# TWO-LEVEL RADIANCE CACHING

## FOR FAST AND SCALABLE REAL-TIME GLOBAL ILLUMINATION IN GAMES

GUILLAUME BOISSÉ

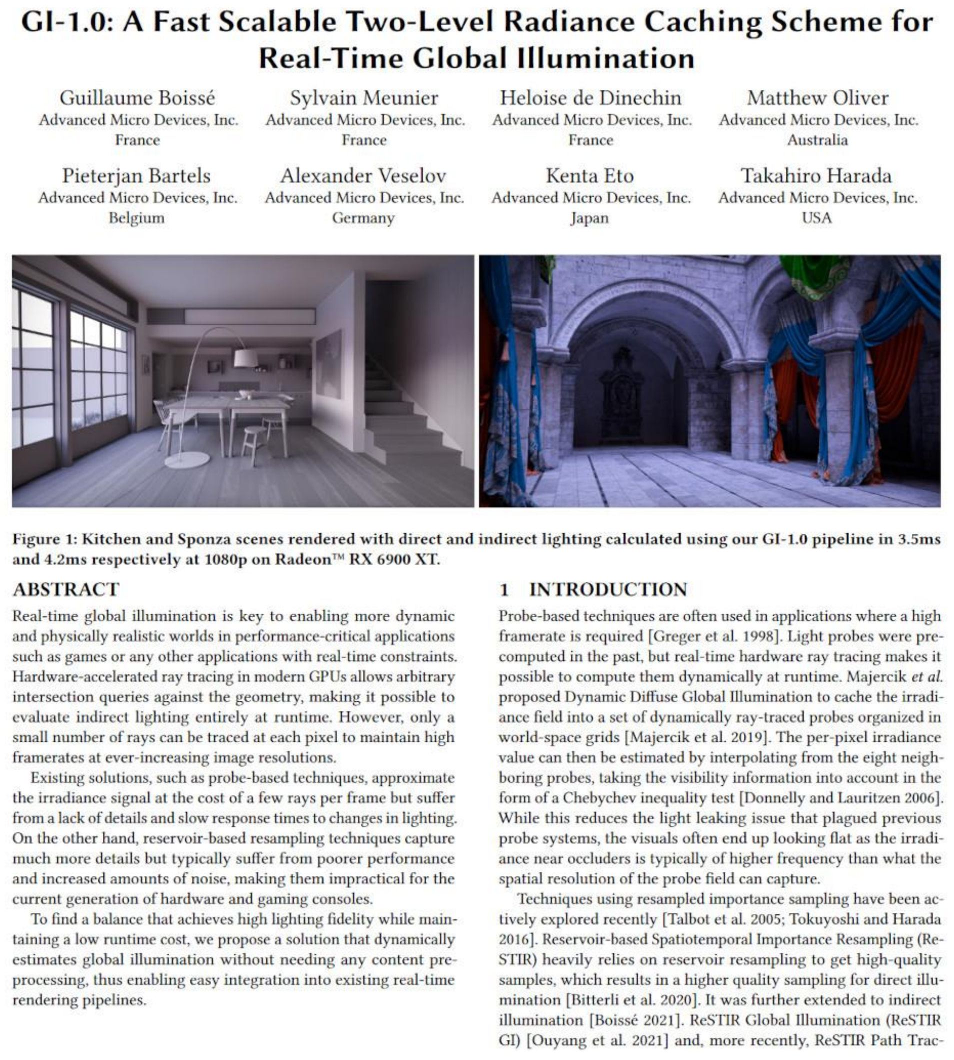
together we advance_

## GI-1.0: TWO-LEVEL RADIANCE CACHING

- Presenting the work of my team at ARR*
  - Sylvain Meunier
  - Heloise Dupont de Dinechin
  - Matthew Oliver
  - Pierterjan Bartels
  - Alexander Veselov
  - Kenta Eto
  - Takahiro Harada

- * Advanced Rendering Research
  - [https://gpuopen.com/advanced-rendering-research/](https://gpuopen.com/advanced-rendering-research/)

- Special thanks to Bruno Stefanizzi and Prashanth Kannan



**GI-1.0: A Fast Scalable Two-Level Radiance Caching Scheme for Real-Time Global Illumination**

Guillaume Boissé, Advanced Micro Devices, Inc., France
Sylvain Meunier, Advanced Micro Devices, Inc., France
Heloise de Dinechin, Advanced Micro Devices, Inc., France
Matthew Oliver, Advanced Micro Devices, Inc., Australia
Pieterjan Bartels, Advanced Micro Devices, Inc., Belgium
Alexander Veselov, Advanced Micro Devices, Inc., Germany
Kenta Eto, Advanced Micro Devices, Inc., Japan
Takahiro Harada, Advanced Micro Devices, Inc., USA

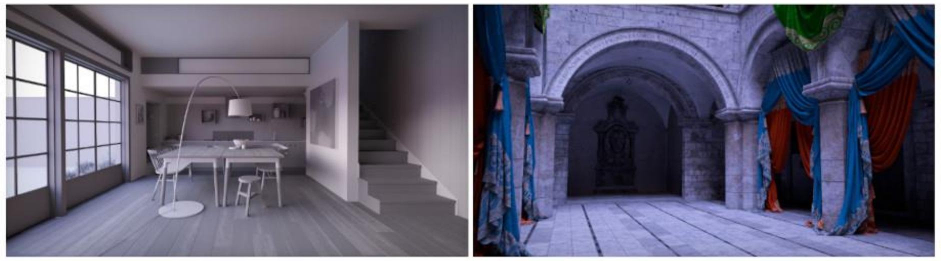


**Figure 1: Kitchen and Sponza scenes rendered with direct and indirect lighting calculated using our GI-1.0 pipeline in 3.5ms and 4.2ms respectively at 1080p on Radeon™ RX 6900 XT.**

**ABSTRACT**

Real-time global illumination is key to enabling more dynamic and physically realistic worlds in performance-critical applications such as games or any other applications with real-time constraints. Hardware-accelerated ray tracing in modern GPUs allows arbitrary intersection queries against the geometry, making it possible to evaluate indirect lighting entirely at runtime. However, only a small number of rays can be traced at each pixel to maintain high framerates at ever-increasing image resolutions.

Existing solutions, such as probe-based techniques, approximate the irradiance signal at the cost of a few rays per frame but suffer from a lack of details and slow response times to changes in lighting. On the other hand, reservoir-based resampling techniques capture much more details but typically suffer from poorer performance and increased amounts of noise, making them impractical for the current generation of hardware and gaming consoles.

To find a balance that achieves high lighting fidelity while maintaining a low runtime cost, we propose a solution that dynamically estimates global illumination without needing any content preprocessing, thus enabling easy integration into existing real-time rendering pipelines.

**1 INTRODUCTION**

Probe-based techniques are often used in applications where a high framerate is required [Greger et al. 1998]. Light probes were precomputed in the past, but real-time hardware ray tracing makes it possible to compute them dynamically at runtime. Majercik *et al.* proposed Dynamic Diffuse Global Illumination to cache the irradiance field into a set of dynamically ray-traced probes organized in world-space grids [Majercik et al. 2019]. The per-pixel irradiance value can then be estimated by interpolating from the eight neighboring probes, taking the visibility information into account in the form of a Chebychev inequality test [Donnelly and Lauritzen 2006]. While this reduces the light leaking issue that plagued previous probe systems, the visuals often end up looking flat as the irradiance near occluders is typically of higher frequency than what the spatial resolution of the probe field can capture.

Techniques using resampled importance sampling have been actively explored recently [Talbot et al. 2005; Tokuyoshi and Harada 2016]. Reservoir-based Spatiotemporal Importance Resampling (ReSTIR) heavily relies on reservoir resampling to get high-quality samples, which results in a higher quality sampling for direct illumination [Bitterli et al. 2020]. It was further extended to indirect illumination [Boissé 2021]. ReSTIR Global Illumination (ReSTIR GI) [Ouyang et al. 2021] and, more recently, ReSTIR Path Trac-

## MOTIVATION

- Why global illumination?
  - Accurate occlusion helps ground the scene elements
  - Color bleeding produces richer, more believable lighting

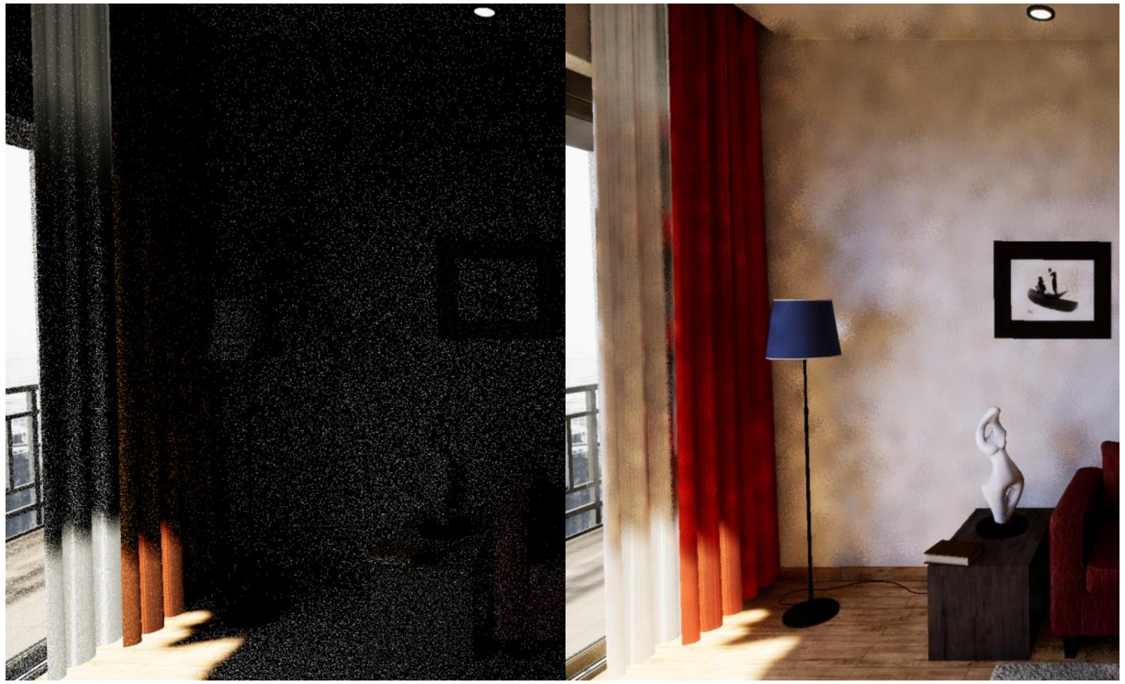
- Why real time?
  - Baked illumination can be high quality (lightmaps, pre-calculated probes, etc.)
  - But limited ability to adapt to changing geometry and lighting
  - Calculating global illumination at runtime allows for more dynamic worlds…
  - … and can improve content creation iteration workflow

- Ray tracing is finally available in HW ☺
  - But it remains expensive ☹
  - So, we need to find solutions that achieve high quality lighting at reduced sample rates

## EXISTING WORK

- Very active area of research
  - DDGI [Majercik et al. 2019]
  - ReSTIR GI [Ouyang et al. 2021]
  - Surfels [Brinck et al. 2021]
  - etc.

- Screen-space radiance caching [Wright 2021]
  - Interesting trade-off between path tracing and probes
  - Less noise using less samples…
  - … sounds like a good idea ☺



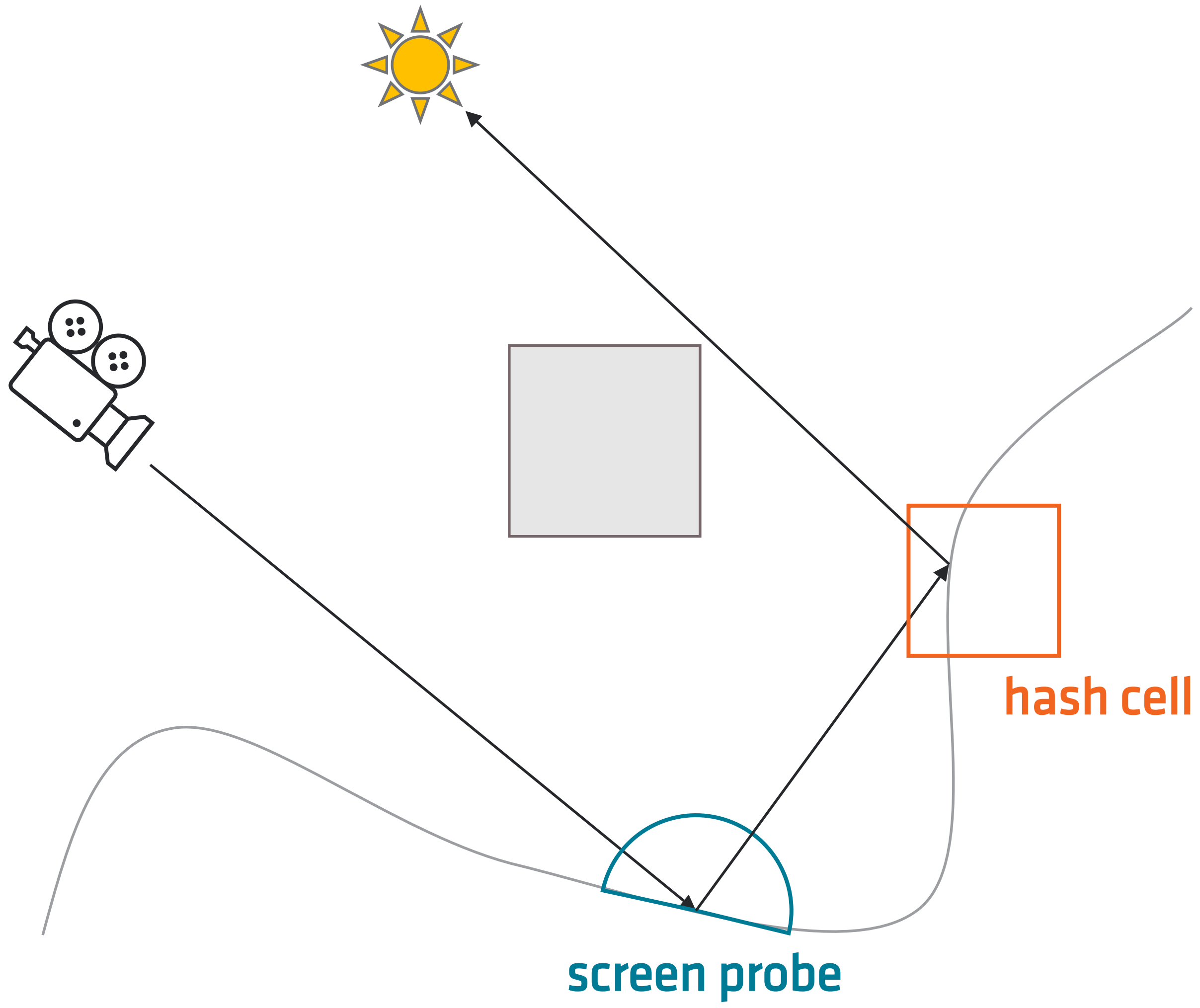
2-spp path-tracing vs. ½-spp screen probes (courtesy of [Wright 2021])

## GI-1.0: REAL-TIME DYNAMIC GLOBAL ILLUMINATION PIPELINE

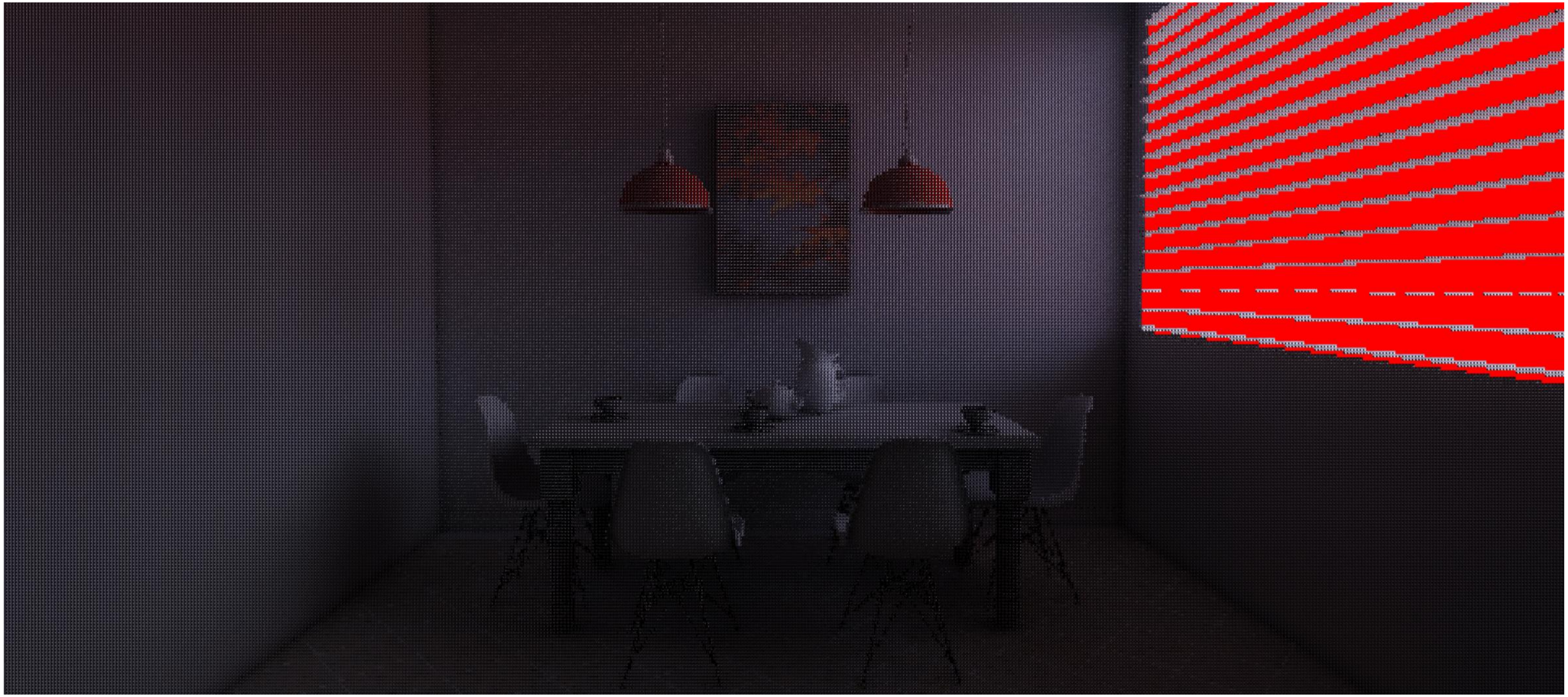
- Implements a **two-level radiance caching** scheme
  - Make the most of every ray…
  - … by caching the radiance…
  - … and using it for sampling

- The **screen probes** are the 1st level of caching
  - Positioned on primary surfaces only
  - Cache radiance across the hemisphere
  - High fidelity as probes are numerous

- The **hash cells** are the 2nd level of caching
  - Positioned anywhere in world space
  - Cache reflected radiance for a direction
  - Less detailed but stable and persistent



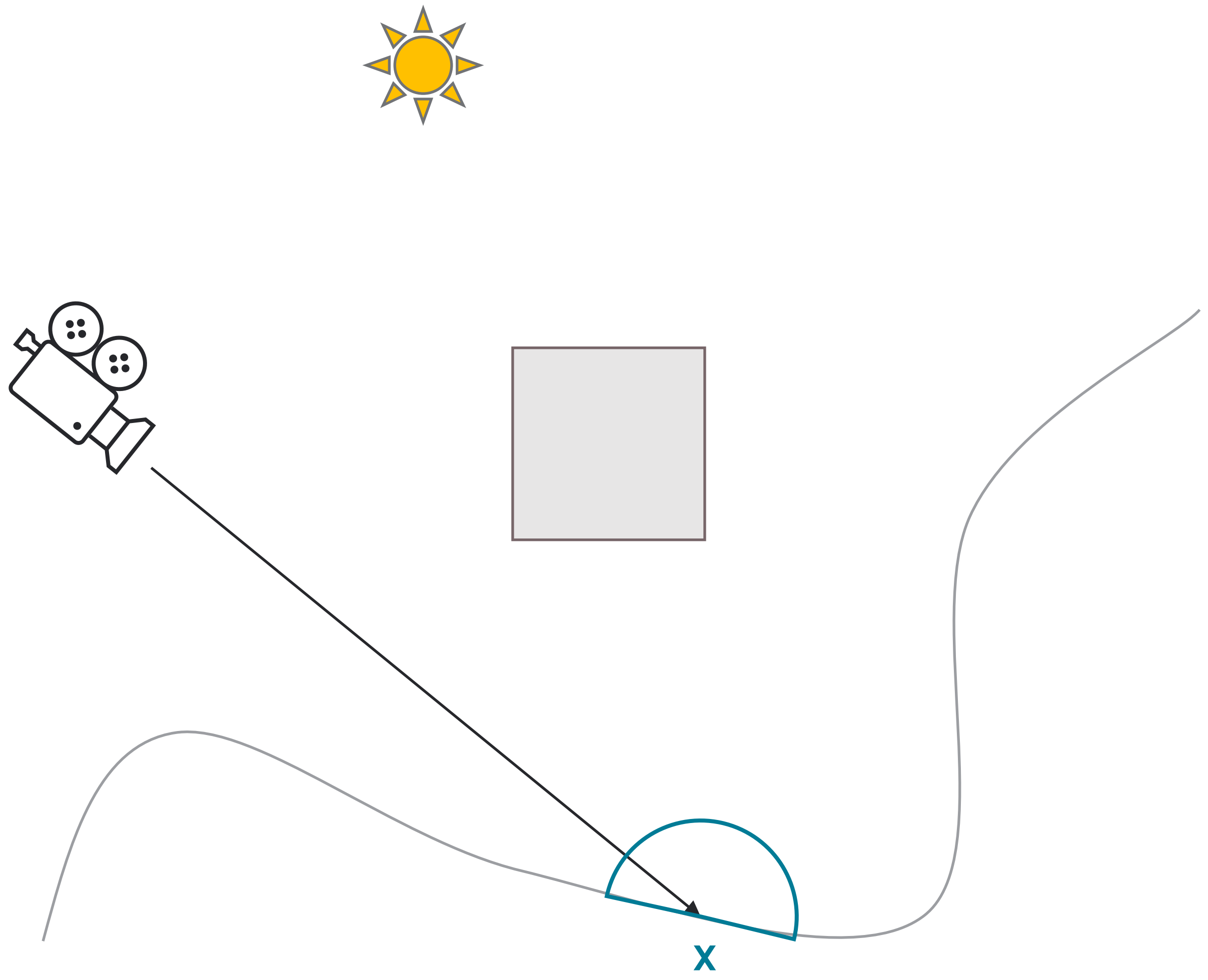
## SCREEN PROBES CACHE THE RADIANCE IN SCREEN SPACE



## SCREEN-SPACE RADIANCE CACHING

- We want to evaluate the lighting at x
  - "Trace" the primary rays using rasterization
  - Store the hemispherical radiance into a probe
  - Can use octahedral mapping [Cigolle et al. 2014]
  - Interpolate per pixel from 4 neighboring probes

- Fixes several issues from traditional probe systems
  - Probe placement is trivial
    - Screen probes are spawned directly on pixels
    - Easier to get rid of light and shadow leaks
  - Probe density is optimal for a given viewpoint
    - No computation is wasted on non-contributing probes
    - Able to capture near occluders for higher quality lighting

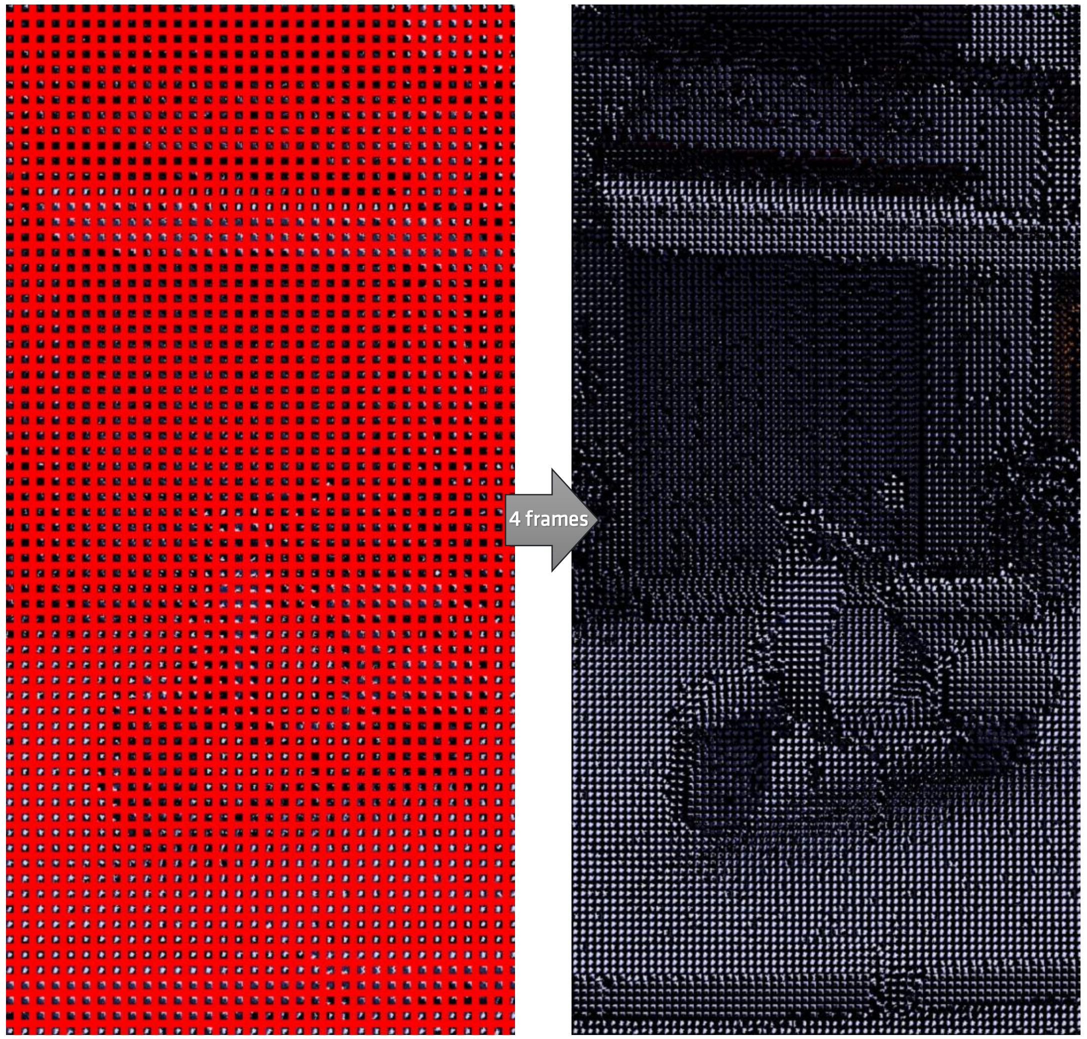


# TEMPORAL UPSCALE

- Cache up to one 8x8 probe for each 8x8 screen tile
- Re-calculating everything requires 1 ray per pixel ☹
- So, only re-calculate ¼ each frame and upscale ☺

- The amount of re-calculated probes is arbitrary
  - Can go higher; makes the system more reactive
  - Can go lower; faster but higher latency to changes

- Overall, the final image quality isn't impacted much
- All sampling and filtering at ¼ rate; big perf. boost!
- Only reprojection and interpolation run at full rate



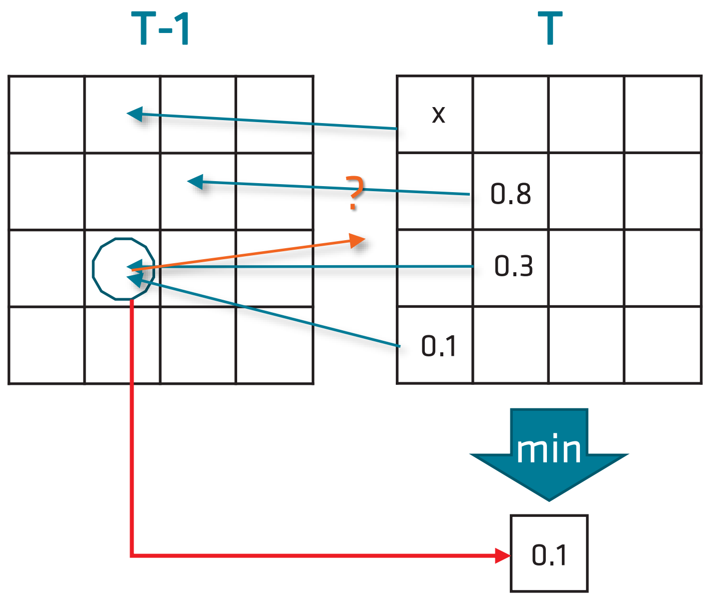
# TEMPORAL REPROJECTION

- Single "two-part" dispatch going over every pixel at target resolution
  - Typically, the resolution of the g-buffers (depth, normals, motion vectors, etc.)
  - Not necessarily equal to the output resolution (FidelityFX™ Super Resolution, etc.)

- 1st part of the dispatch goes over every pixel
  - Locate the nearest probe in the previous frame (if any)
  - Determine whether it is valid for placement on the pixel
  - Calculate a **reprojection score**, quantifying the reuse error

- We then pick the best pixel candidate across the group
  - Encode the **reprojection score** onto the top 16 bits of a 32-bit integer
  - And the lane index onto the bottom 16 bits of that same integer
  - Use a single **LDS**-based `InterlockedMin()` operation (Local Data Share)

- 2nd part of the dispatch reads back the reprojection result from **LDS**
  - Reproject the previous probe onto the new pixel in the current frame
  - Flag the tile as a disocclusion if no candidate was submitted

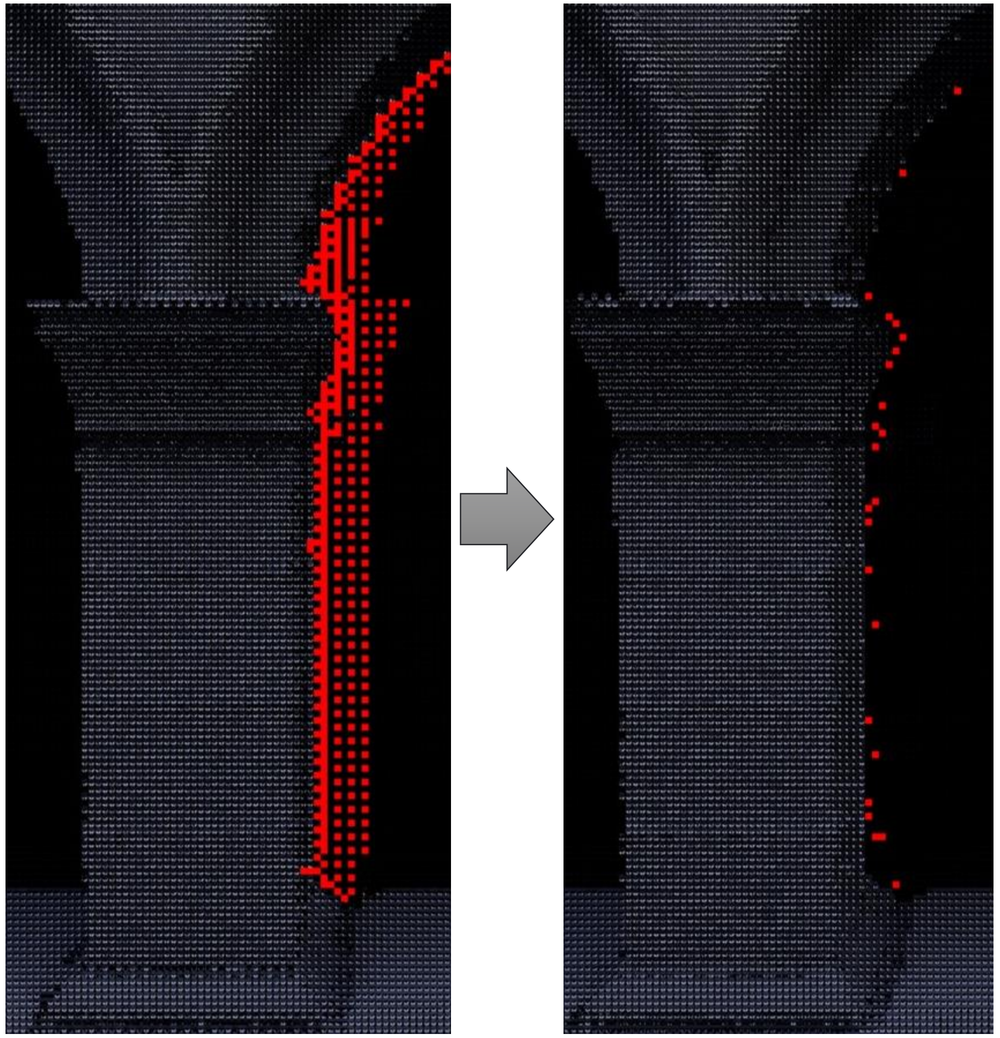


Use pixel as destination for reprojected probe

# ADAPTIVE SAMPLING

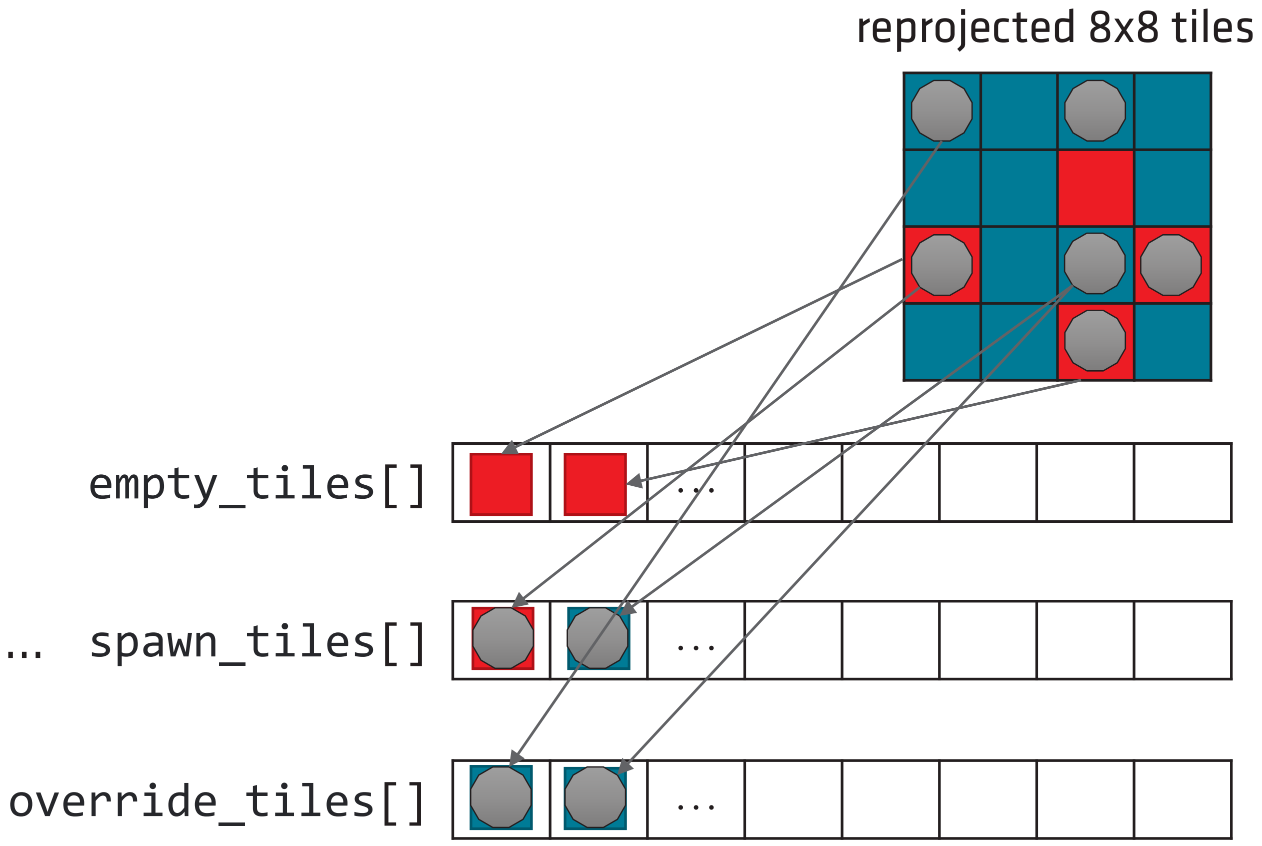
- We use a fixed global pattern to spawn new probes
- Which undergoes compaction to better fill the GPU
- But disocclusions show holes, where ¾ is missing ☹

- Re-balance the rays away from converged probes…
- … and towards filling the holes in a stochastic way
- Less noticeable disocclusions at no extra cost ☺

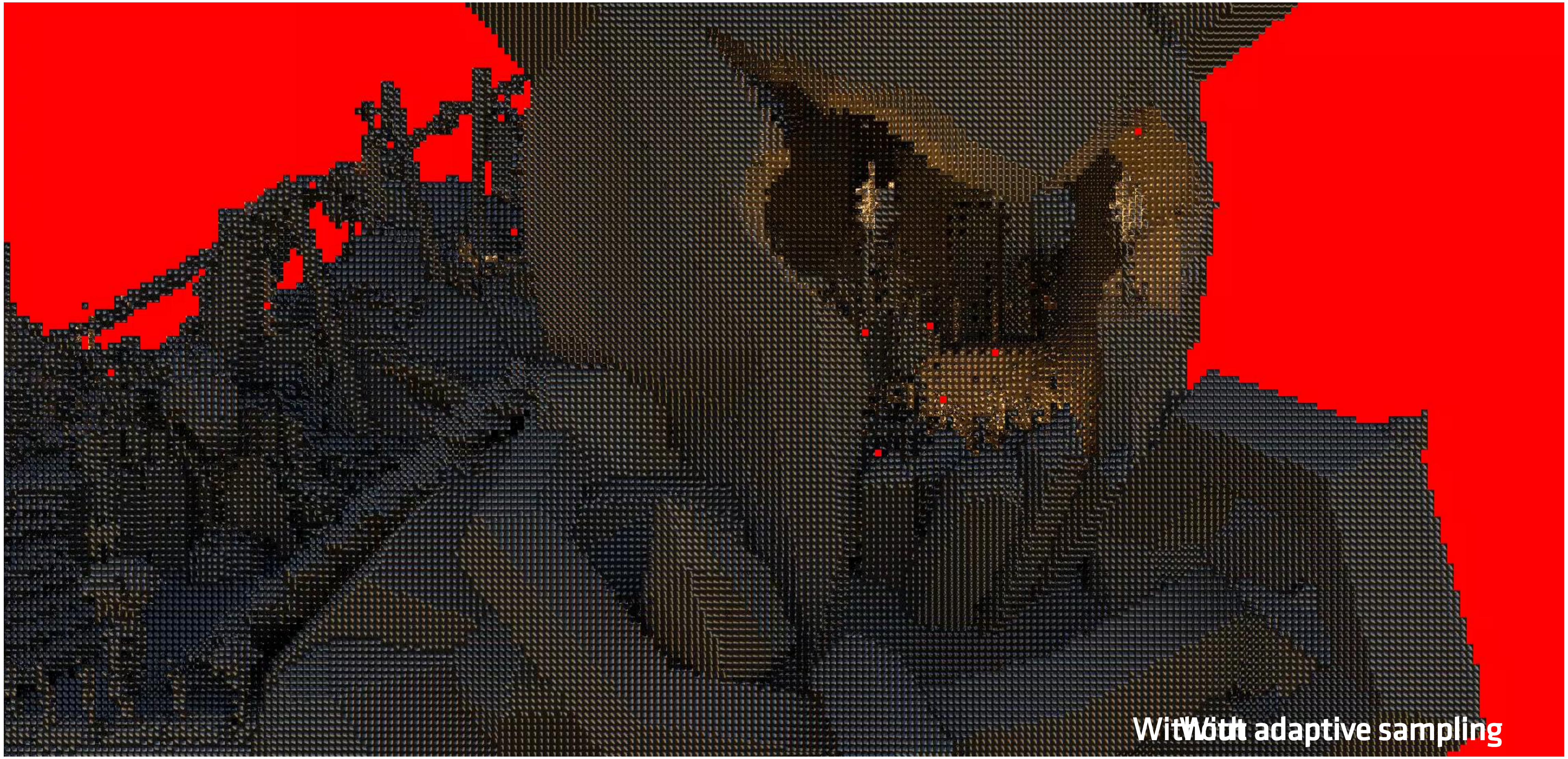


# PROBE PATCHING

- Start with the results of our reprojection
  - Some tiles succeeded reprojection
  - Some tiles failed…
- Push out the tiles that failed to a queue
- Spawn the new probes using our regular pattern…
- … and push spawned converged tiles to another queue



```
empty_tile = empty_tiles[tid];
override_tile = override_tiles[random_index];
InterlockedExchange(spawn_tiles[override_tile], empty_tile);
```


Without adaptive sampling
With adaptive sampling

## RAY GUIDING

- We can also use the reprojection to estimate the content of the new probes without tracing a ray
  - We call this process **hemisphere reconstruction**
  - Uses reprojected probes in a 3x3 tile neighborhood

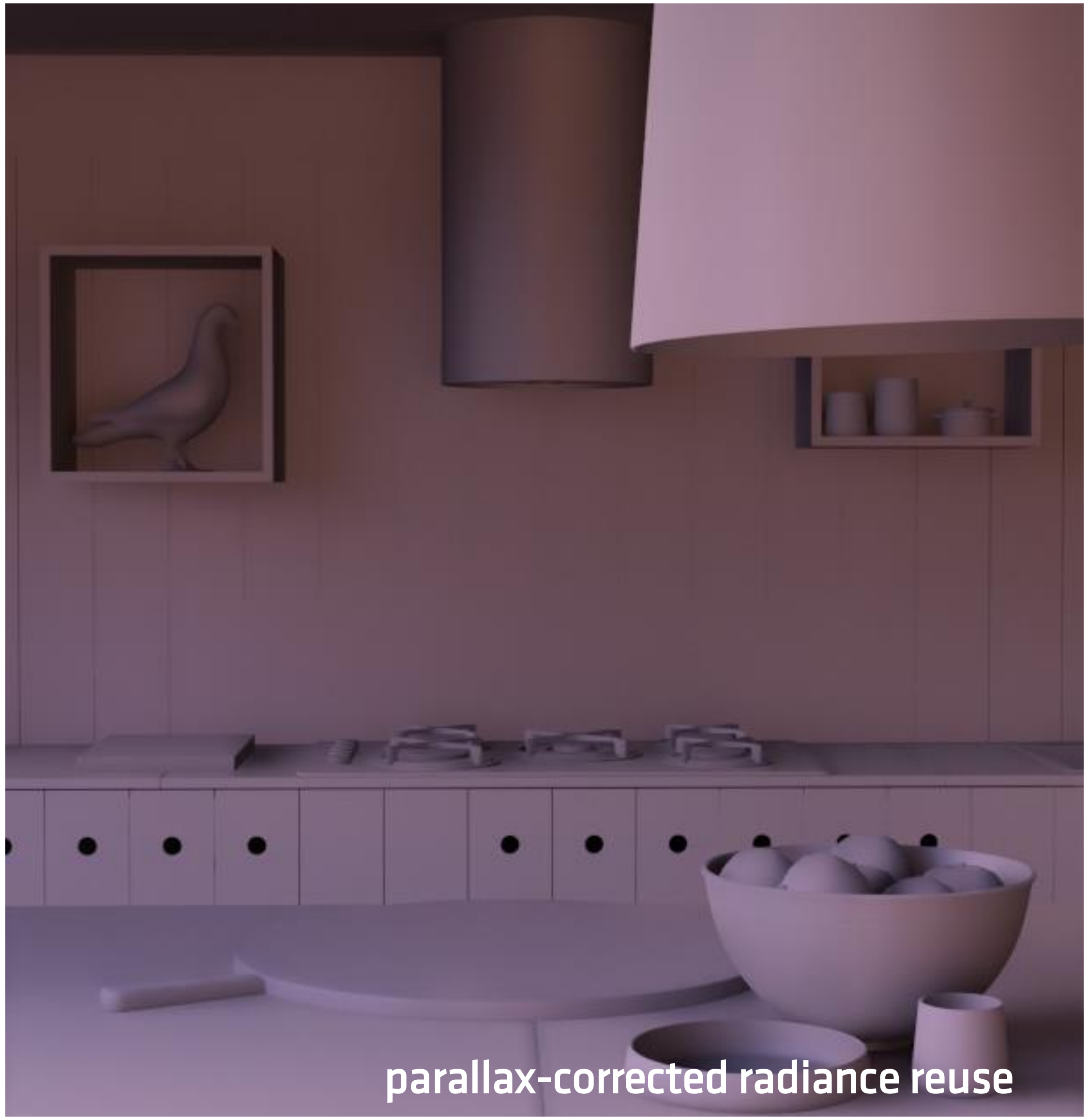
- We can use this to guide the rays [Wright 2021]
  - When the **reconstruction** succeeds anyway…
  - … falling back to **uniform sampling** otherwise

- If the **reconstruction** succeeds, use it for sampling
  - Store the cell's luminance in **LDS** (Local Data Share)
  - Scan into a **CDF** (Cumulative Distribution Function)
  - Use **CDF** to pick a cell proportional to its intensity

# PARALLAX-CORRECTED RADIANCE REUSE

- For the guiding to be successful, we want the **hemisphere reconstruction** to be precise
  - Uses reprojected probes in 3x3 tile neighborhood
  - Lighting parallax may be different to shading point
  - Can make it difficult to hit bright areas consistently

- So, do a parallax correction during **reconstruction**
  - Reproject the direction into newly spawned probe
  - Scatter radiance w/ 4x **LDS**-based `InterlockedAdd()`
  - Normalize and resolve history radiance value into cell



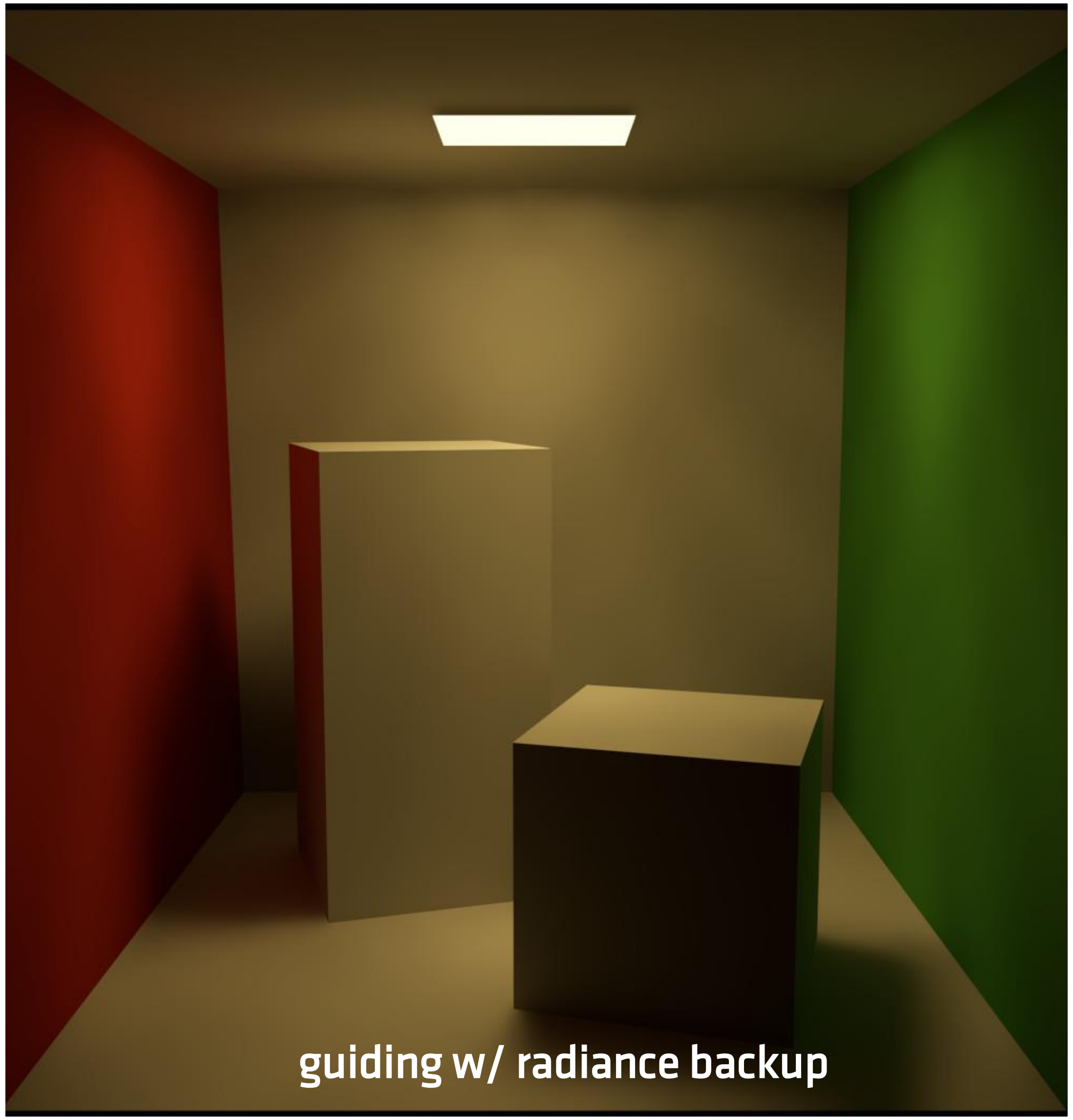
## RADIANCE BACKUP

- Due to the guiding, some cells are not visited leading to a loss of energy if left black

- There are multiple solutions for this
  - Trace more rays to ensure every cell has at least a ray
  - Do not trace more rays & rely more heavily on history

- We would like to avoid option 1
  - Constant ray budget helps performance stability
  - But option 2 introduces strong temporal artefacts

- So, we approximate the missing information
  - Average the radiance from the traced cells in **LDS**…
  - … and distribute uniformly across the untraced ones

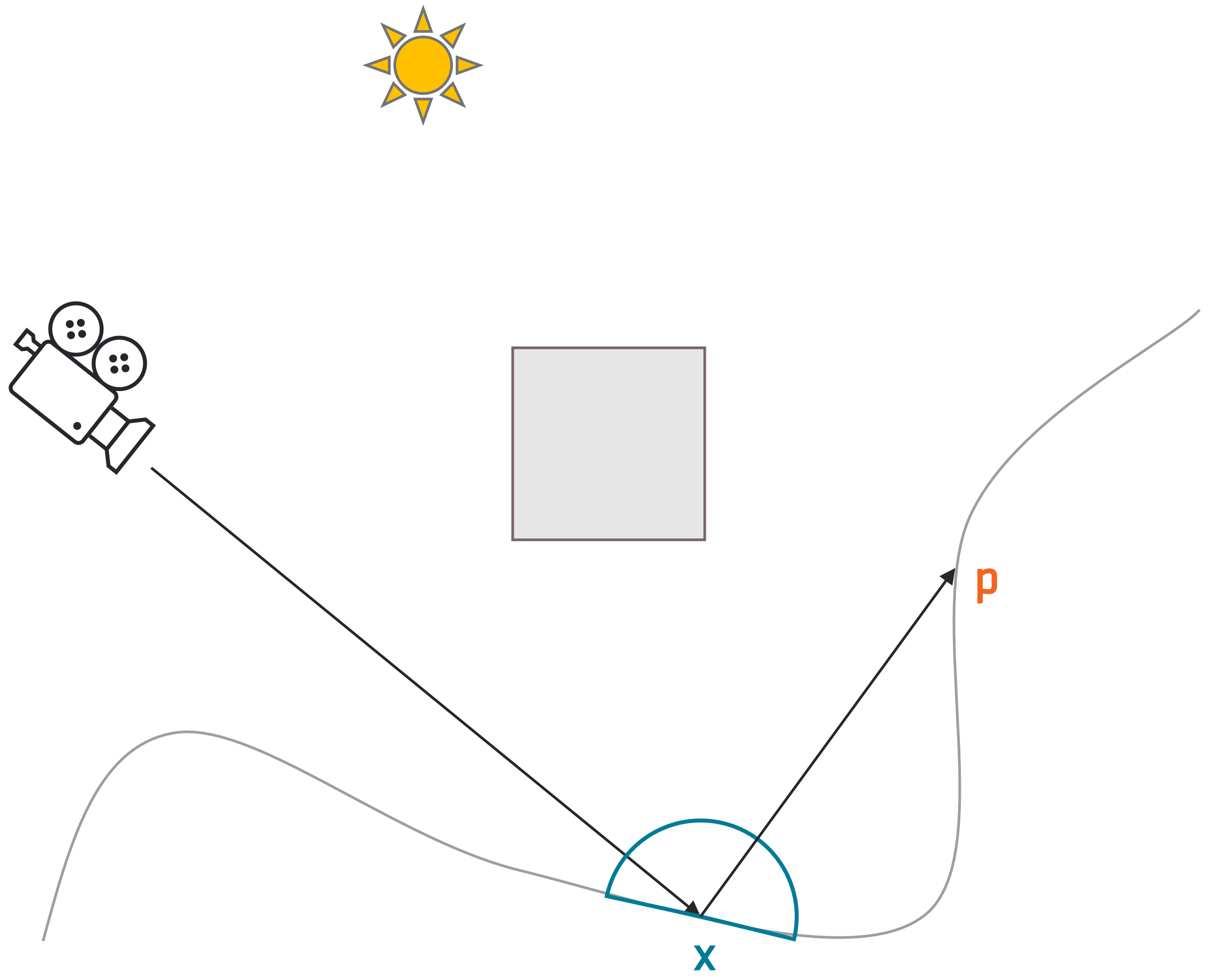


# RADIANCE BLENDING

- All we've done so far is pick a cell to visit
  - Let's generate a 2D jitter inside that cell…
  - … and back project to a world-space direction
  - We can now trace the ray as a closest hit query

- We then need to evaluate the lighting at p
  - This is the job of our 2nd level of caching
  - So, we'll skip this for now…
  - … and assume we're getting some radiance back

- We can blend the **radiance estimate** with our history
  - Exponential moving average [Karis 2014] hurts quality
  - Indeed, probe cells cover a large area of the scene…
  - … leading to excessive blurring of the radiance

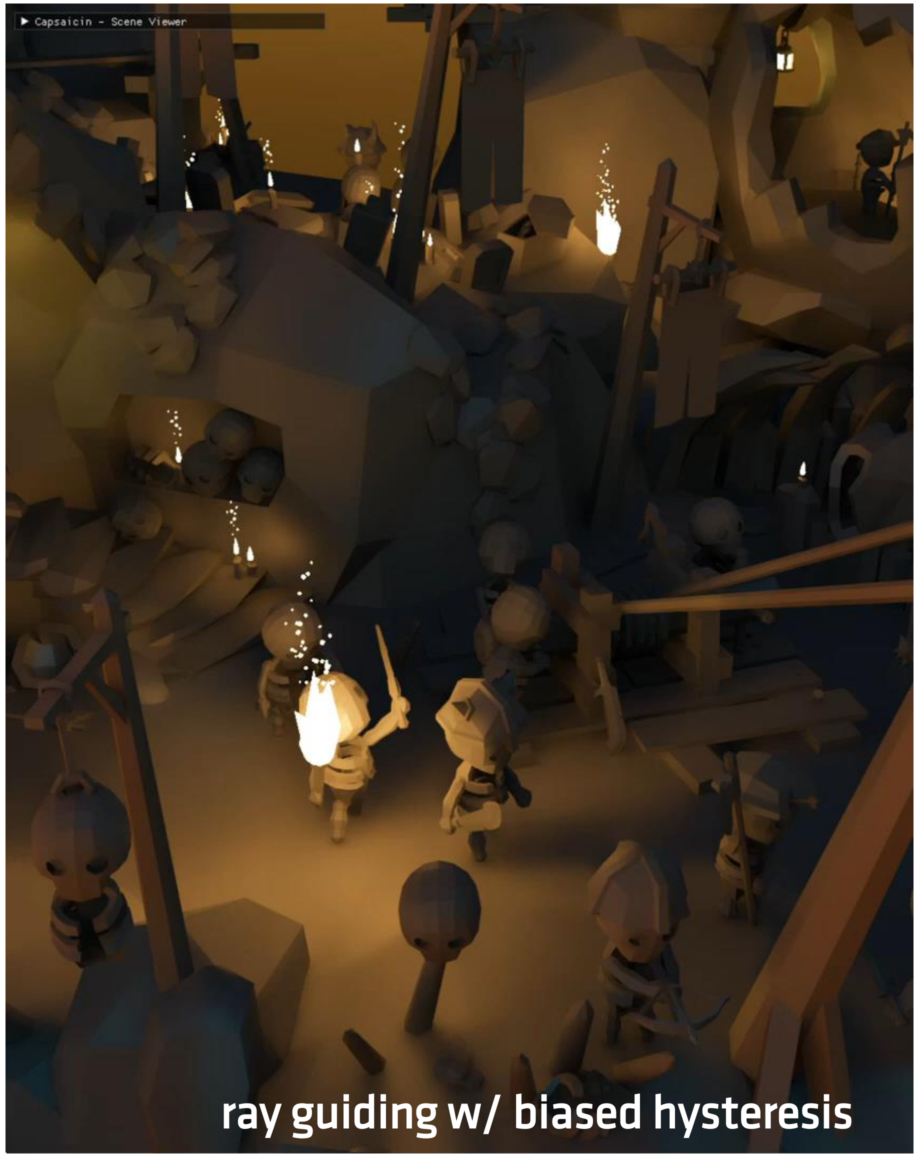


# BIASED TEMPORAL HYSTERESIS

- We adapt the temporal blending factor instead
  - Use error between history and new estimate
  - Favors darker samples over brighter ones

- Helps stabilize the image temporally
  - While preserving occlusion and shadows…
  - … at the expense of some image darkening

- In effect, eliminate light sources smaller than the granularity of the guiding
  - Guiding efficiency is limited by the probe's resolution
  - Eliminate signal from sources we cannot hit reliably



ray guiding w/ biased hysteresis

## PROBE MASKING

- Probes can be on any of the 8x8 pixels within a tile

- We therefore maintain a **probe mask** texture
  - Storing pixel position of each probe inside the tile…
  - … or an invalid marker if the reprojection has failed

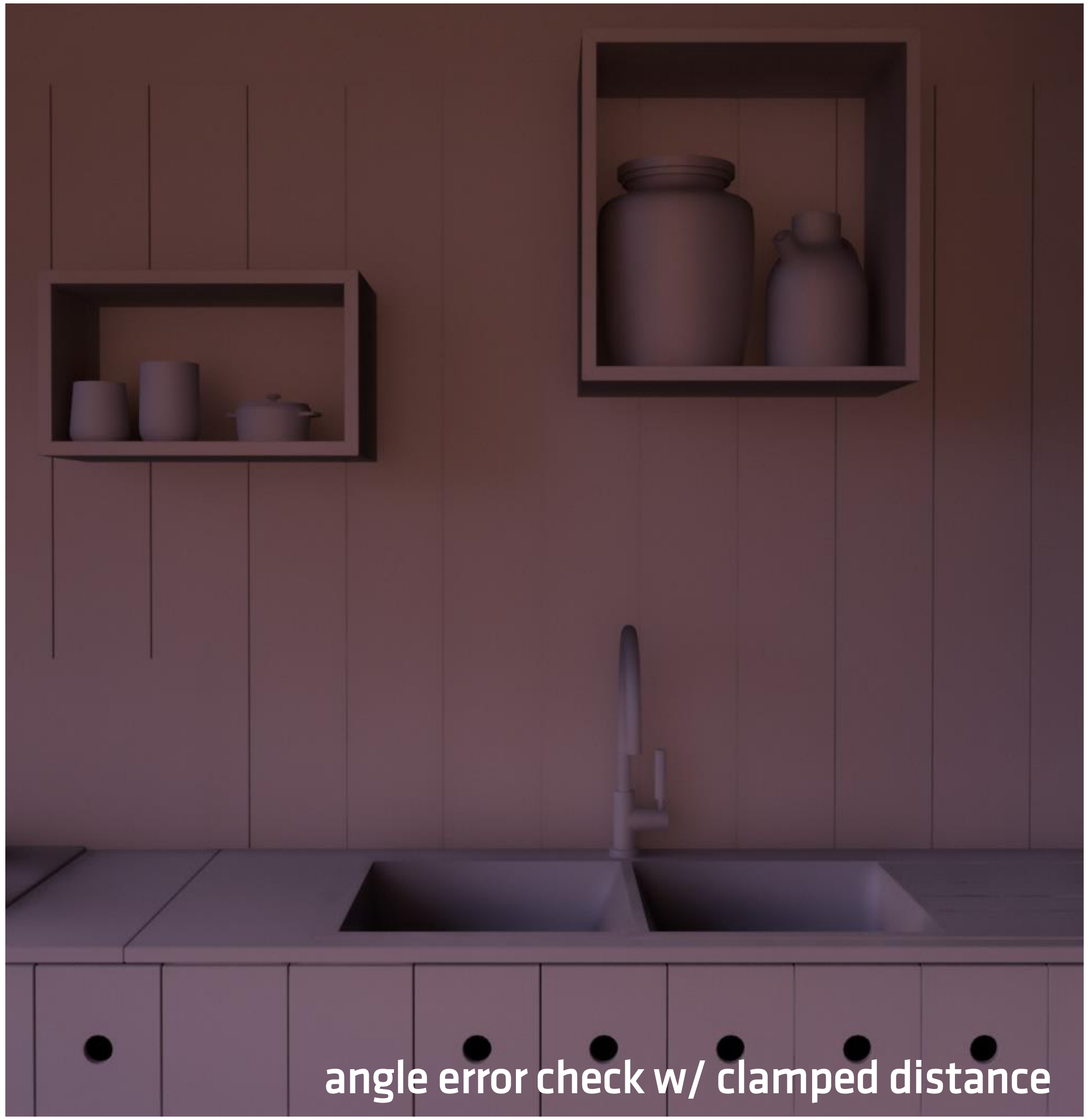
- Areas with missing probes can be large
  - We simply build a mip chain of the **probe mask**
  - By keeping 1st valid probe from the 2x2 upper values

- The mask is used to do large-scale probe search
  - Used extensively during the probes spatial filtering
  - As well as during the final interpolation pass…

## PROBE FILTERING

- Simple 7x7 separable blur in probe space
  - Use probe mask for searching across holes
  - Use distance to plane to avoid light leaks

- Estimate angle error after hit reprojection
  - Using **clamped distance** to hit point [Wright 2021]
  - Prevents loss of short scale occlusion details

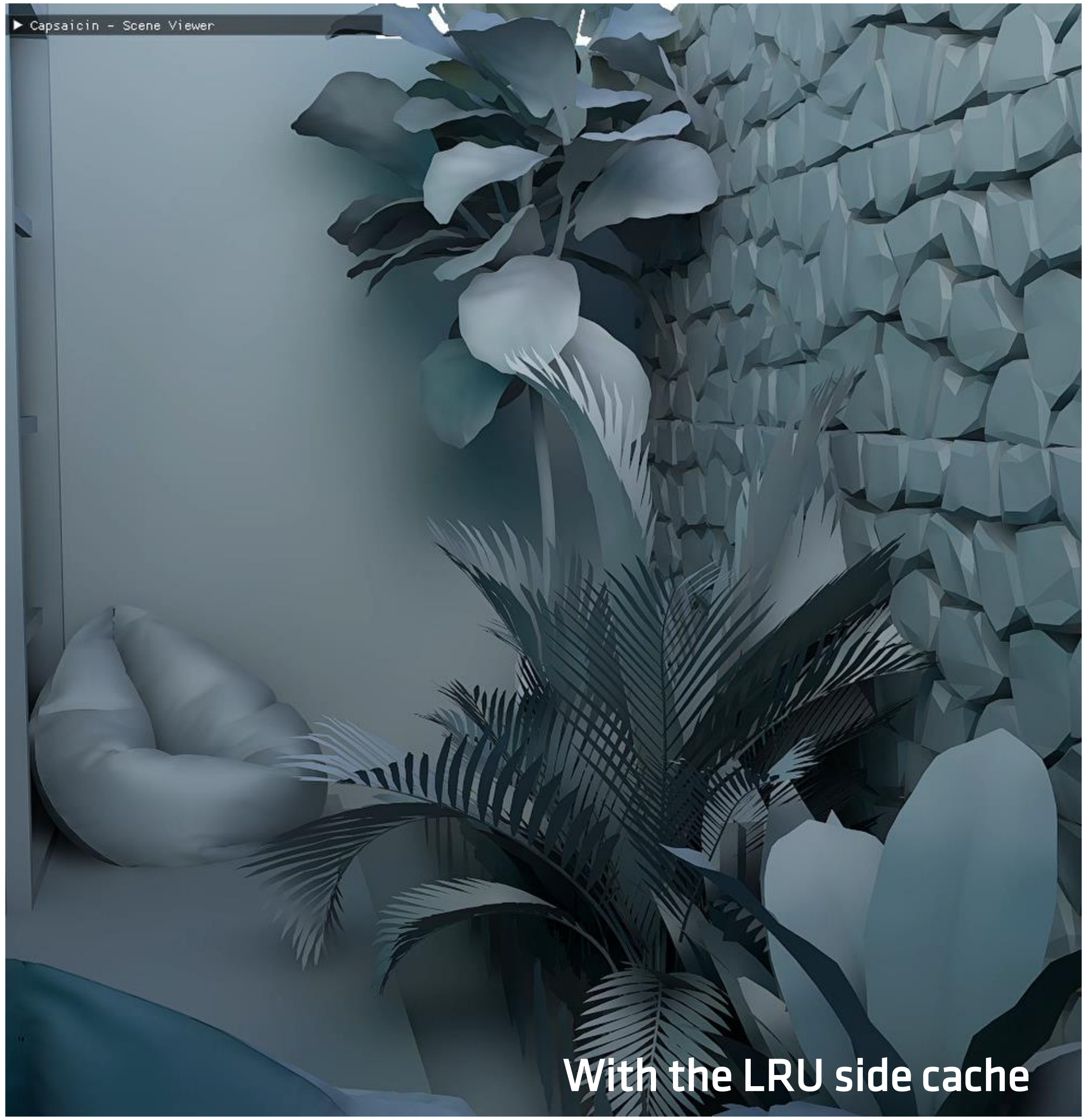


# PERSISTENT LRU SIDE CACHE

- Even after upscaling, we’re still very under sampled
- This is fine as we can interpolate between probes
- But thin geometry can cause temporal flickering

- Here, probe 1 is spawned and calculated…
- … but next frame, probe 2 cannot reuse from 1…
- … and finally, probe 3 cannot reuse from probe 2…
- … so temporal reuse & guiding are always failing ☹

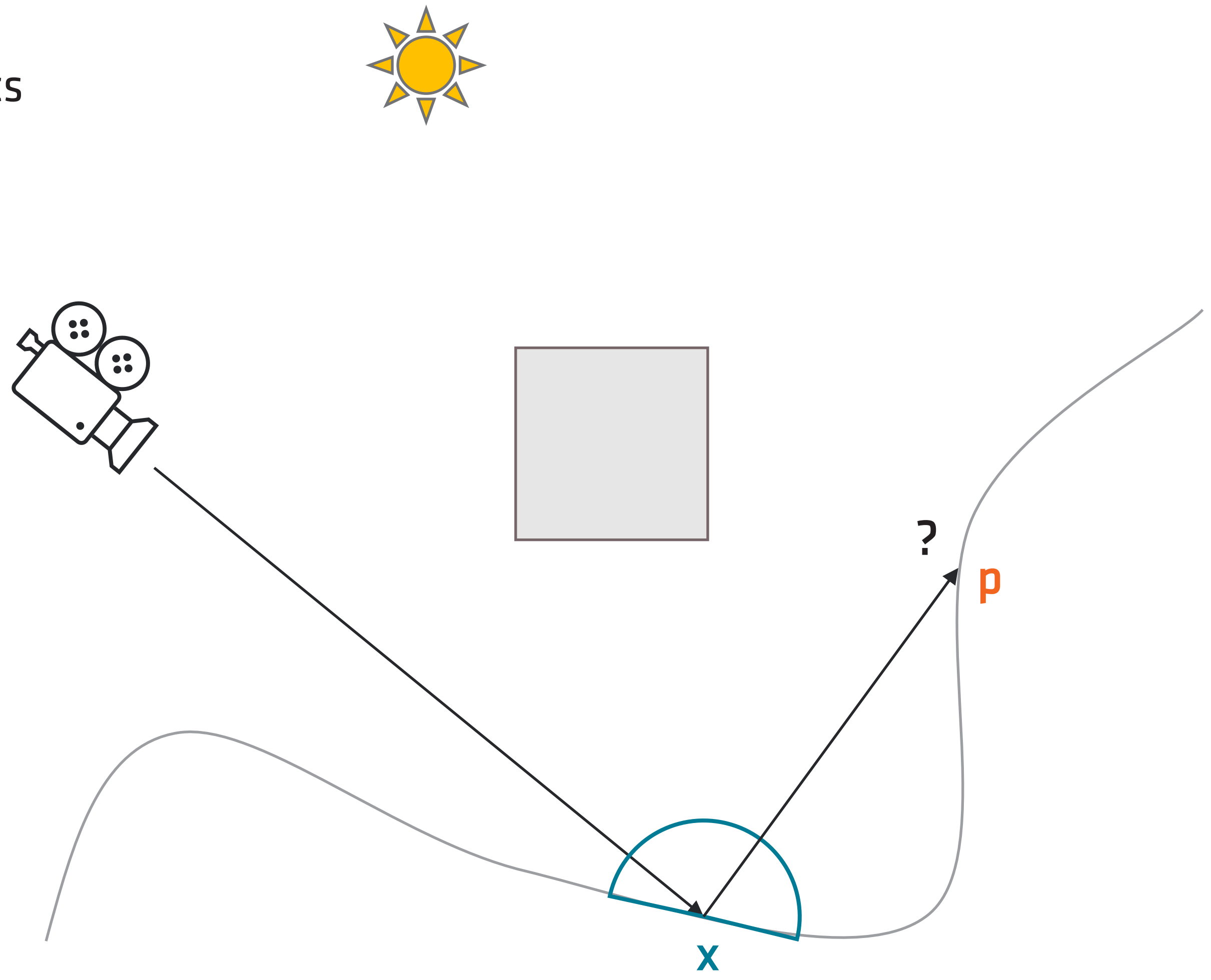
- So, cache “evicted” probes into a persistent queue
- Detected if new probe cannot reuse from center tile
- Can reuse from older probes many frames later ☺



## EVALUATING THE LIGHTING AT INDIRECT HITS

- Let’s get back to evaluating the lighting at p
  - We skipped this detail earlier on
  - This is where we calculate direct lighting on indirect hits
  - This is how we get our global illumination!

- We use a mix of 2 solutions
  - Temporal **radiance feedback**
  - Hash grid radiance caching (i.e., **hash cells**)



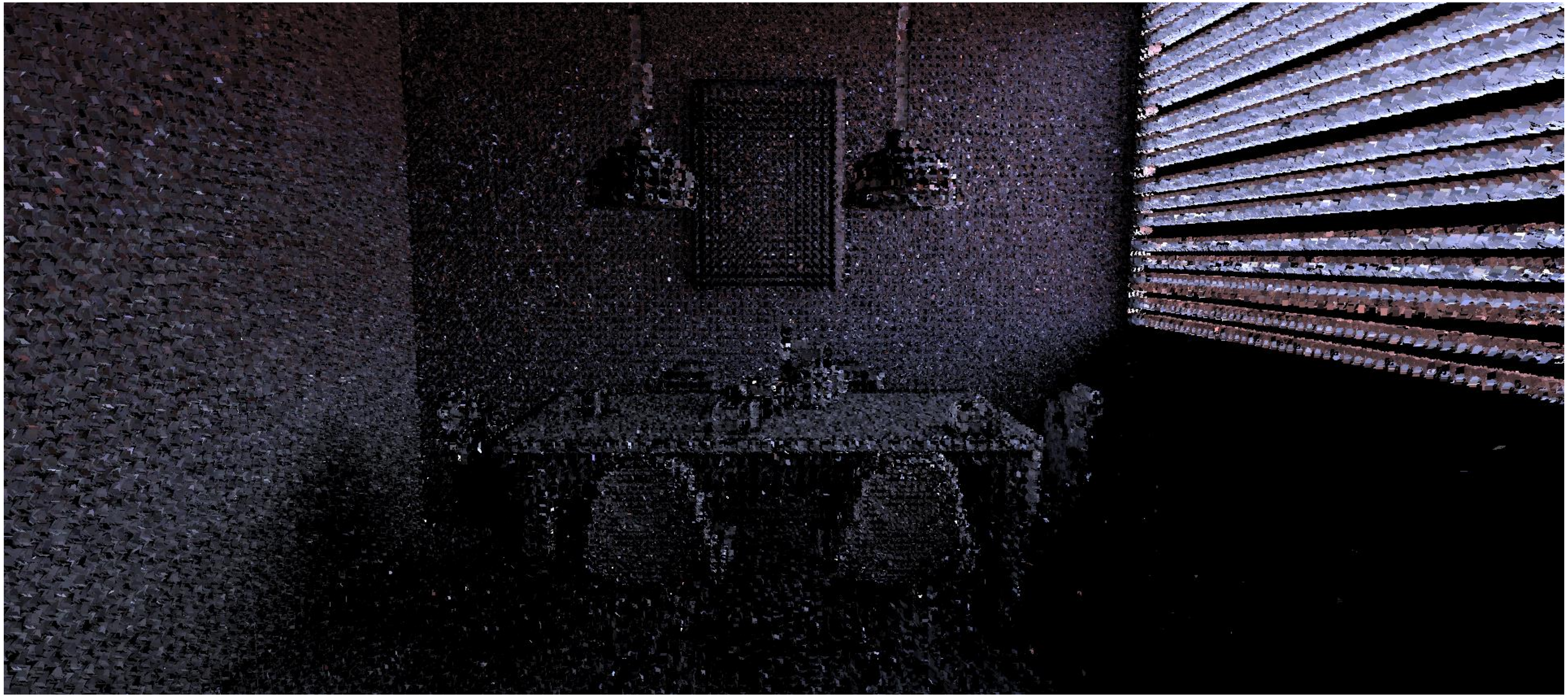
## TEMPORAL RADIANCE FEEDBACK

- Simply check whether p is visible in the last frame
  - Use the motion vectors to reproject temporally
  - Use previous depth and normal values to reject

- As previous frame’s lighting is direct+indirect, we get (limited) infinite multi-bounce for free
  - Great for eliminating many expensive shadow rays
  - But only works when lighting can be reprojected
  - Another mechanism is needed for off-screen hits…

without radiance feedback
with radiance feedback

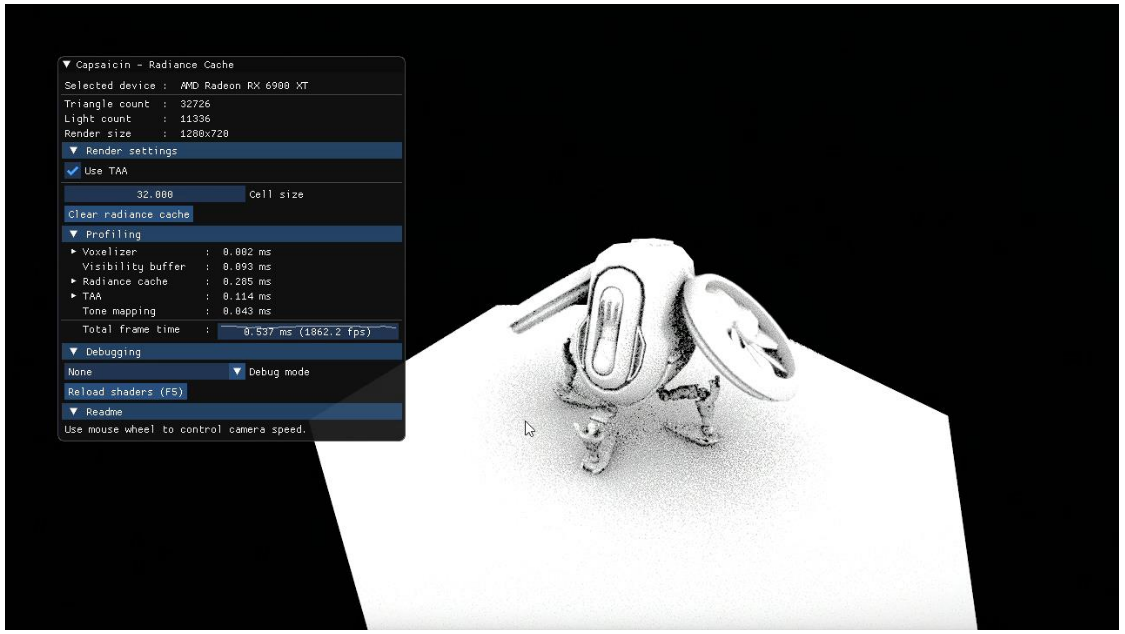
## HASH CELLS CACHE THE RADIANCE IN WORLD SPACE



## HASH GRID RADIANCE CACHE

- Based on spatial hashing [Binder et al. 2019]
  - Built on the fly (i.e., "build as you traverse")
  - No pre-processing needed, works on all geometry
  - Only allocate memory where information is needed

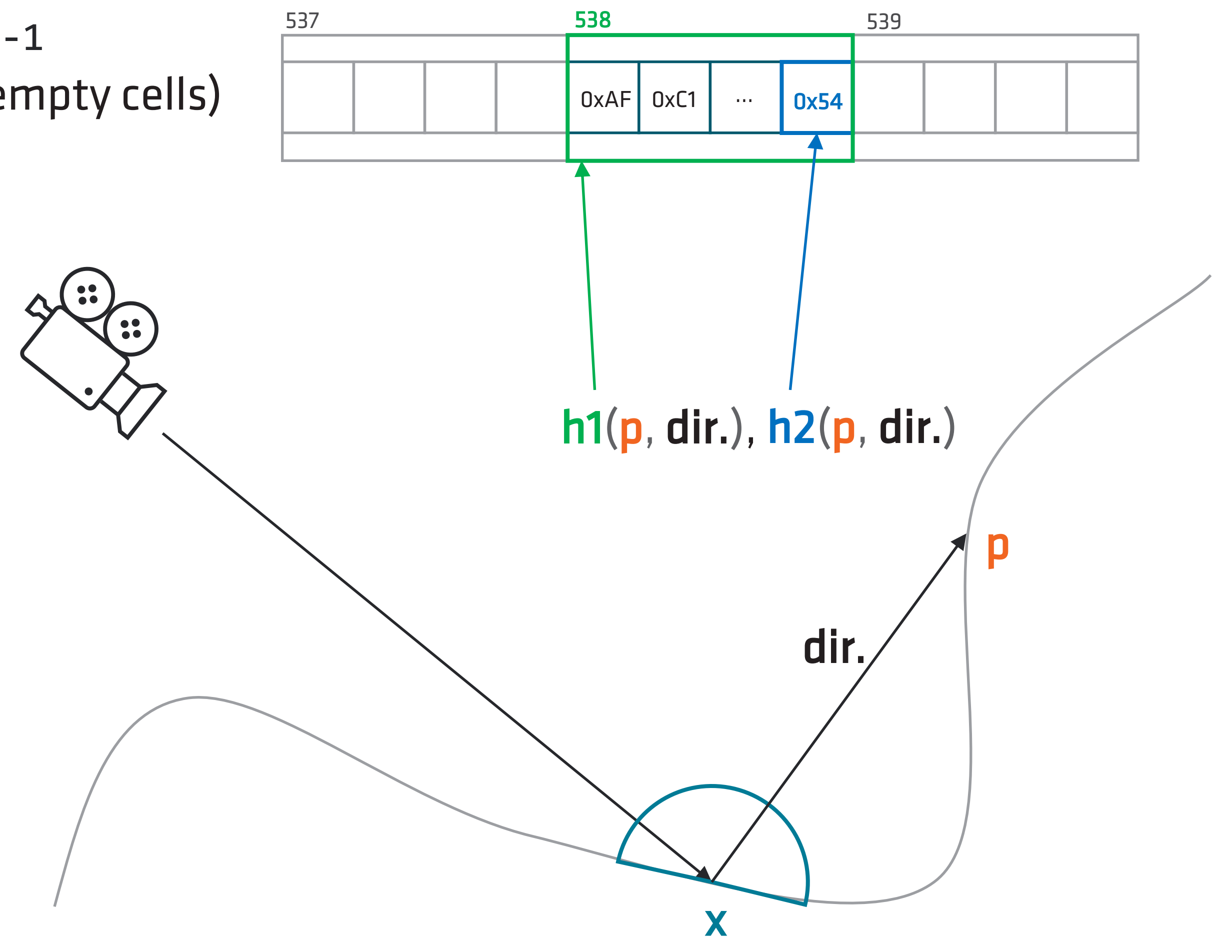
- Each cell is associated with a decay value
  - Reset every time the cell is "touched"…
  - … and decayed towards `0` otherwise
  - Once decay finishes, the cell is evicted



```
▼ Capsaicin - Radiance Cache
Selected device :  AMD Radeon RX 6900 XT
Triangle count  :  32726
Light count     :  11336
Render size     :  1280x720
▼ Render settings
[✓] Use TAA
32.000  Cell size
Clear radiance cache
▼ Profiling
 ▸ Voxelizer          :  0.002 ms
   Visibility buffer  :  0.093 ms
 ▸ Radiance cache     :  0.285 ms
 ▸ TAA                :  0.114 ms
   Tone mapping       :  0.043 ms
   Total frame time   :  0.537 ms (1862.2 fps)
▼ Debugging
None ▼  Debug mode
Reload shaders (F5)
▼ Readme
Use mouse wheel to control camera speed.
```

## SPATIAL HASHING

- Hash the vertex descriptor twice
  - Descriptor is quantized position p and direction **dir.**
  - 1st hash (i.e., h1) returns value between `0` and `capacity-1`
  - 2nd hash (i.e., h2) returns anything but `0` (reserved for empty cells)
- Use the first hash to locate the **bucket**
  - Simple linear indexing into array
  - Every bucket contains 16 cells
- Use the second hash to locate the **cell**
  - Use `InterlockedCompareExchange()` to find/insert
  - Resolved cell stores our locally filtered radiance
- Accumulation is performed using 4x `InterlockedAdd()`
  - Performed into some pre-cleared scratch memory
  - Later resolved to a half-precision `float4`



## ADAPTIVE FILTERING HEURISTICS

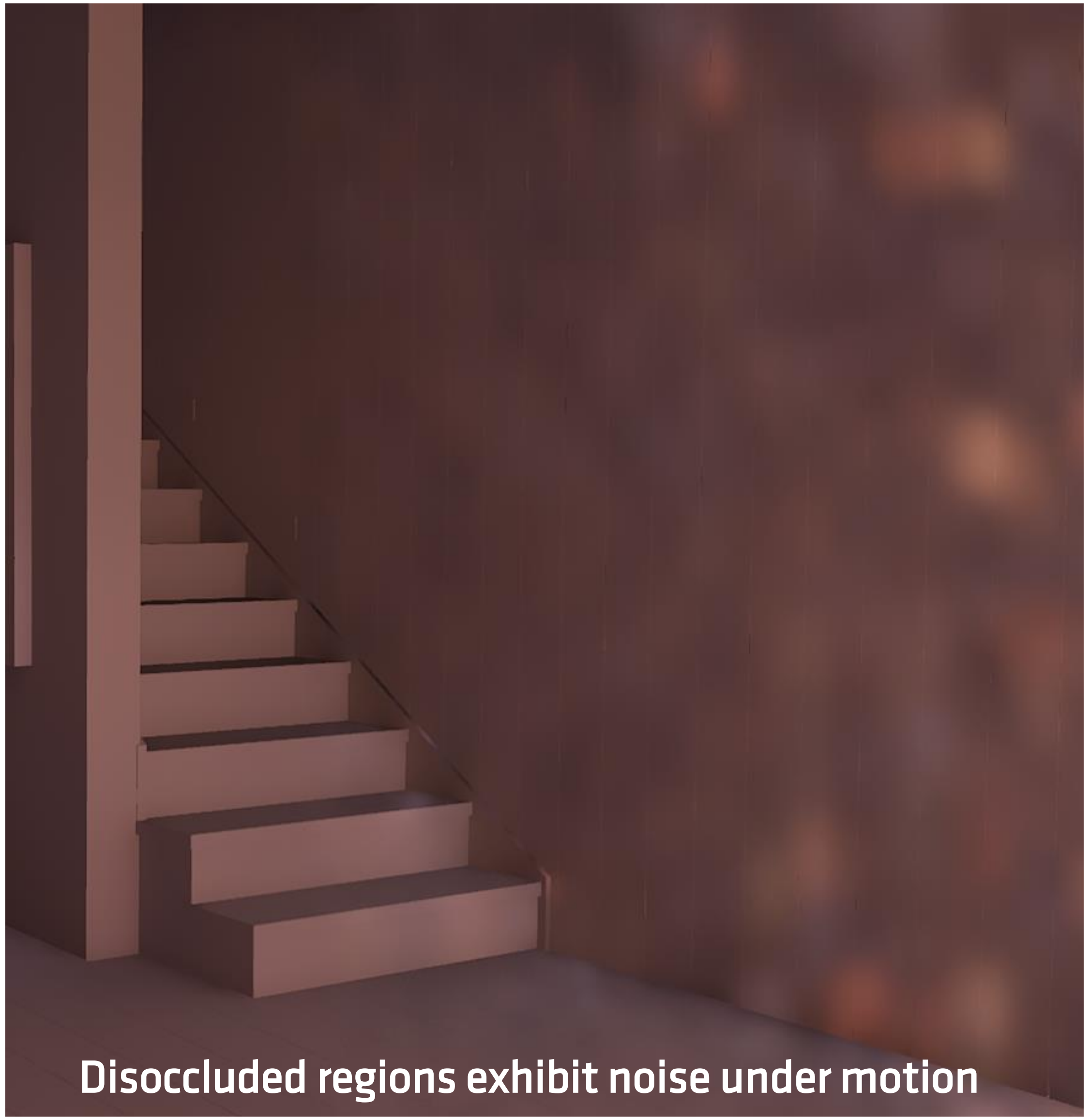
- At a distance, geometrical detail density increases
  - Possibly far beyond sampling & memory budget!
  - Need a way to keep memory use under control…
- So, create radiance LODs (Level Of Details)
  - Simply adapt the quantization amount (i.e., **cell size**)
  - Target a roughly constant screen size after projection
- We apply a similar approach to our screen probes
  - Relax the radiance reuse heuristics at a distance
  - Trade off additional bias for temporal stability

gracefully degraded further out

## FILTERING IN HASH SPACE

- The size of a cell describes the amount of radiance filtering
  - Large cell size → less variance, more bias
  - Small cell size → more variance, less bias

- Ideally, we'd want to be able to pick the cell size adaptively
  - Especially since filtering in hash space is very costly
  - A simple 3x3x3 kernel requires **54** hash operations…
  - … and up to ~**400** reads ☹

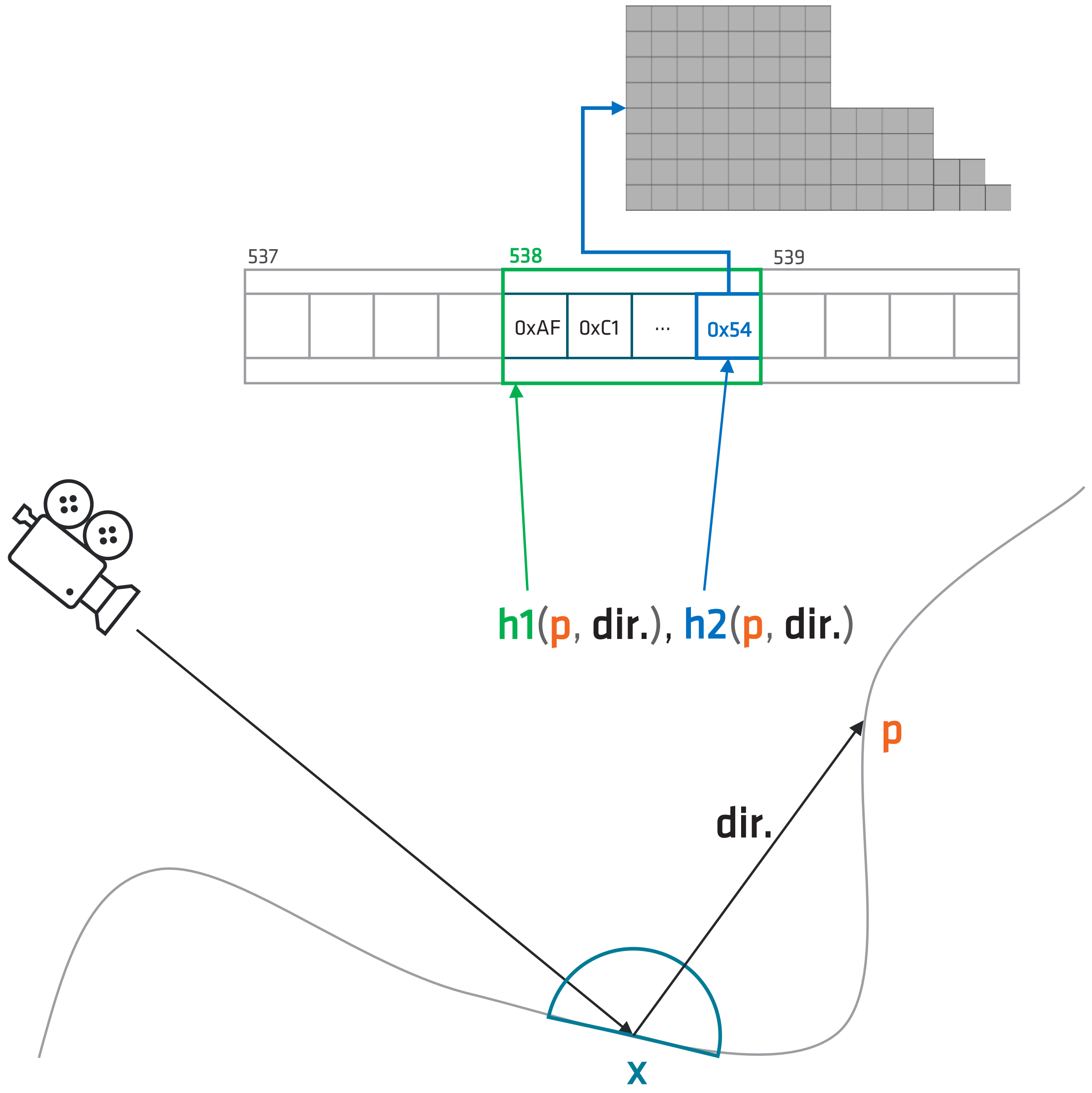


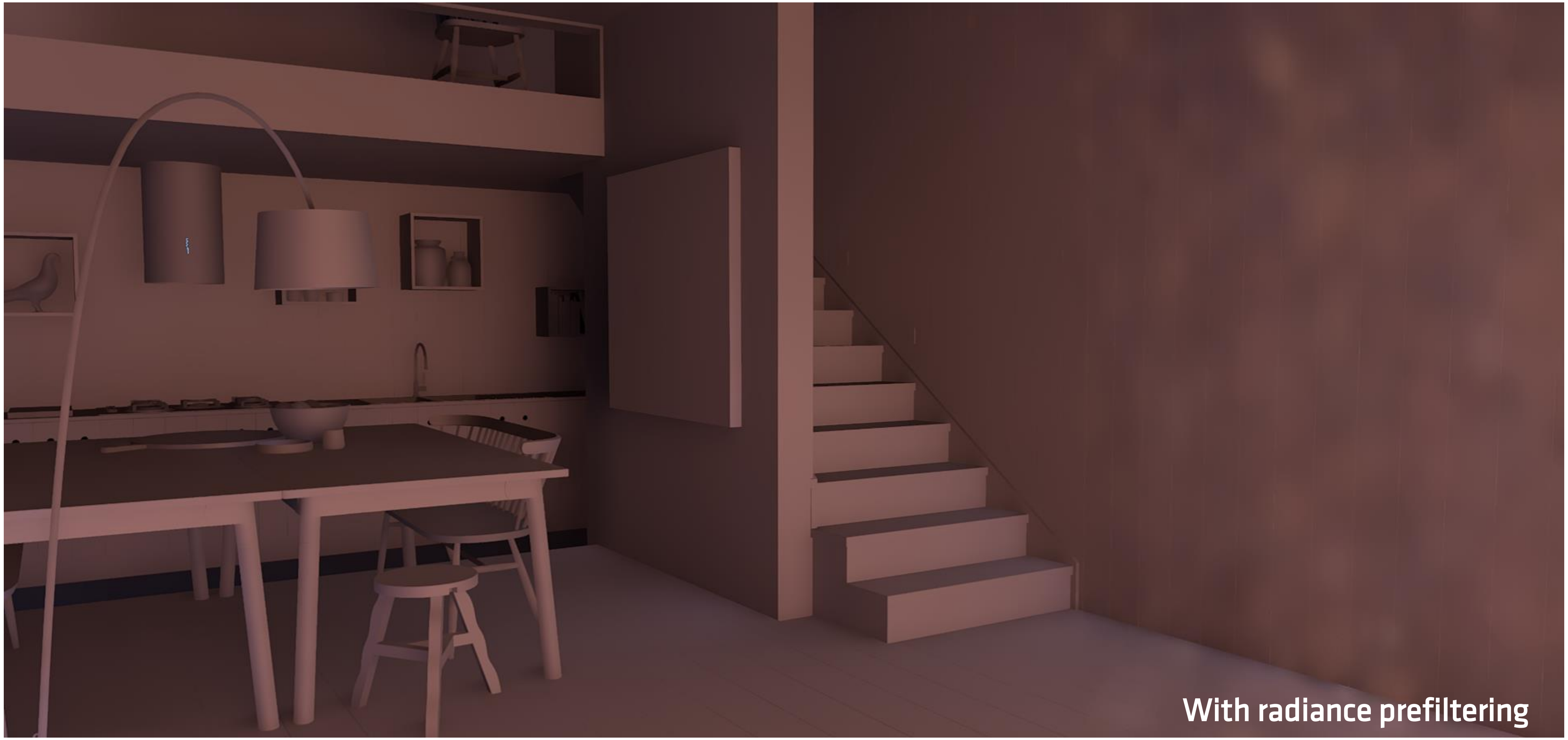
# RADIANCE PREFILTERING

- So, we make the **hash grid** into a **two-level structure**
  - **Buckets** now point to **2D tiles**…
  - … of **8x8 cells** of radiance data

- Tile orientation is determined based on ray direction
  - Project onto the **XY**, **XZ**, or **YZ** plane
  - Drop the largest component from **dir.**

- Tiles can be prefiltered similarly to mipmapping
  - Prefiltering is fast and happens entirely in **LDS**
  - Can pick the right mip for a given noise level ☺



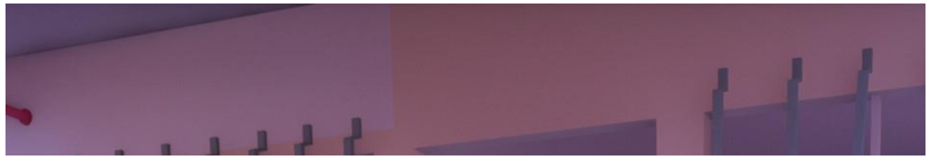

With radiance prefiltering

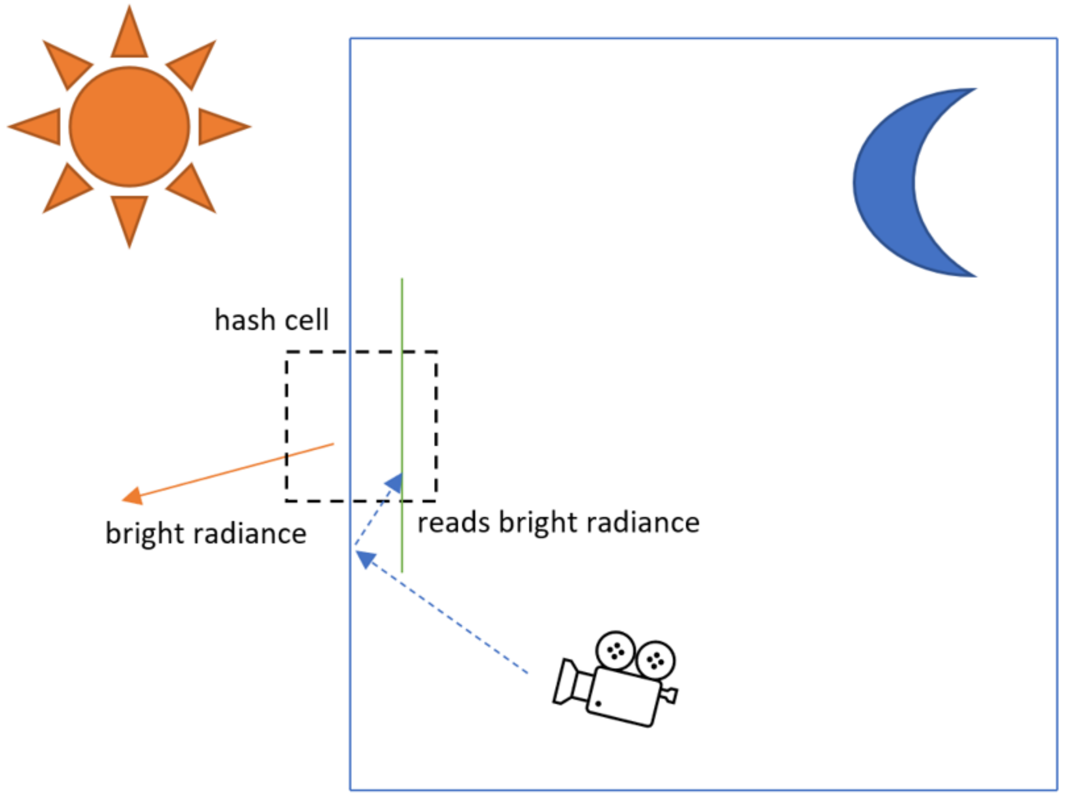
# LIGHT LEAKING

- Some scene setups may introduce light leaks
  - Here, the bounce ray hits the **green** inner wall…
  - … and yields same position hash than the outer wall
  - Direction is parallel enough to **bright radiance** event
  - So, direction hash also returns the same value!

- Both events use the same cell and share radiance…
- … resulting in a light leak ☹

- We found this mostly happens when the ray length is less than the cell size
  - So, extend the descriptor with `rayLength < cellSize`
  - Splits the cell in 2, which helps separate the events
  - Solves our light leak ☺





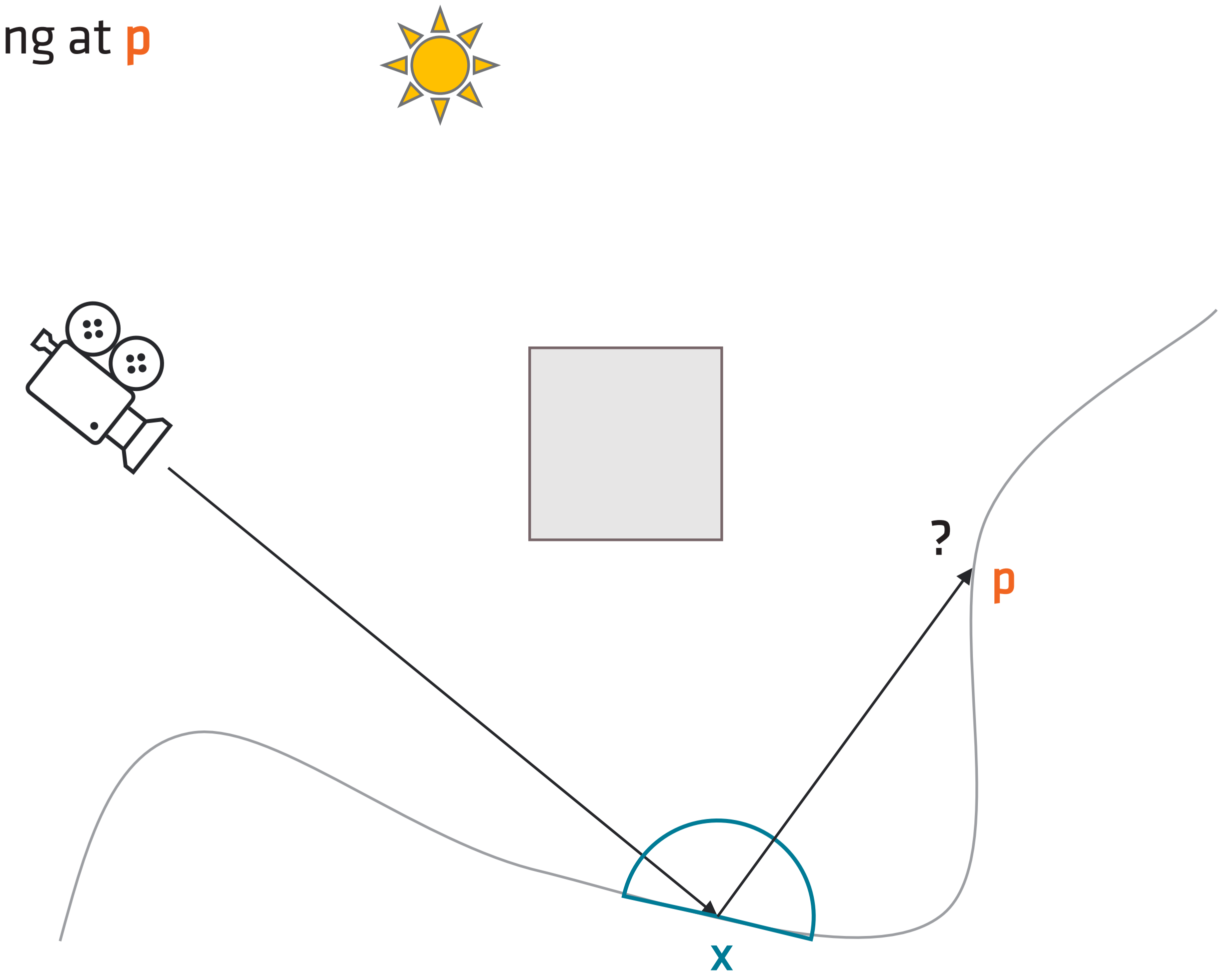
With light leak heuristic

# EVALUATING THE LIGHTING AT INDIRECT HITS (AGAIN)

- We've seen how we were caching the radiance in world space…
- … but we still haven't seen how to evaluate the lighting at p

- Multiple solutions are possible
  - Per-light shadow maps
  - Ray tracing the lights

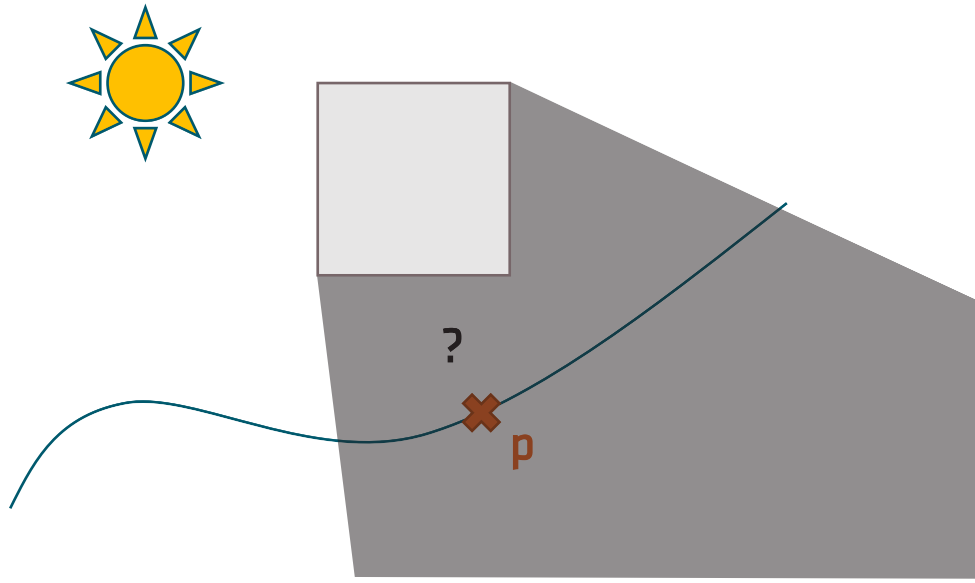
- Our implementation focuses on ray tracing
  - More forward looking (many-light scenarios, etc.)
  - But shadow maps can be used!
  - Simply make 2 light lists…
  - … one for **SMs**, one for **RT**



## LIGHT SAMPLING

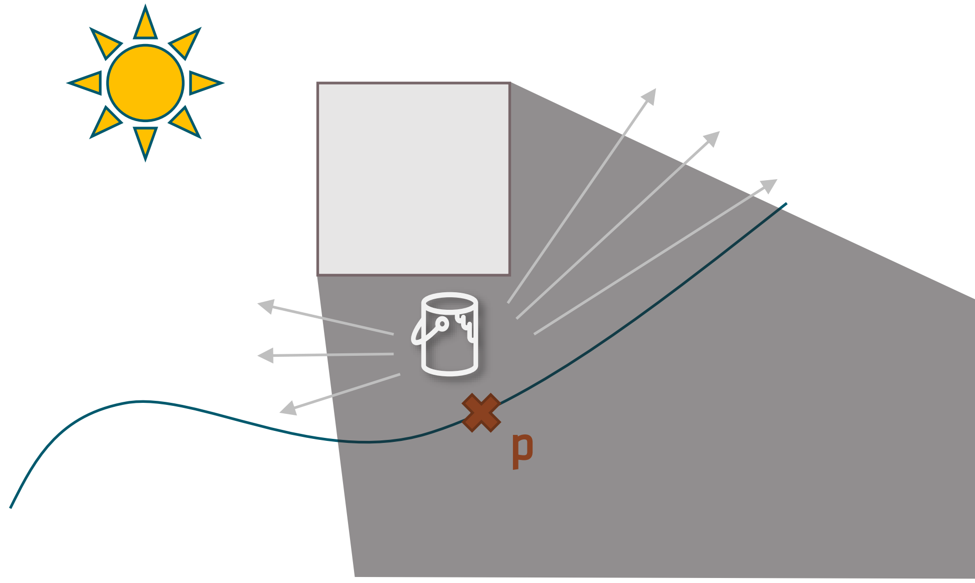
- Ray tracing the lighting from p can produce noise
  - Many shadows rays are wasted as they do not hit the light
  - This has a strong impact on the variance of the **hash cells**…
  - … as well as the noise in the **screen probes**…
  - … and therefore, the final on-screen noise!

- We could solve this by throwing in more rays/samples
  - But we'd like to maintain the ray count low and constant
  - Need to be smarter about our sampling…



## RESERVOIR-BASED RESAMPLING

- Reservoir-based SpatioTemporal Importance Resampling (ReSTIR) [Bitterli et al. 2020]
    - Very promising technique for real-time sampling of many ray traced lights
    - Generate a reservoir storing the best candidate for a shading point…
    - … and share it with your neighbors through resampling
- But ReSTIR’s reuse originally happens in image space
    - Neighbor path vertices aren’t necessarily close in image space
    - Screen-space reservoir reuse does not work ☹
- So, perform the reuse in world space instead [Boissé 2021]
    - Create a list of reservoirs for every overlapping vertex in a cell
    - Use spatial hashing again to maintain the data structure ☺



## LIGHT GRID STRUCTURE

- ReSTIR provides great variance reduction
  - But resampling is only as good as its initial state…
  - … so, work on improving that initial state ☺
- Build a **light grid** to help pick the best candidates
  - Store a list of the most important lights per cell
  - Weight based on illumination over the cell's volume
  - Cull lights when contribution falls below threshold
- Select lights from list by looking up the grid cell
  - Pick a random subset (8 in our implementation)
  - Generate the reservoir using resampling…
  - … and append to the world-space cache

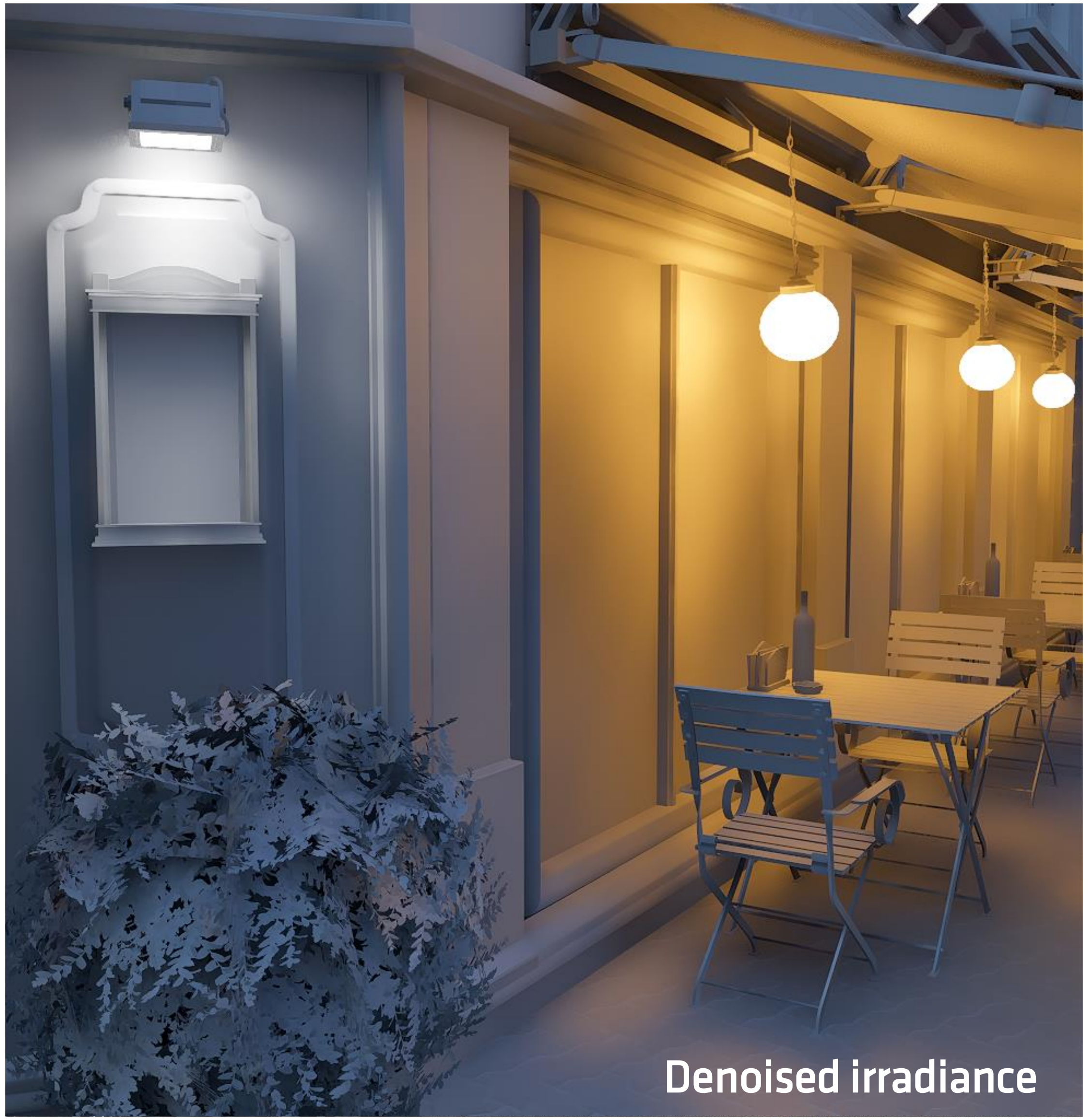
reservoir resampling

## PER-PIXEL INTERPOLATION & DENOISING

## IRRADIANCE INTERPOLATION

- Locate 4 neighbor probes for every pixel
  - Depth-based check for rejecting invalid probes
  - Hide artefacts using blue noise [Heitz et al. 2019]

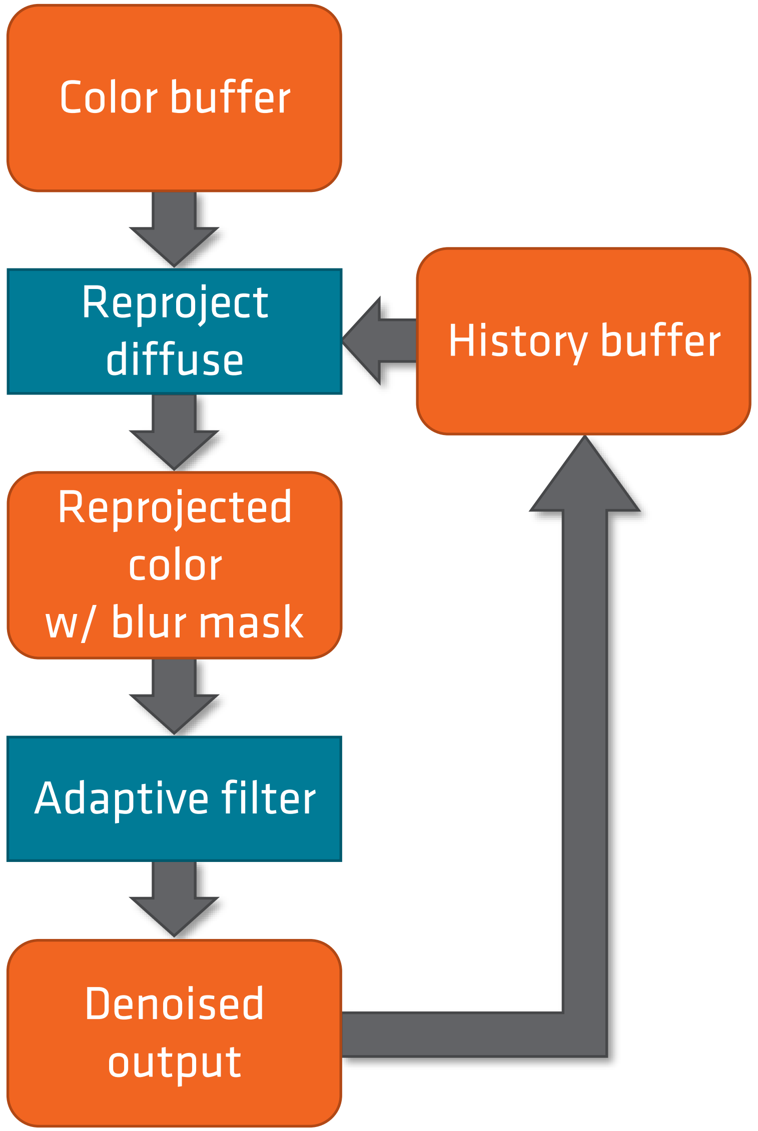
- Noise is moderate thanks to the caching system…
- … so, no heavy denoising is required ☺



# DENOISING PIPELINE

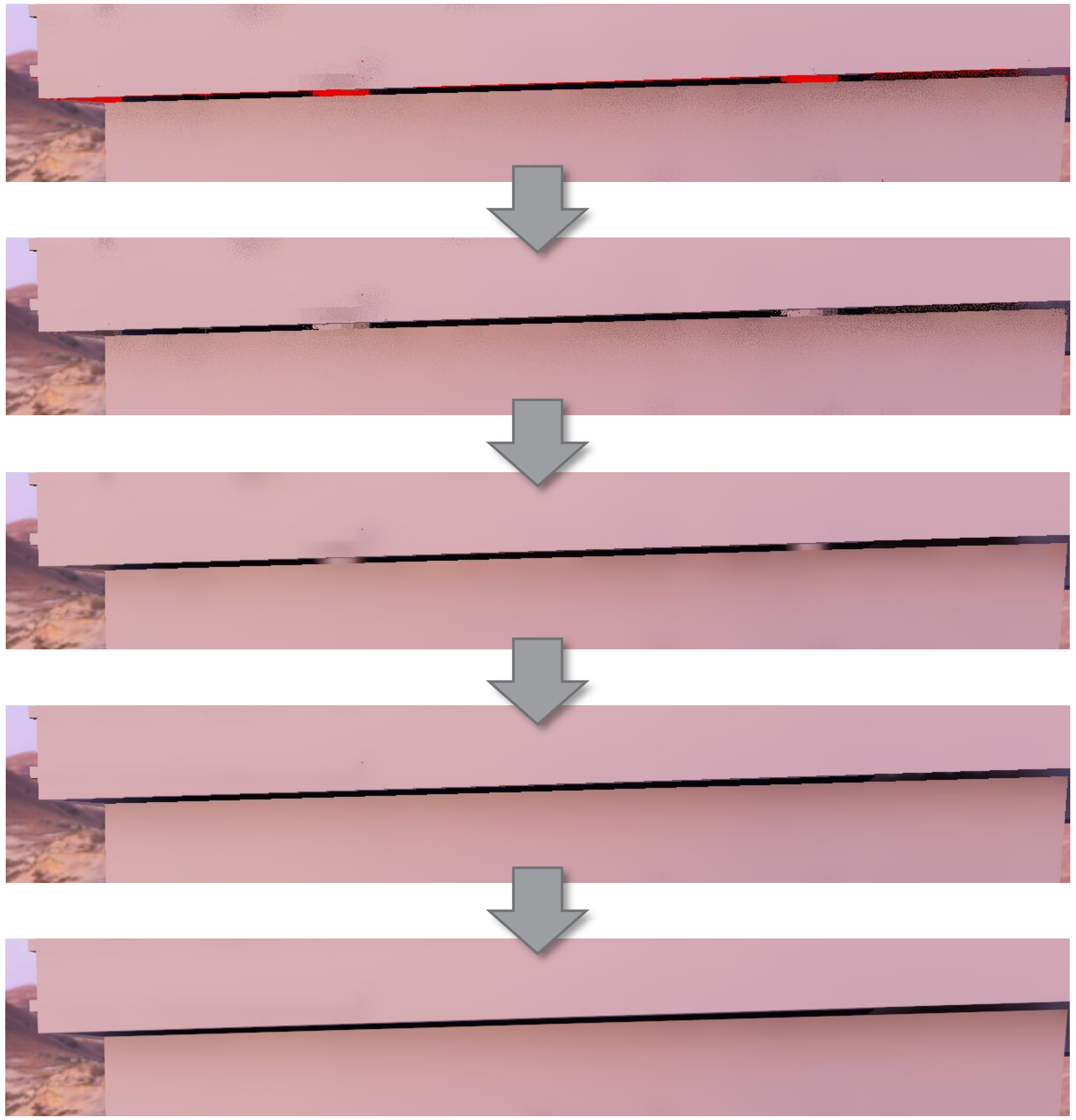
- Simple denoiser (i.e., more of a “cleanup” pass)
  - Tries to rely solely on temporal reprojection
  - Yields best quality with lowest bias/blur
  - But reprojection can fail at disocclusions…
  - … so, output a **blur mask** to adaptive filter

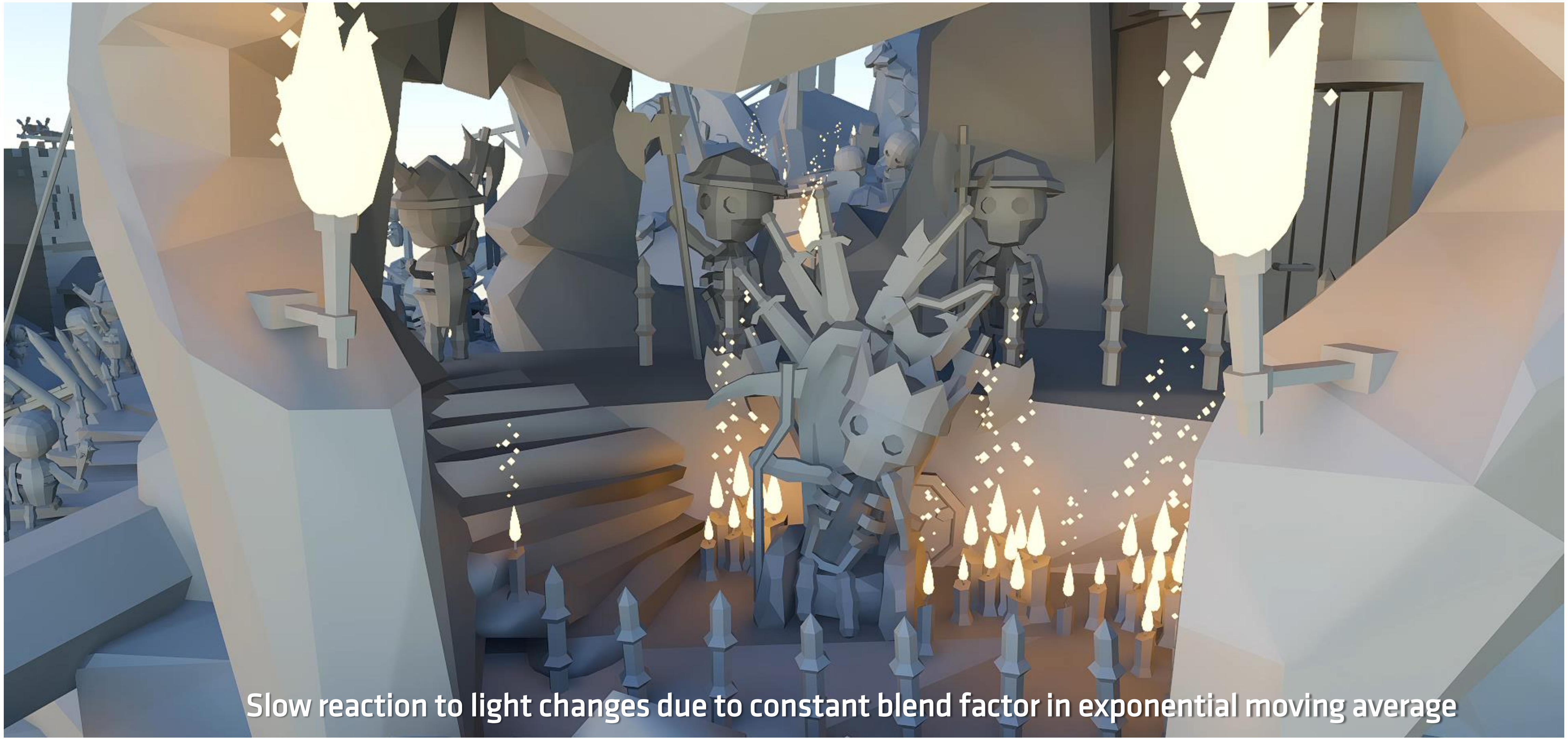
- In practice, only blur “new pixels” for ~8 frames…
- … using an ever-decreasing radius (i.e., **blur mask**)
- Rely only on temporal filtering for the rest

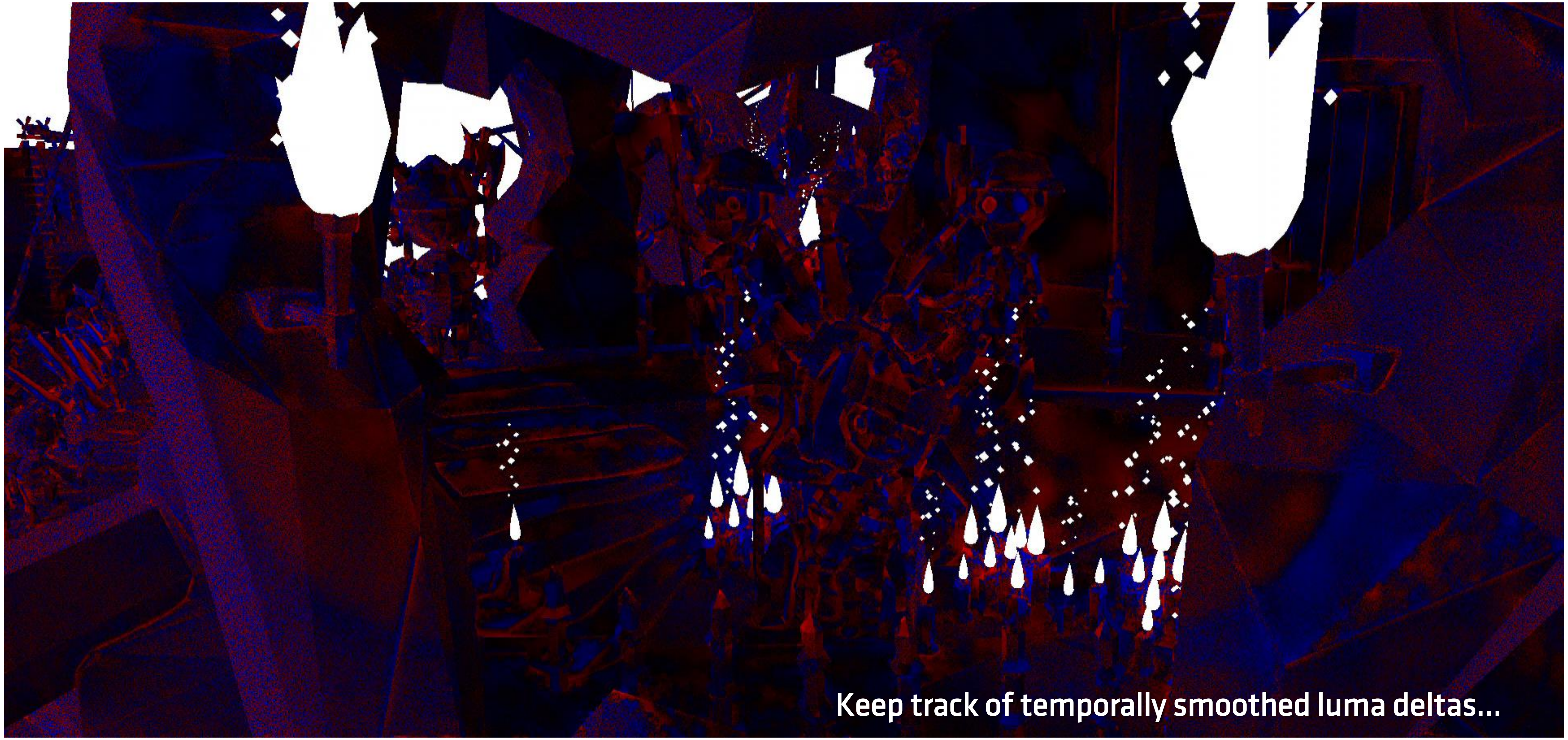


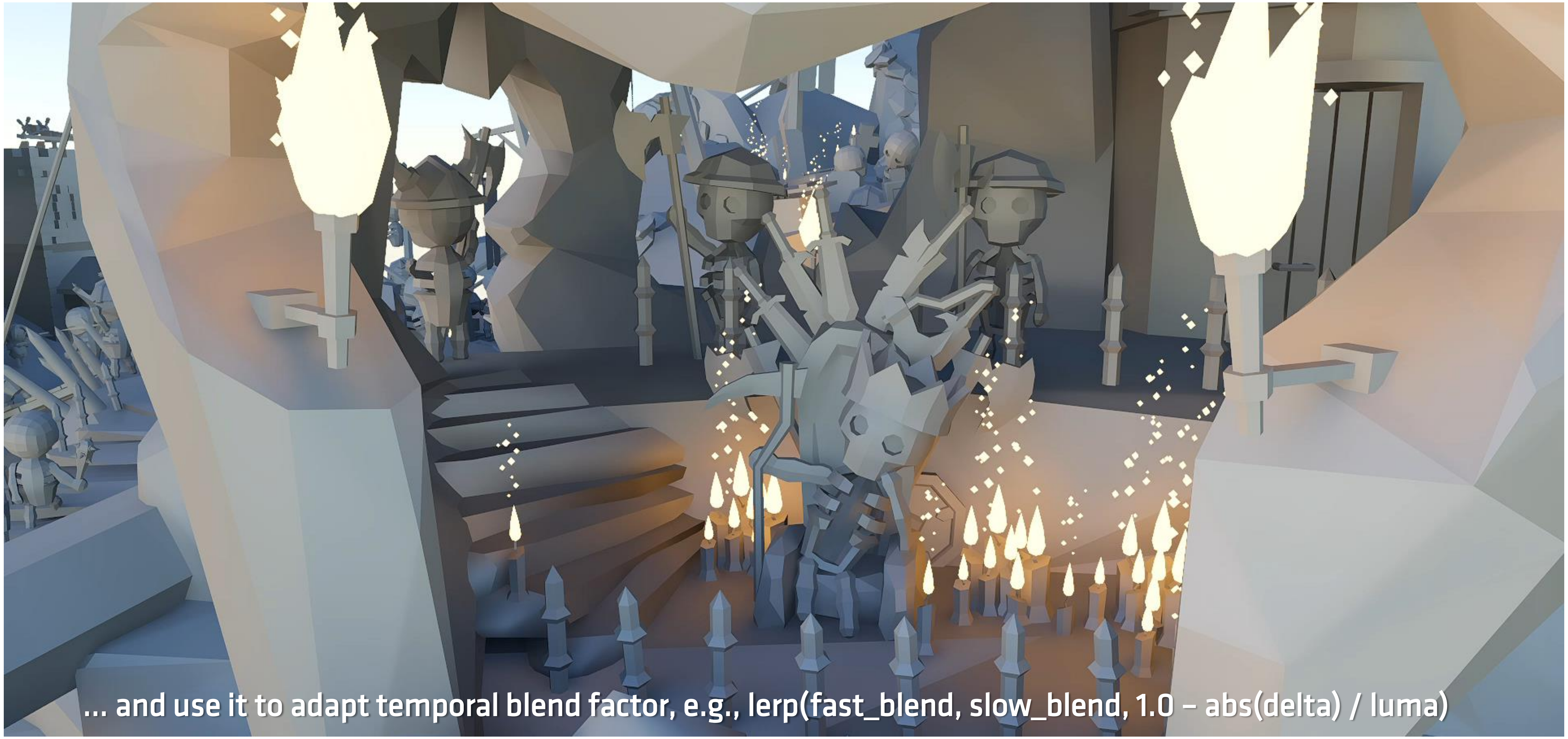
## INTERPOLATION FAILURE

- Due to depth-based rejection, interpolation can fail
  - All 4 probes get associated a weight of `0`
  - We don't have anything to interpolate ☹
- In this case, we simply switch back the weights to `1`
  - Effectively doing a simple average of the 4 probes
  - Which introduces some undesirable light leaking
- Fixed by flagging these pixels in the alpha channel
  - Essentially sending a hint to the denoiser
  - Denoiser will try to avoid using this sample




Slow reaction to light changes due to constant blend factor in exponential moving average


Keep track of temporally smoothed luma deltas…



… and use it to adapt temporal blend factor, e.g., lerp(fast_blend, slow_blend, 1.0 – abs(delta) / luma)

## DYNAMIC GEOMETRY

- Moving objects “work”
  - Screen probes use motion vectors for reprojection
  - Hash cells can adapt to the moving content

- Not heavily tested still, more research is needed

Capsaicin - Scene Viewer

## MISCELLANEOUS

- More elements go into the irradiance interpolation
  - Not covered in depth here for time reasons
  - See the paper for more details [Boissé et al. 2022]

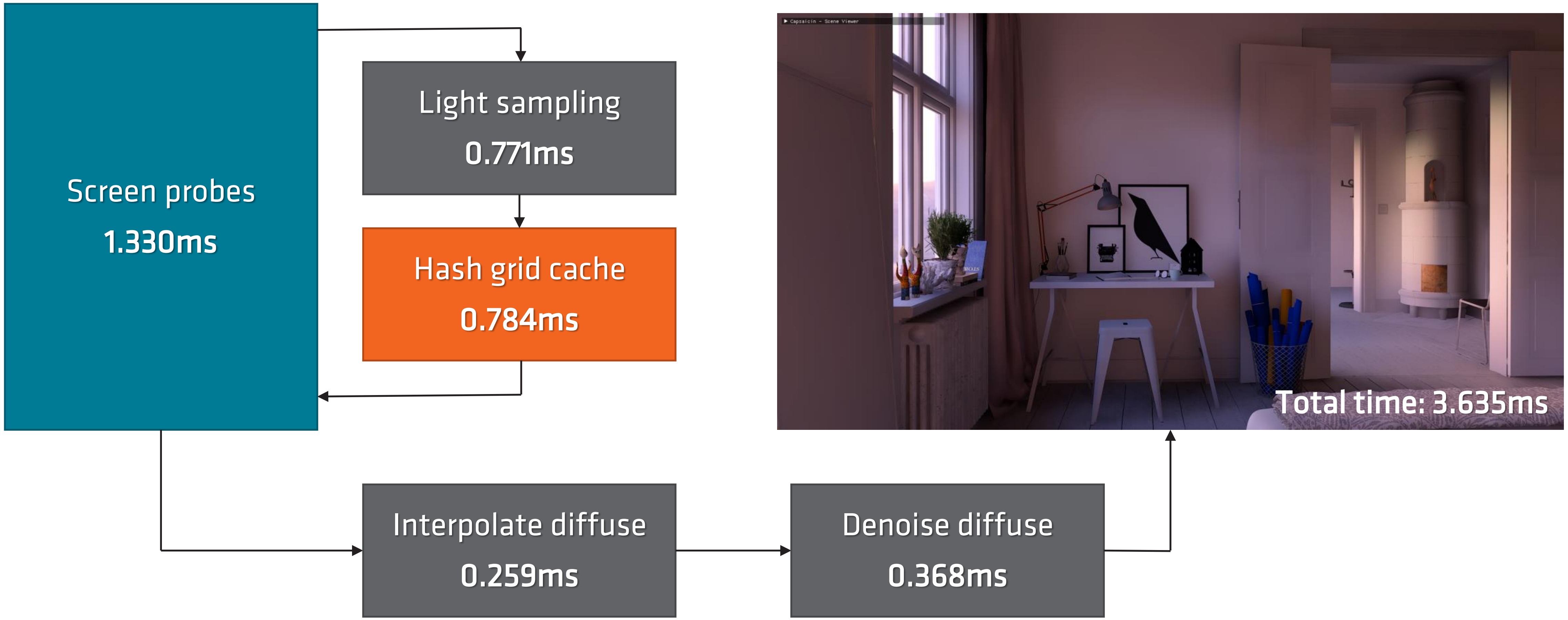
- Turn the probes into SHs [Ramamoorthi et al. 2001]
  - Update probe reprojection pass to reproject SHs too
  - Use SHs for noise-free irradiance estimate

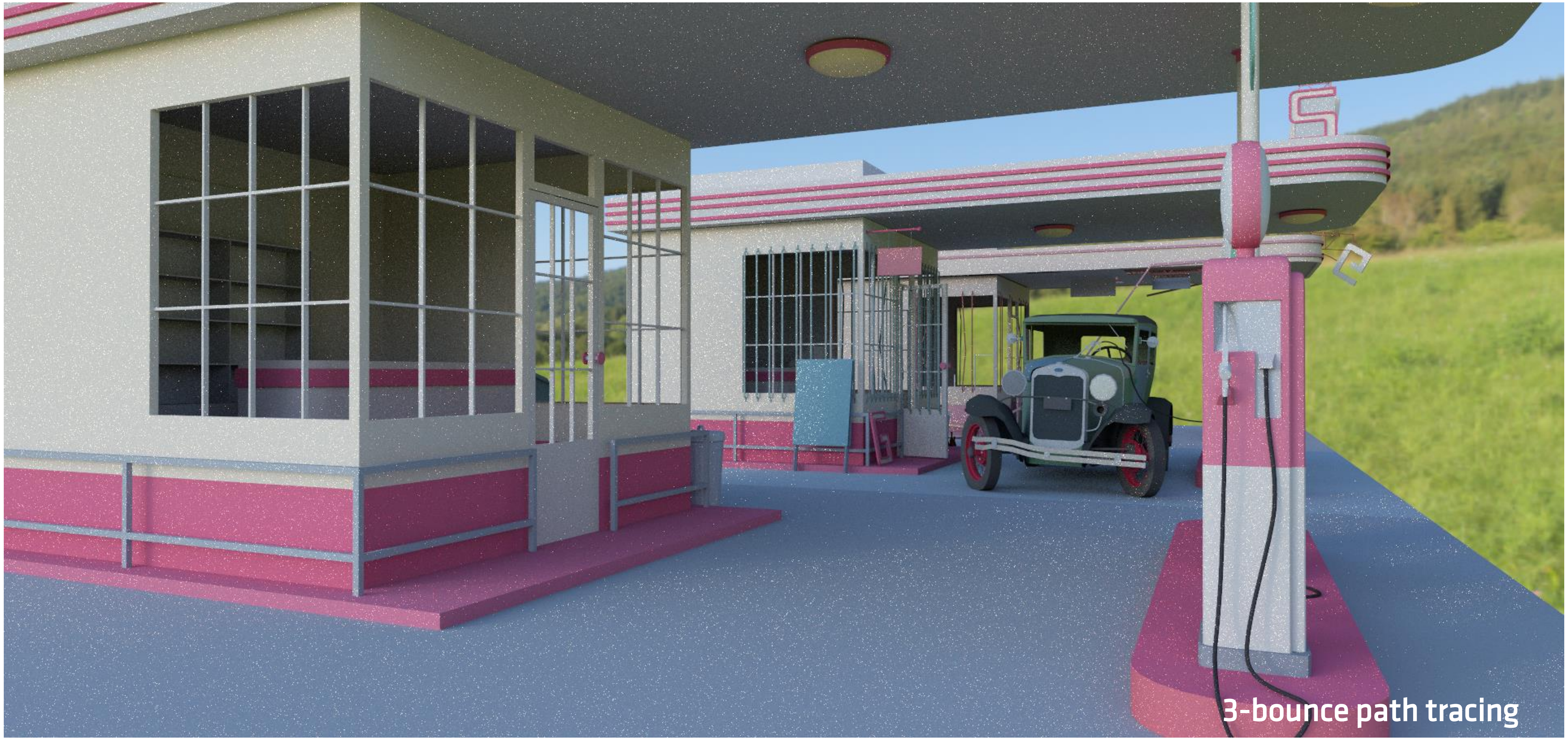
- Use screen-space GI to recover small-scale details
  - Probes can't capture details smaller than ~20 pixels
  - Implemented horizon-based GI [Jimenez et al. 2016]

w/ horizon-based GI

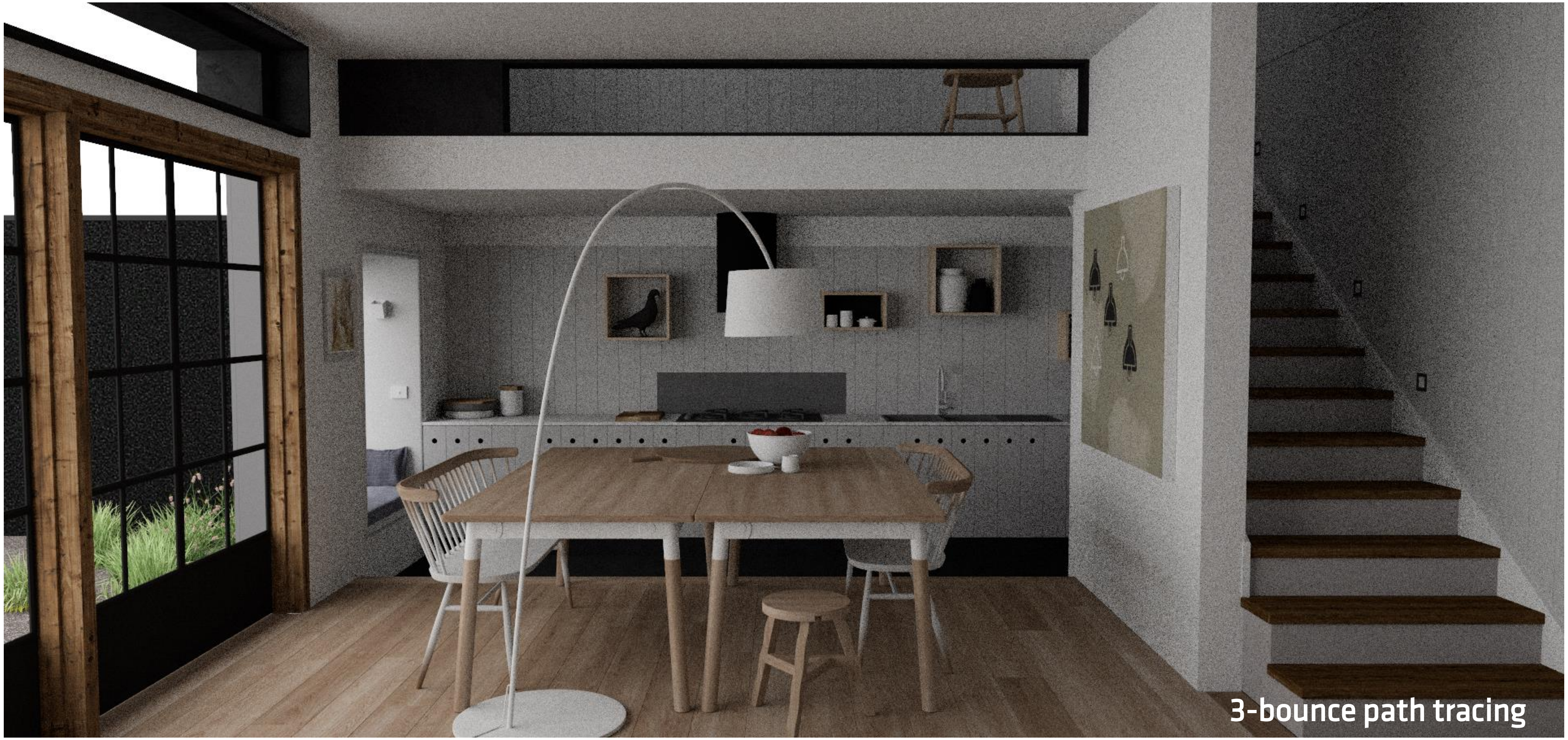
## PERFORMANCE RESULTS

- Measurements performed at 1080p on an AMD Radeon™ RX 7900 XTX




3-bounce path tracing

GI-1.0


3-bounce path tracing

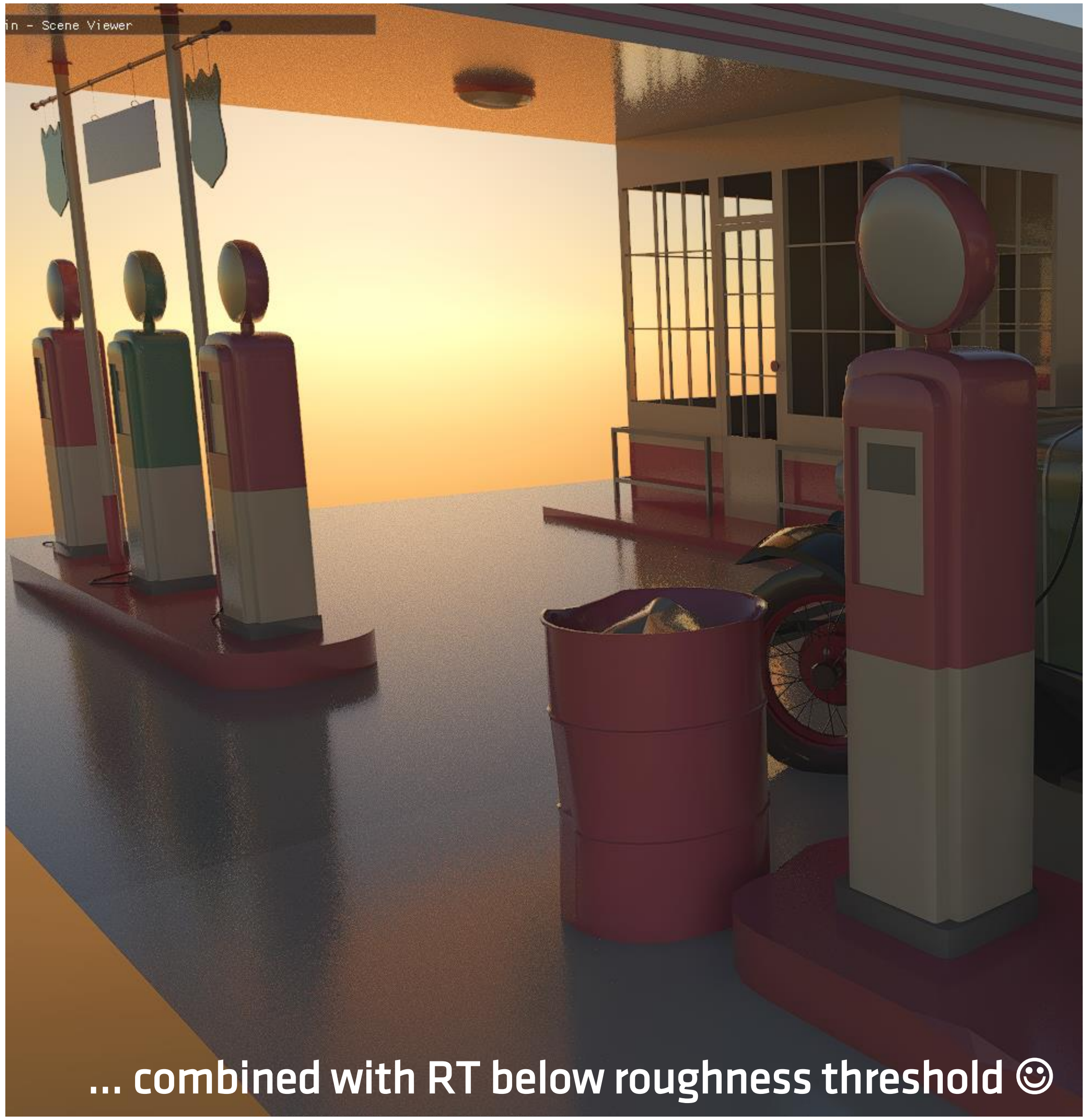
GI-1.0

3-bounce path tracing

GI-1.0

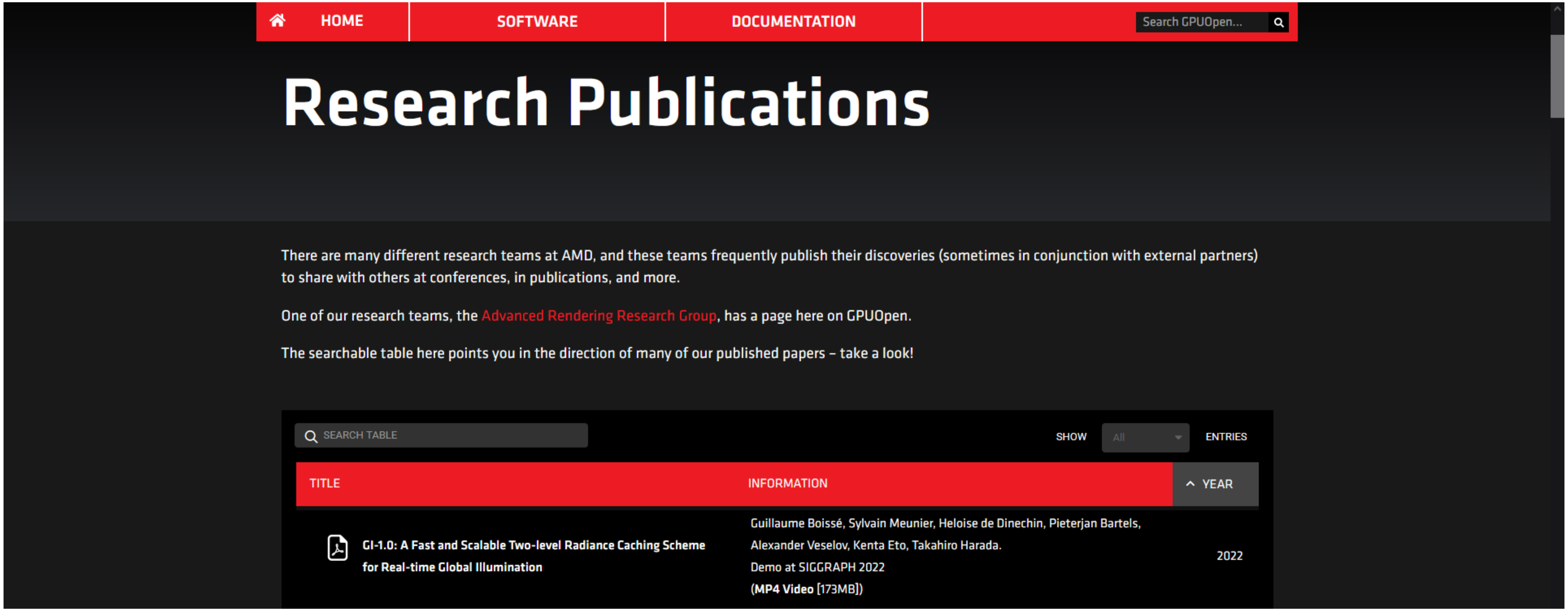
## CONCLUSION & FUTURE WORK

- Presented a complete global illumination pipeline
  - Diffuse indirect lighting for all lights (delta, sky, etc.)
  - Diffuse direct lighting for some lights (emissive, sky)
- Interior scenes are too dark due to limited multi-bounce support
  - Trace continuation rays from the **hash cells**
  - Get reliable multi-bounce everywhere
- Working on adding support for glossy reflections
  - Use **screen probes** for mid to high roughness
  - Use reduced rate ray tracing for low roughness
- Test & tweak on scenes with animation, foliage, etc.



## FIND OUR PAPER ON GPUOPEN

https://gpuopen.com/learn/publications/



Research Publications

There are many different research teams at AMD, and these teams frequently publish their discoveries (sometimes in conjunction with external partners) to share with others at conferences, in publications, and more.

One of our research teams, the Advanced Rendering Research Group, has a page here on GPUOpen.

The searchable table here points you in the direction of many of our published papers – take a look!

| TITLE | INFORMATION | YEAR |
|---|---|---|
| GI-1.0: A Fast and Scalable Two-level Radiance Caching Scheme for Real-time Global Illumination | Guillaume Boissé, Sylvain Meunier, Heloise de Dinechin, Pieterjan Bartels, Alexander Veselov, Kenta Eto, Takahiro Harada. Demo at SIGGRAPH 2022 (MP4 Video [173MB]) | 2022 |

## REFERENCES

| | |
|---|---|
| Binder et al. 2019 | "Massively Parallel Path Space Filtering" |
| Bitterli et al. 2020 | "Spatiotemporal Reservoir Resampling for Real-Time Ray Tracing with Dynamic Direct Lighting" |
| Boissé 2021 | "World-Space Spatiotemporal Reservoir Reuse for Ray-Traced Global Illumination" |
| Boissé et al. 2022 | "GI-1.0: A Fast Scalable Two-Level Radiance Caching Scheme for Real-Time Global Illumination" |
| Brinck et al. 2021 | "Global Illumination Based on Surfels" |
| Cigolle et al. 2014 | "A Survey of Efficient Representations for Independent Unit Vectors" |
| Heitz et al. 2019 | "A Low-Discrepancy Sampler that Distributes Monte Carlo Errors as a Blue Noise in Screen Space" |
| Jimenez et al. 2016 | "Practical Real-Time Strategies for Accurate Indirect Occlusion" |
| Karis 2014 | "High-Quality Temporal Supersampling" |
| Majercik et al. 2019 | "Dynamic Diffuse Global Illumination with Ray-Traced Irradiance Fields" |
| Ouyang et al. 2021 | "ReSTIR GI: Path Resampling for Real-Time Path Tracing" |
| Ramamoorthi et al. 2001 | "An Efficient Representation for Irradiance Environment Maps" |
| Wright 2021 | "Radiance Caching for Real-Time Global Illumination" |

# THANK YOU!

## DISCLAIMER

AMD GPUOpen

AMD together we advance_

AMD EPYC

AMD RYZEN

AMD RADEON