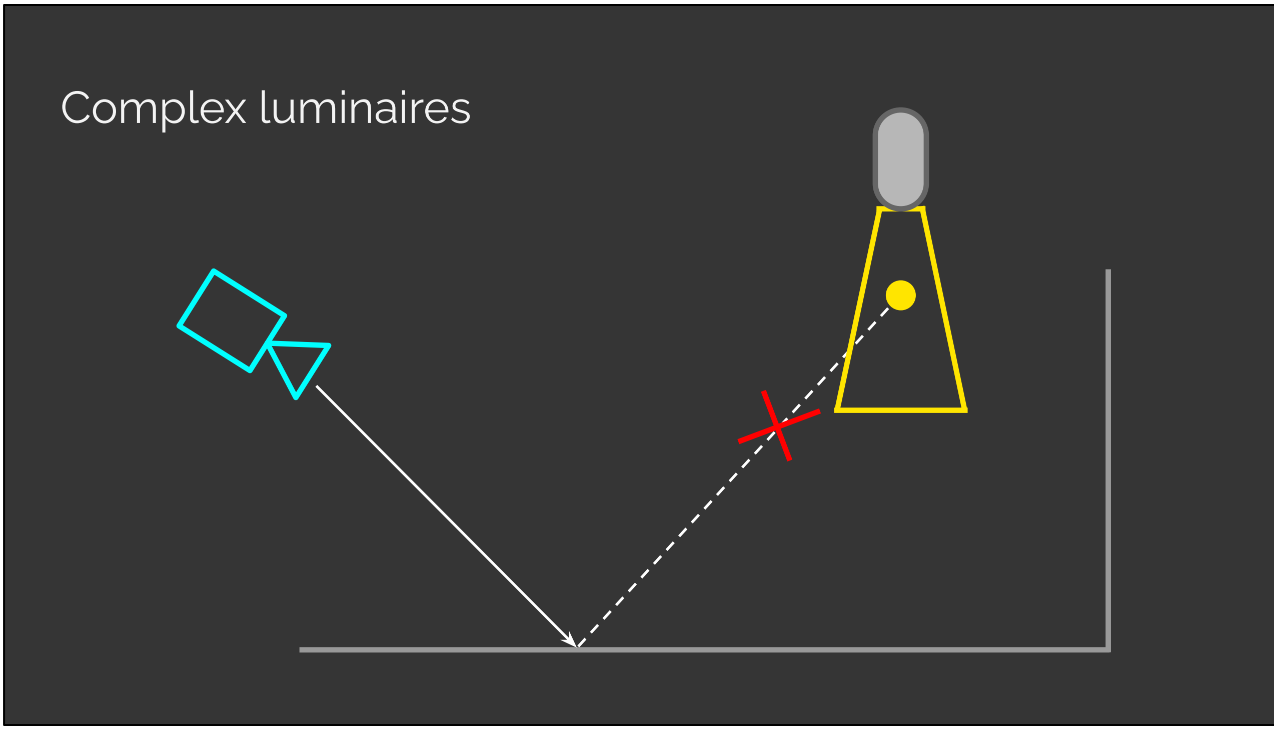
- Production rendering
- Real-time rendering
- Differentiable rendering

Why unidirectional path tracing? Because among ray tracers it’s almost exclusively used for production, real-time and differentiable rendering.

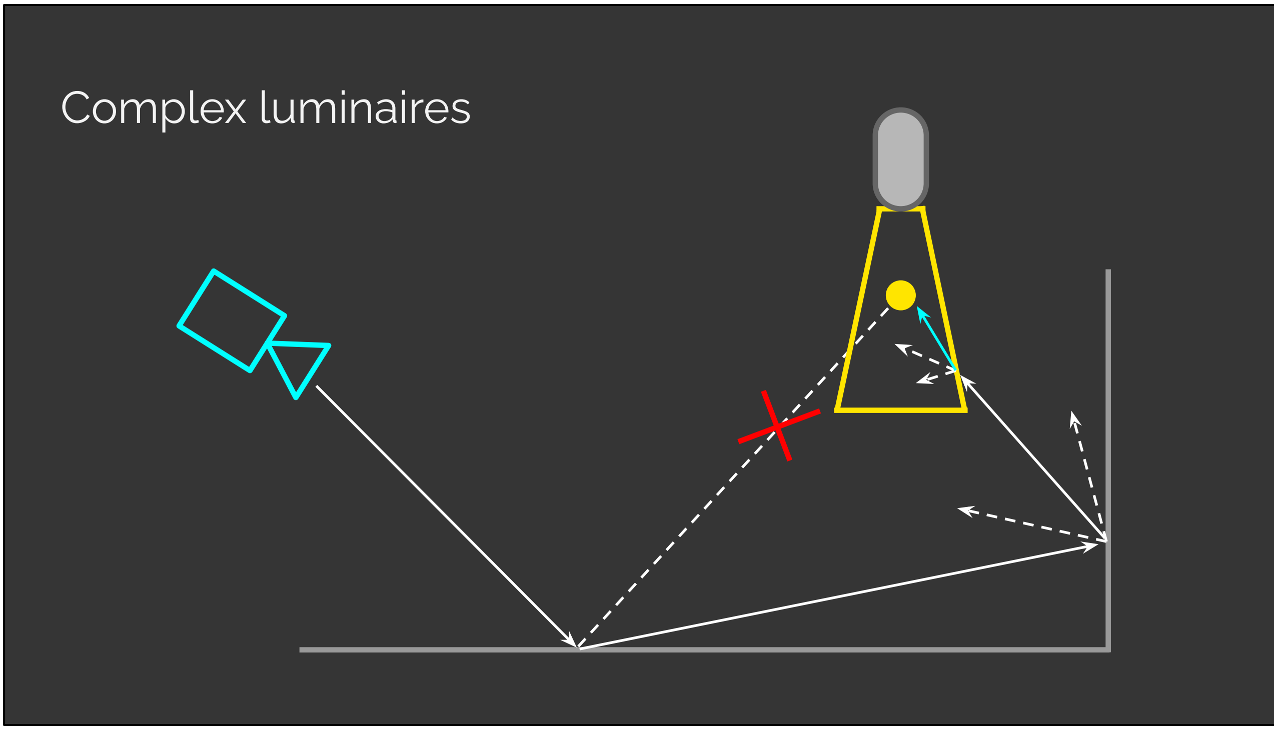
Complex luminaires

- Diffusers
- Reflectors
- Glass
- Lenses
- Occluders

Comlex luminaires are realistic light fixtures with emitters that are hidden behind various interfaces, e.g., diffusers (see the table lamp on the slide), or glass, reflectors, lenses, etc.

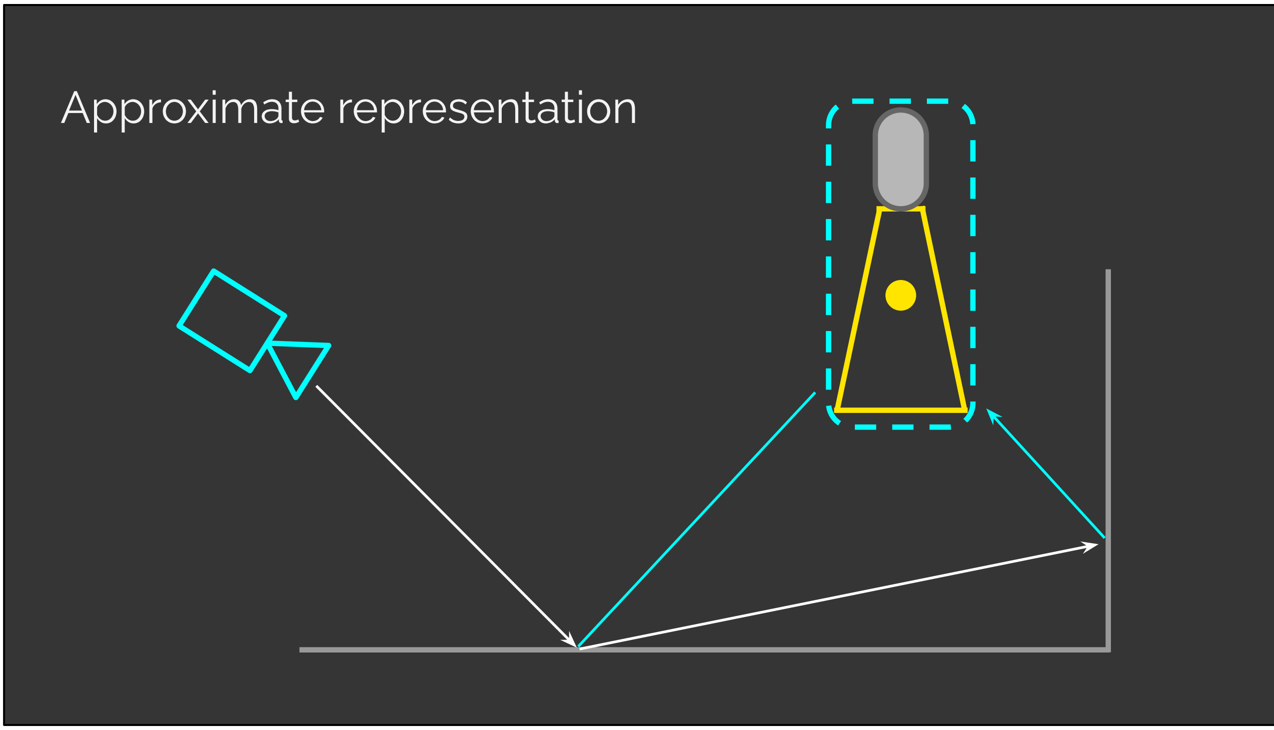


Thus, during path tracing next event estimation is not possible most of the time…

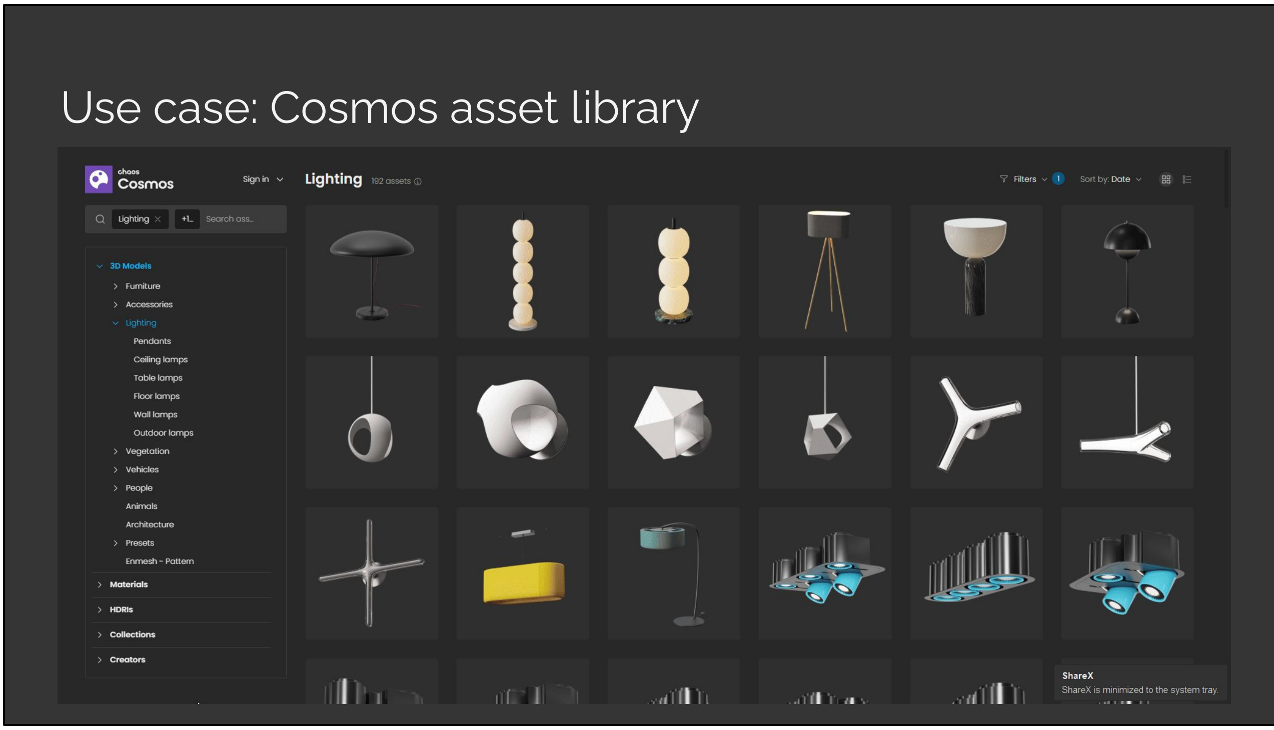


…and we need to rely on recursive path tracing to find contributing light paths.

Depending on the complexity of these paths, rendering such fixtures can range from challenging to totally unmanageable.



So the idea behind these approaches is to pre-compute an approximation and store it in a format that is suitable for efficient rendering.



One of our use cases is improving the rendering of hundreds of luminaires that are part of our growing online asset library.

Currently, the proposed method is provided for nearly 200 of them.

The method must offer fast pre-computation and low memory footprint to process and store large number of assets.

Common previous limitations

- Heavy pre-computation (Zhu et al. 2021, Condor and Jarabo 2022)
- High memory requirements (Condor and Jarabo 2022, Velázquez-Armendáriz et al. 2015)
- Very expensive field evaluation (Zhu et al. 2021)
- No importance sampling (Condor and Jarabo 2022, Velázquez-Armendáriz et al. 2015)
- No MIS support (Velázquez-Armendáriz et al. 2015)
- Aliasing artifacts (Velázquez-Armendáriz et al. 2015)

We developed our technique to address common practical limitations in previous work that prevent us from using previous methods for our use cases.

Note that previous methods have at least two of these limitations.

These include:

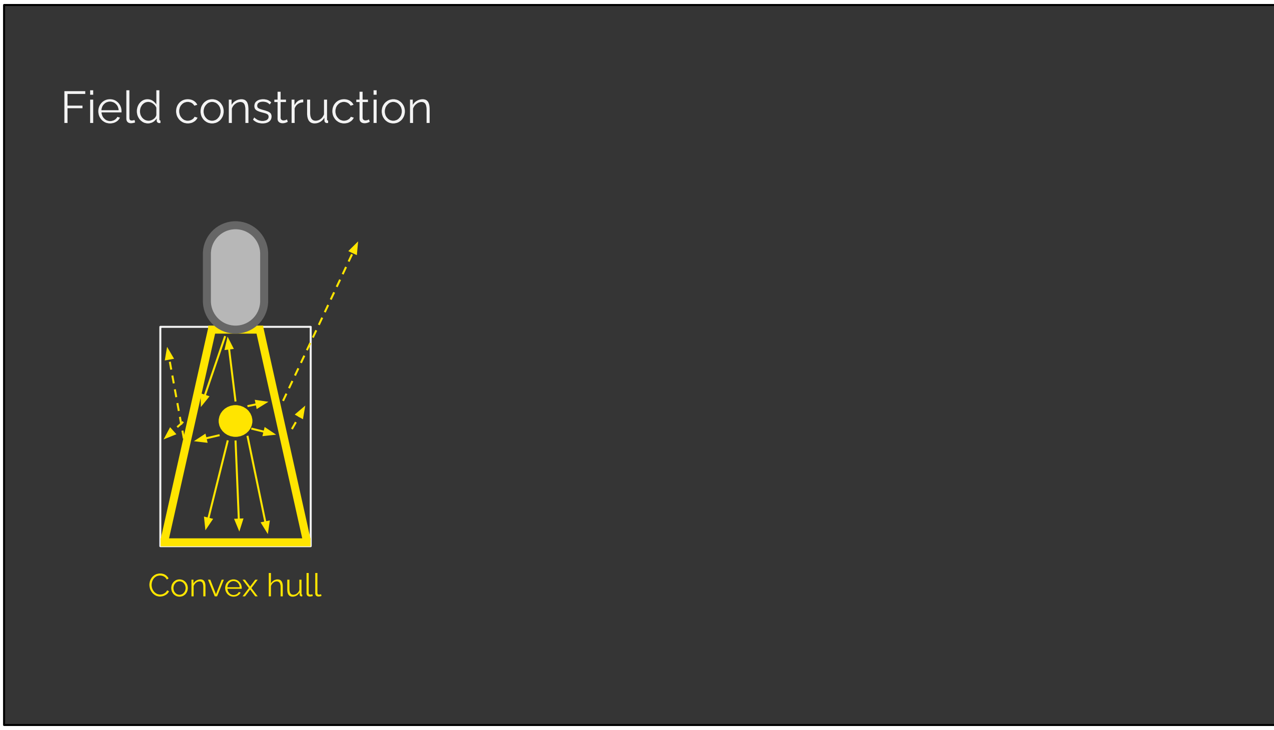
- very heavy pre-computation - some require multiple days, other multiple hours to pre-compute a single luminaire while our method usually runs in seconds
- high memory usage
- or expensive field evaluations that requires dedicated GPU for the task
- or lack suitable importance sampling strategy and MIS support
- some methods show aliasing artifacts

Our technique

Next I will describe our solution.

We pre-compute the luminaire's 4-dimensional exitant radiance over a convex hull encompassing the main emissive parts of the luminaire.

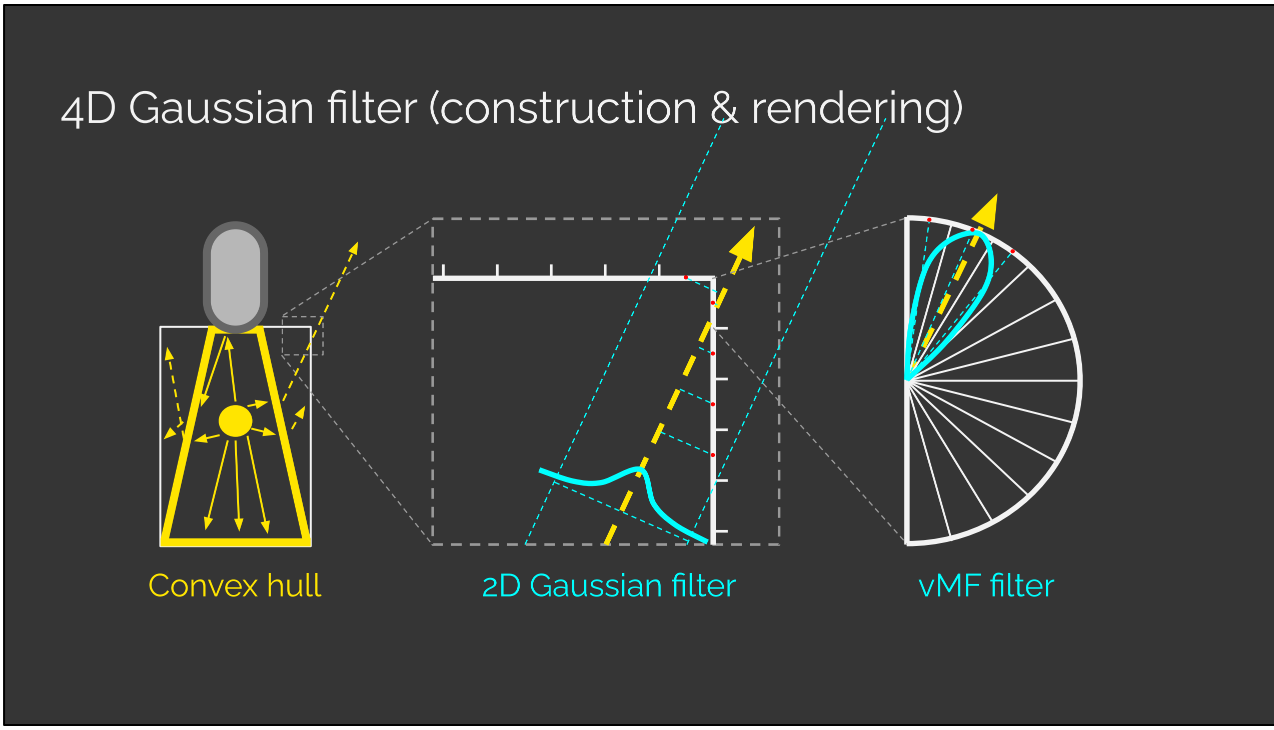
You see two examples on the slide with the emitting convex hulls shown in yellow.



We construct the field during a photon tracing phase.

We trace photons from emitters and when they exit the convex hull we record them to a spatio-directional grid on the convex hull bounding box.

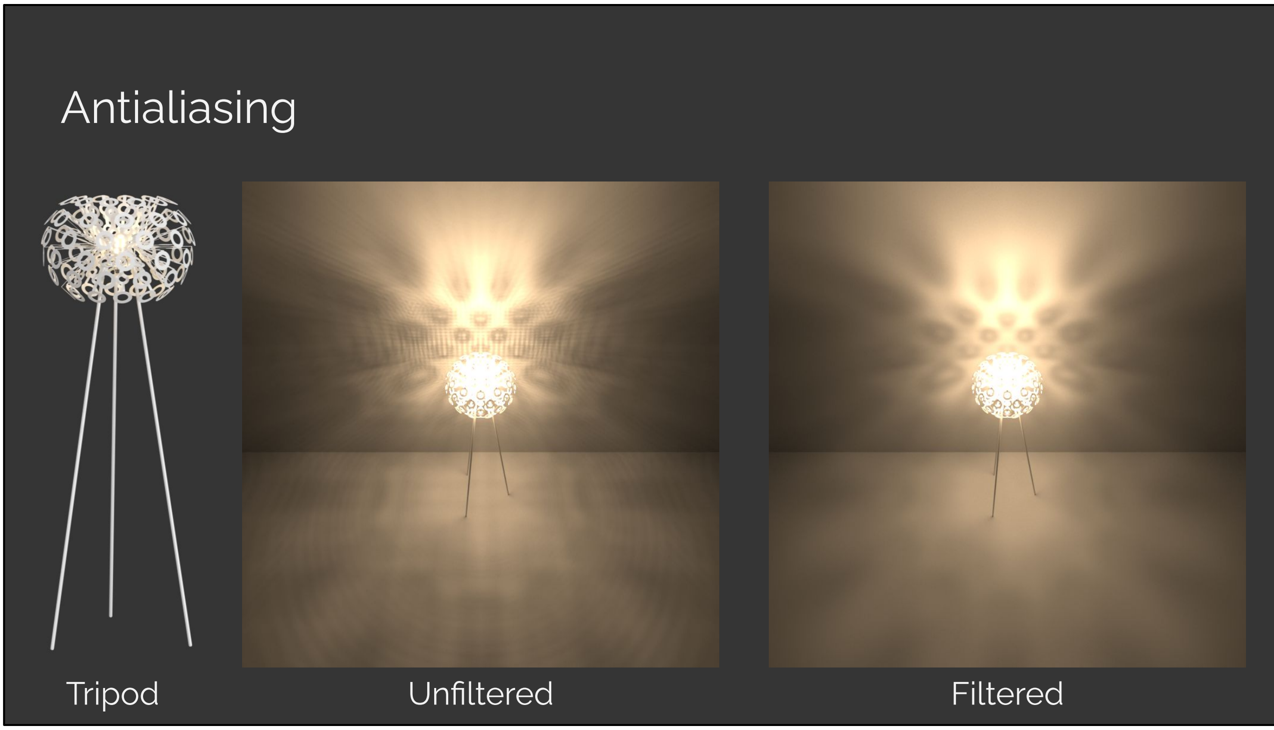
Note that the field is represented on the convex hull and we only use the bounding box to parametrize the hull.



This representation is prone to aliasing, thus we employ a combination of two 4D Gaussian filters: one during construction and one during rendering.

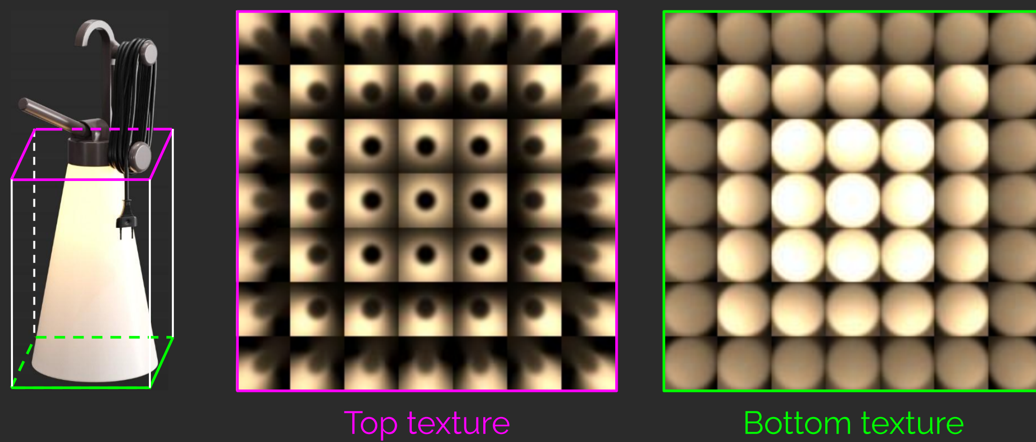
During construction, we splat the photon onto the 4D spatio-directional grid: we use a 2D Gaussian filter for the positional component and a vMF filter for the directional component.

During rendering, we use an analogical 4D Gaussian filter but we apply it stochastically and we fine-tune the filter parameters to reduce the variance.

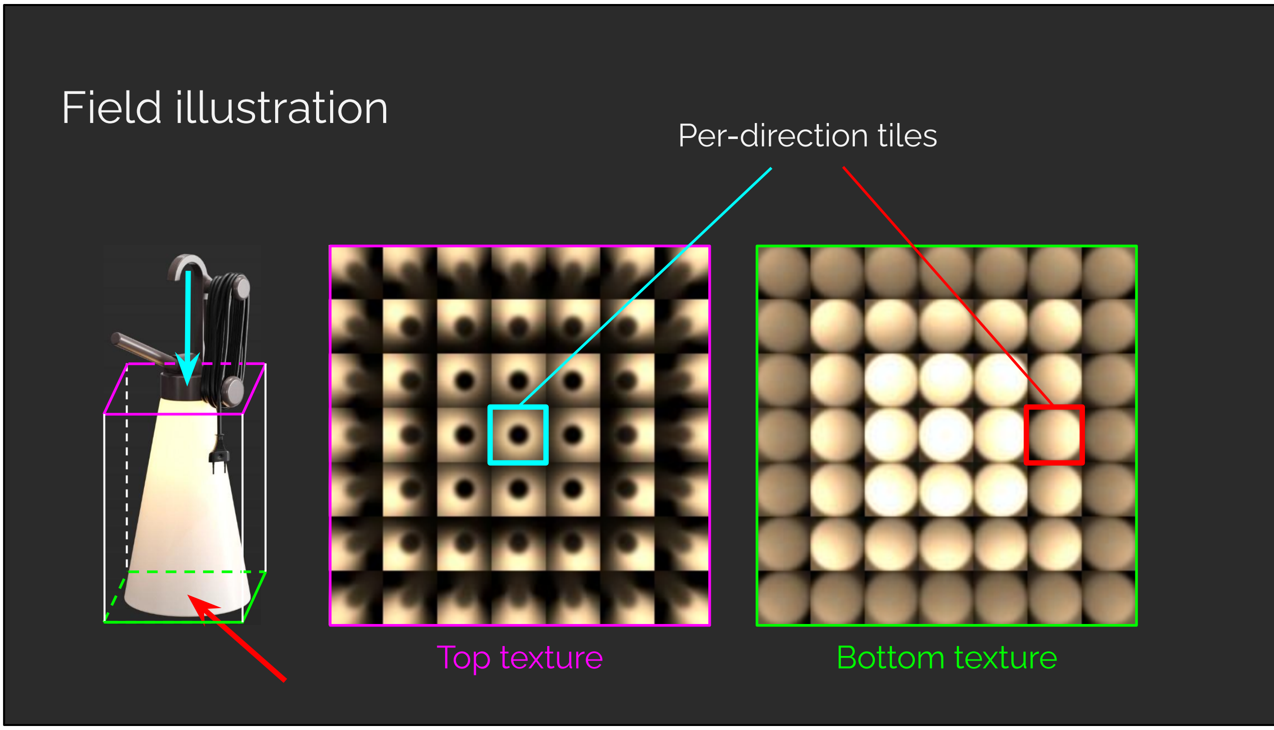


Here's an example with the Tripod luminaire: the unfiltered field shows grid artifacts due to aliasing while our filtering strategies eliminate the grid artifacts.

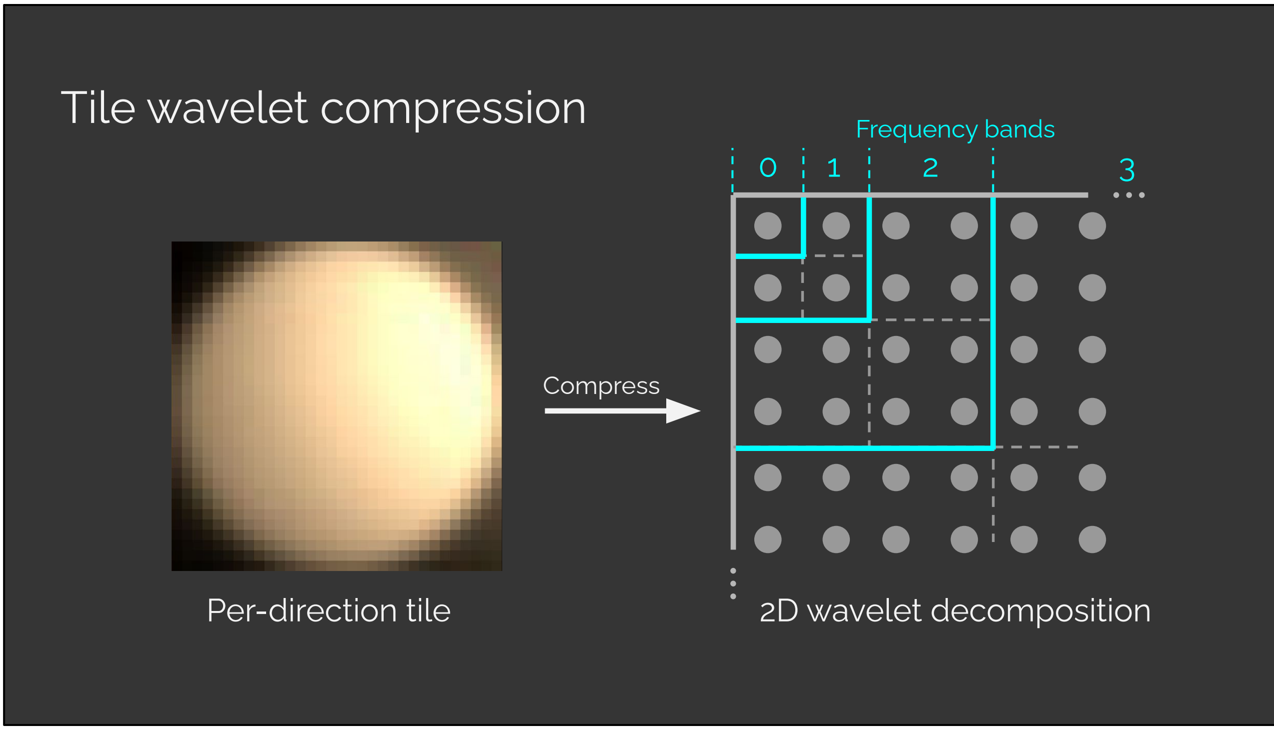
Field illustration



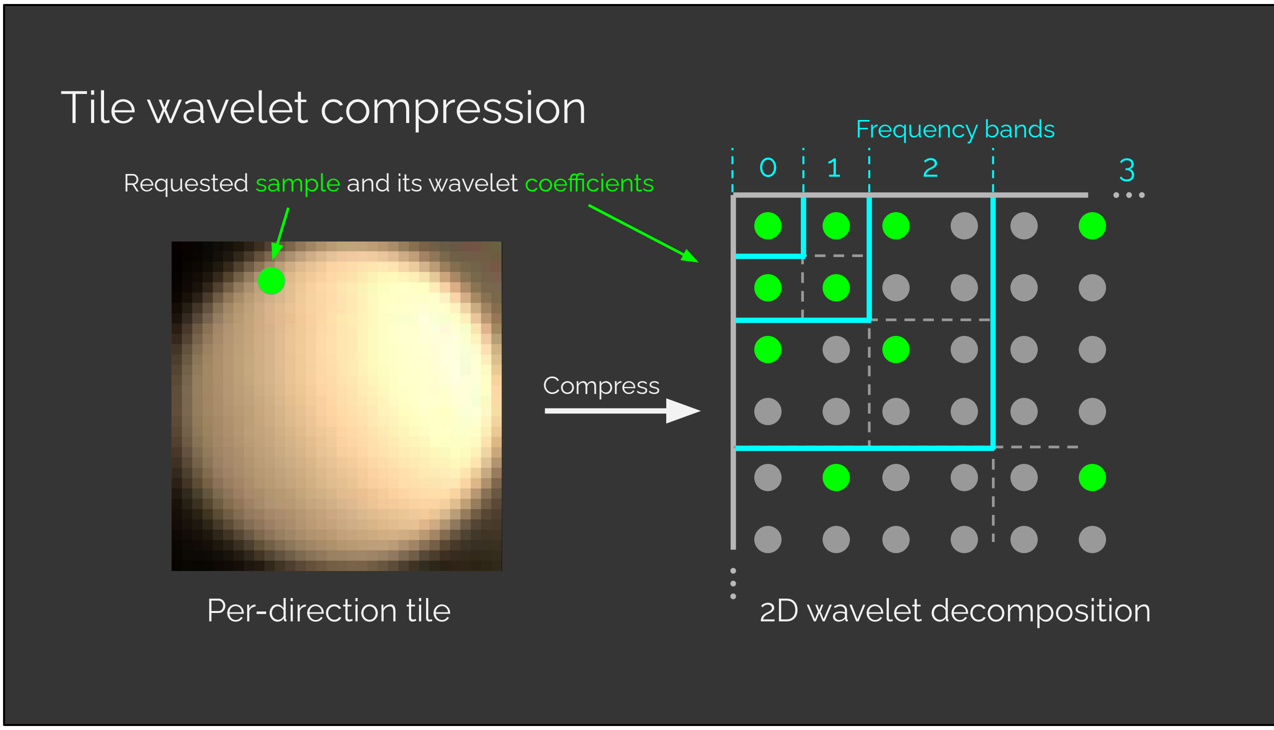
This is a low-resolution illustration of the field. These are two out of the six field textures that correspond to the hull’s box.



These textures are composed of per-direction tiles: each tile encodes how the field looks from one particular direction.



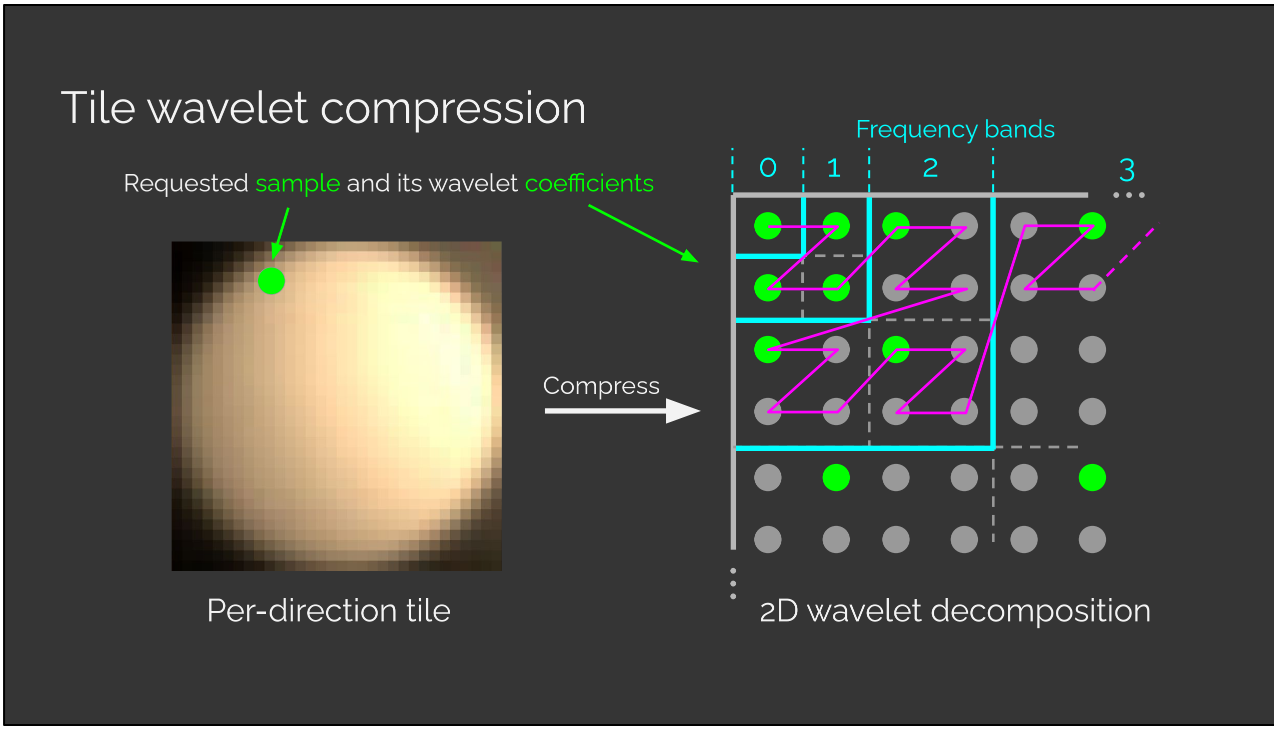
We compress each tile using a wavelet compression scheme and we do not ever decompress the whole field afterwards.



During rendering we load the compressed field in memory and we reconstruct each requested sample from the sparse wavelet coefficients.

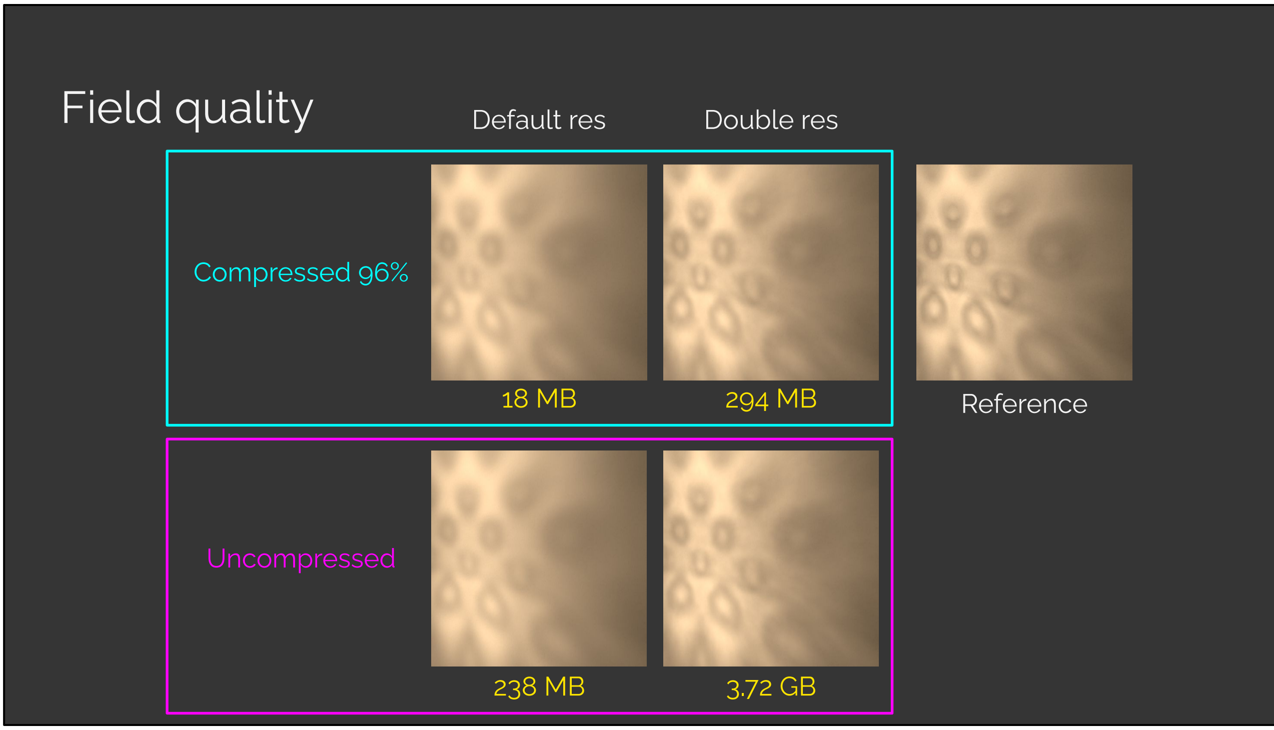
For each reconstruction, we need three wavelet coefficients from each frequency band (as shown on the slide).

Note that most of these coefficients are truncated during compression.



We sort the sparse wavelet coefficients along a z-curve, because the z-curve has a very useful property: it traverses all coefficients in ascending frequency band order.

In this way, the coefficients are in the exact order in which they come in the reconstruction formula, and we can run reconstructions using single cache-friendly passes.

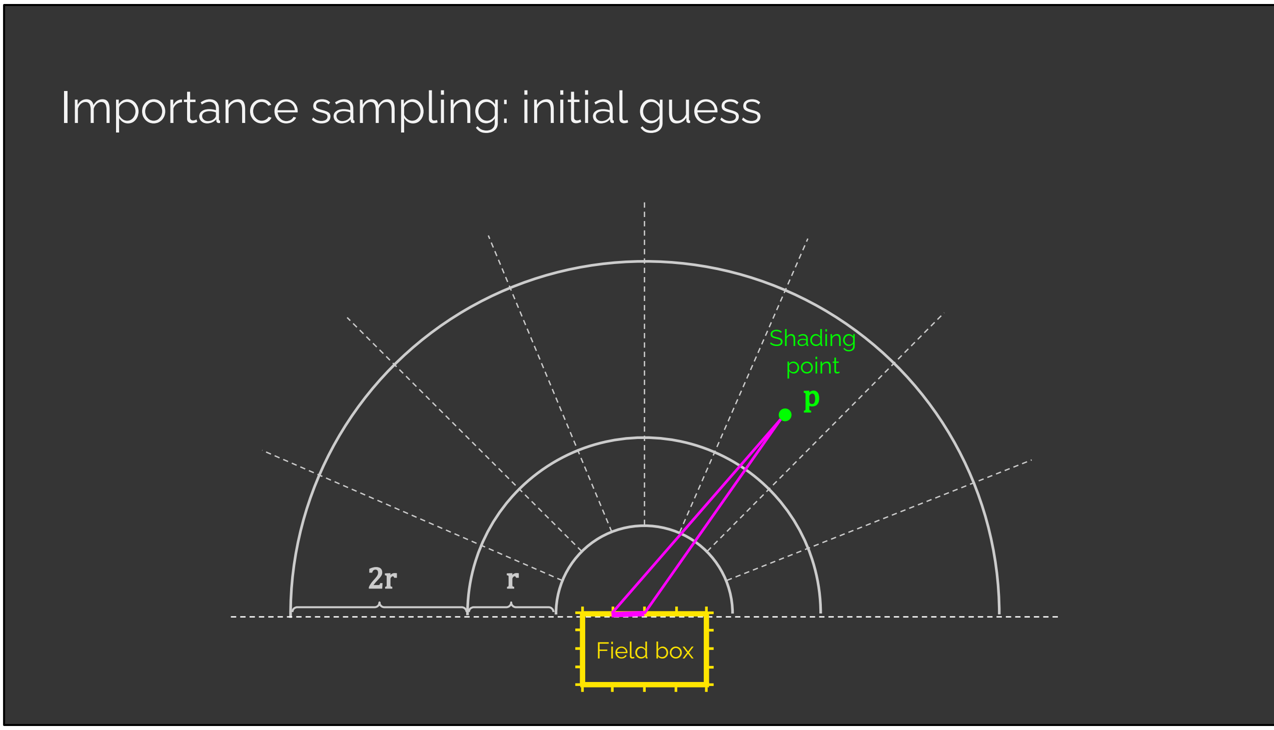


Here's an example showing the field quality of our method for the Tripod luminaire.

The first row has two renders of the compressed field while in the second row are renders of the uncompressed field. The difference between the two is imperceptible.

In the first column, we showcase our default resolution.

In the second column, we show that when we double the field resolution the quality improves while the field size increases 16 times.

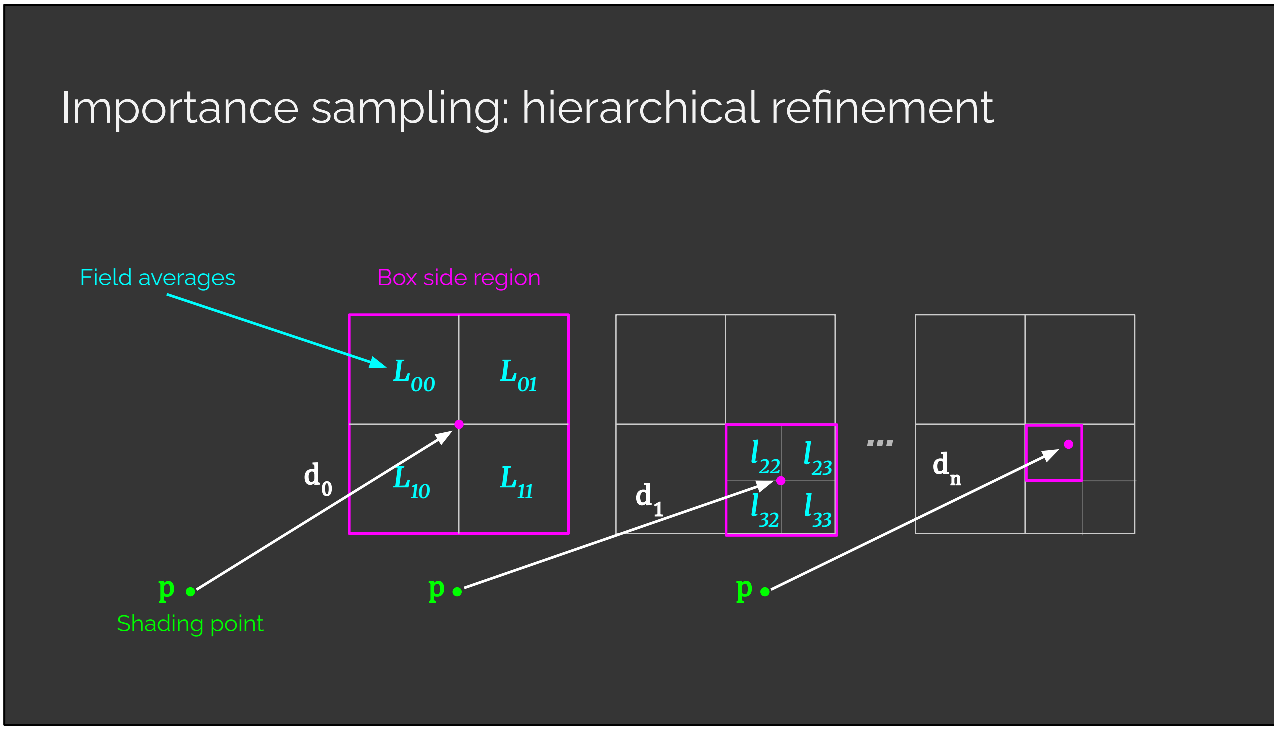


And the last crucial component of our technique - the importance sampling:

For each box side we pre-compute one hemispherical grid: each grid cell contains a 2D sampling distribution over the box side.

To keep the memory low, we use low resolutions.

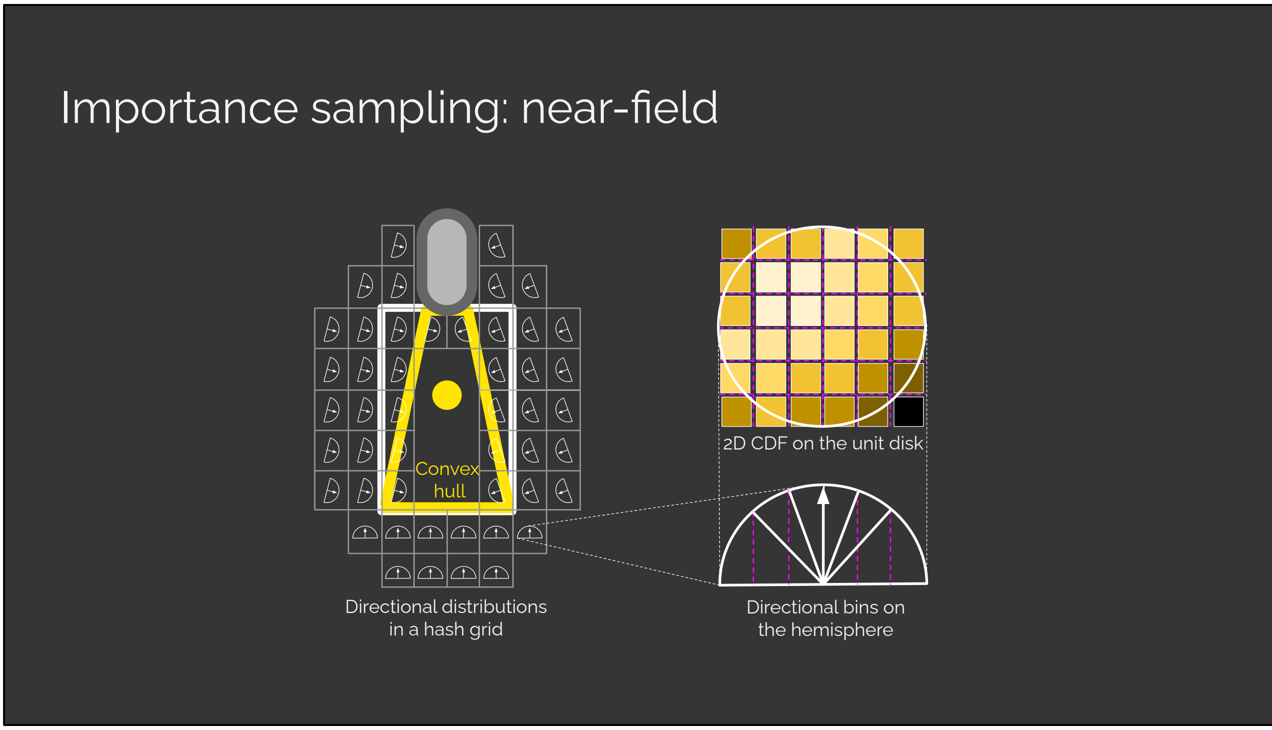
During rendering, we first stochastically pick a visible box side proportional to its solid angle and average intensity and then we use the pre-computed distribution to sample a region on the box side.



We use this region as initial guess.

Then we perform a few refinement iterations over the wavelet frequency bands inside this region, until we reach the full field resolution.

In each iteration, we run partial field reconstructions to compute the averages of the four sub-quadrants, and we choose one of these quadrants stochastically based on the average intensity.



We also need a specialized sampling strategy for shading points close to the field box and in between the box and the convex hull.

For the task, we pre-compute a hash grid of directional distributions defined on hemispheres facing the convex hull.

We use these distributions to generate samples on the hull.

Results

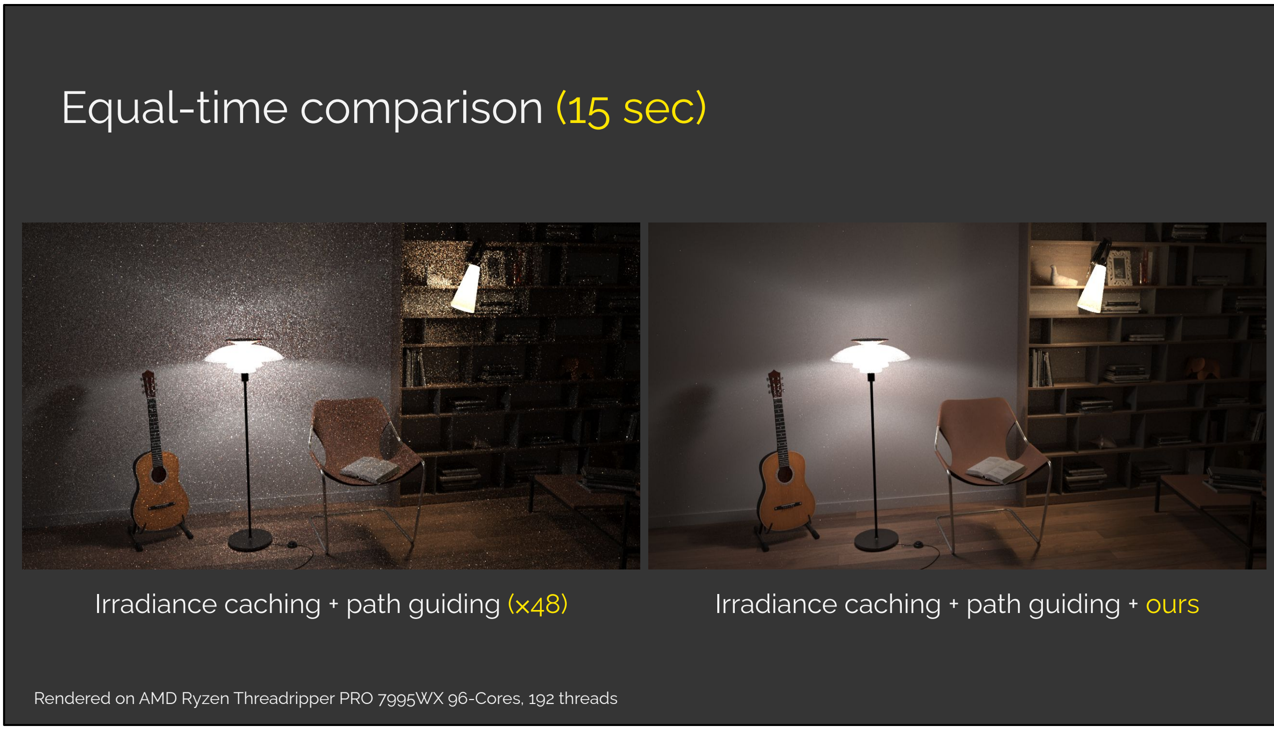
I will show some results.

Path tracing vs. our method

Rendered on AMD Ryzen Threadripper PRO 7995WX 96-Cores, 192 threads

This is a comparison between standard path tracing and our method in a scene with 4 complex luminaires.

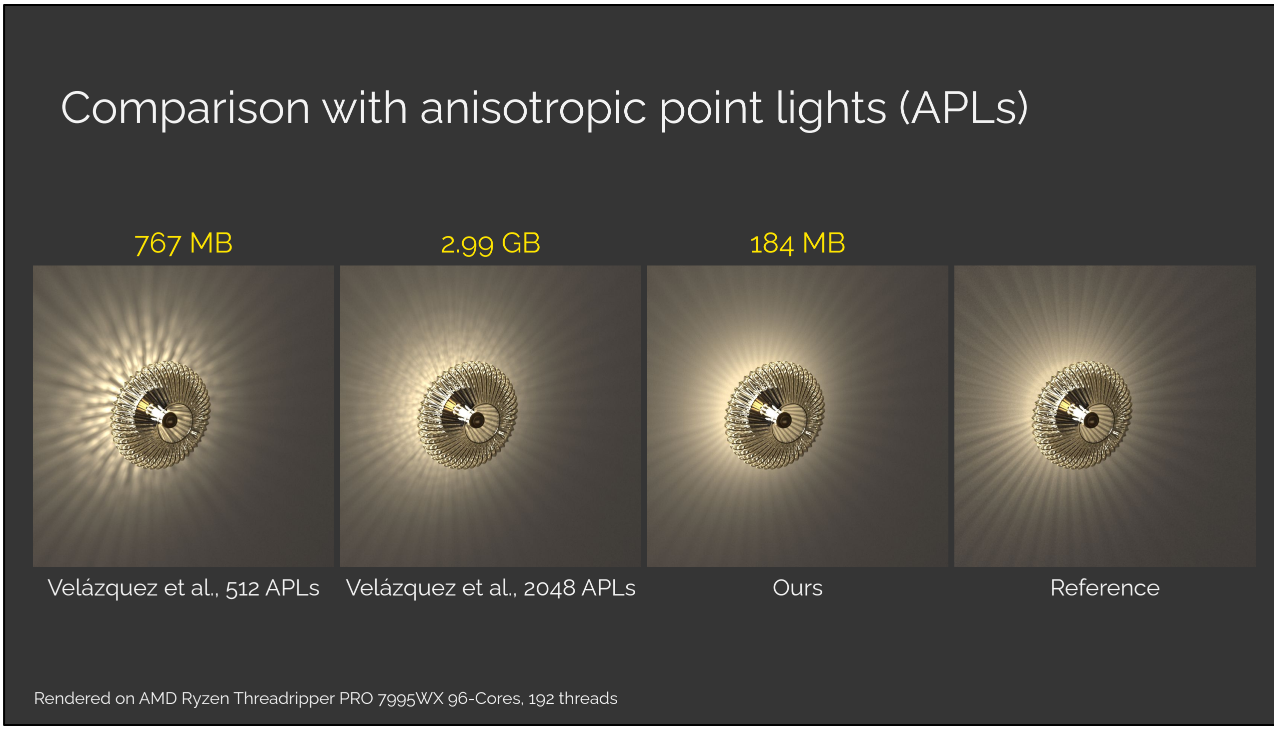
Our method renders a clean image in 3 minutes while the path tracing is still much noisier after 17 hours.



Here’s another scene with two complex luminaires rendered with irradiance caching and path guiding - our renderer’s default configuration.

This is an equal-time comparison for 15 seconds: the left image is rendered without our technique while the right images includes it.

Introducing our technique speeds up the total render time 48 times to achieve equal quality.



This a comparison between our method and the method by Velazquez and others - they pre-compute a cloud of textured point lights sources.

From left to right, we see that their default 512 point lights produce severe aliasing artifacts for this particular luminaire due to its high-frequency illumination.

Increasing the number of point lights reduces the artifacts but does not eliminate it completely.

The third and forth images are our method and a reference.

The error of our method manifests as a slight blur which is controllable through the field resolution.

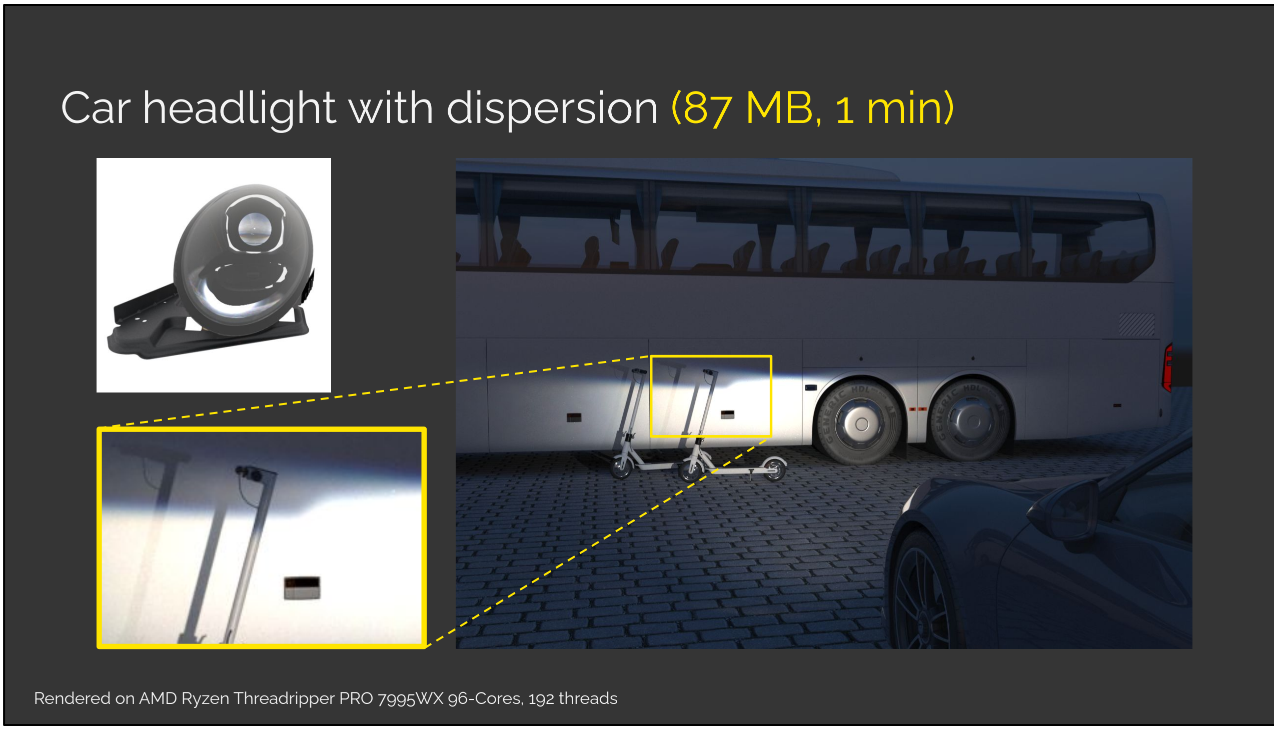
Equal-time comparison with APLs (~150 sec)

Velázquez et al., 2048 APLs

Ours

Rendered on AMD Ryzen Threadripper PRO 7995WX 96-Cores, 192 threads

This is an equal-time comparison between the two methods: the previous method is visibly noisier for the same render time.



And finally, an experiment to render car headlights with our method.

This headlight has a bulb-shaped mesh emitter that is placed behind a barrier, a lens and a transparent cover.

We account for dispersion during photon tracing.

On the slide is a one minute render with an 87 MB cache.

Acknowledgements

- Reviews: Martin Mirbauer, Dian Nikolov, anonymous reviewers
- Car headlight: Oliver Kossatz
- Testing: Emanuele Lecchi, Chaos Cosmos team, V-Ray for 3ds Max QA team
- Workstation: AMD

For more results and more details about our work I refer you to the paper.

Thank you for your attention!