Hello everyone, and welcome to the presentation about Stochastic Tile-Based lighting!

Introduction

SIGGRAPH 2025
Vancouver+ 10-14 August

Jarkko Lempiäinen
Principal Graphics Engineer @ hypehype

@JarkkoPFC
linkedin.com/in/jarkkol

I'm Jarkko Lempiäinen and I work as a Principal Graphics Engineer at HypeHype. Over the past 25 years, I have focused mainly on developing AAA rendering technology at companies like Ubisoft, Unity and Treyarch. I joined HypeHype about a year ago, where we are building a User-Generated Content platform that runs on everything from $99 budget phones to high-end PCs.

Currently, HypeHype has about half a million games created by our users, which anyone can play or remix for their liking. HypeHype's goal is to make game creation easy and fun for everyone, including casual users with no technical background. This has big implications on the technology we develop, because it just has to work with minimal technical skills while having good quality and high performance across all the devices. We are building intuitive creation tools that run even on mobile devices and are integrating AI into the workflow to make the barrier to entry even lower.

Stochastic Tile-Based Lighting
in HypeHype

SIGGRAPH 2025
Vancouver+ 10-14 August

When I joined HypeHype, the engine supported only shadowed sunlight, and to boost the visual quality of the games, I started to implement a local lighting solution. In this talk, I'll take you through the stochastic lighting algorithm that we developed to run even on budget mobile phones. I'll cover the motivation behind it, implementation details, and the results we've achieved.

SIGGRAPH 2025
Vancouver+ 10-14 August

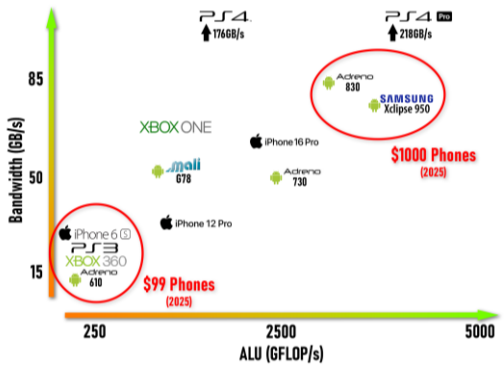
Motivation

for a new lighting algorithm

SIGGRAPH 2025 Advances in Real-Time Rendering in Games *course*

Motivation
for a new lighting algorithm

SIGGRAPH 2025
Vancouver+ 10-14 August



Mobile Challenges

- $99 budget phones
  - ~15 GB/s, ~250 GFLOP/s
- Passive cooling
  - Bandwidth usage generates a lot of heat
  - Performance throttling to manage heat
- Tile-Based Rendering & Frame Buffer Compression
  - To reduce bandwidth
  - Must use of pixel shaders (instead of compute shaders)
- Write shaders with fp16 to optimize ALU

To support local lighting on mobile phones, we need to deal with some unique constraints that significantly influence the lighting algorithm design.

One of the most critical limitations is the memory bandwidth, which is very constrained on mobile, especially in the low-end. $99 budget phones today have around 15GB/s of peak bandwidth and around 250GFLOP/s of ALU power, which is about the same as Xbox360 and PS3 released 20 years ago. Even high-end phones peak only at around 85GB/s. For comparison, Xbox Series X and PS5 Pro have over 500GB/s of bandwidth, and respectively 12 and 17TFLOP/s of ALU power.

Unlike PCs and consoles, mobile devices are passively cooled, so bandwidth-heavy operations quickly translate to heat and throttling. To put things in perspective, moving just a byte of data between GPU and DRAM consumes hundred times more power than a simple ALU operation.

Mobile GPUs rely largely on Tile-Based Rendering to conserve bandwidth. In this architecture, the frame is split to tiles and rendered tile by tile into small but fast on-chip memory. Once all triangles for a tile are rasterized, the tile render target data is written out to main memory, often using lossless framebuffer compression to further reduce the memory transfer costs.

To take advantage of this, rendering algorithms must be written with pixel shaders instead of compute shaders and the number of render passes should be minimized to reduce DRAM round-trips.

To optimize the ALU usage, algorithms should be written from ground up to work with fp16, since mobile devices have widely adopted double-rate fp16 pipelines. Besides this there are the common GPU optimizations, like minimizing branch divergence and optimizing VGPR usage.

Motivation
for a new lighting algorithm

SIGGRAPH 2025
Vancouver+ 10-14 August

UGC Challenges

- Casual creators
  - No performance tuning
  - Need robust & intuitive lighting controls
  - Consistent visuals & performance on all devices

From the user side, our platform is designed for casual creators, which poses some extra challenges for lighting.

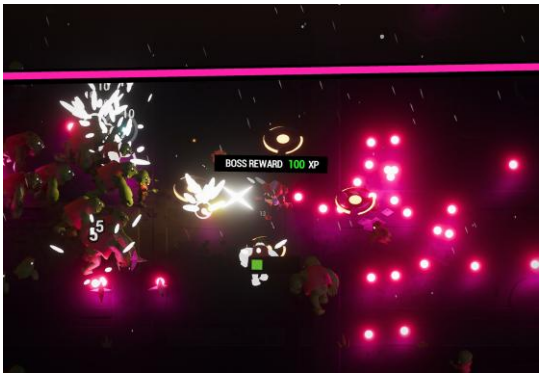
We can't expect our creators to understand performance trade-offs or tweak lighting parameters for different hardware.

Lighting controls must also be intuitive and robust to make lighting the scenes simple and fun for our creators.

And we must deliver consistent results, whether the game is created on high-end PC or phone and played on a budget Android device. The performance and consistency must hold up, or it could lead to gameplay-breaking issues.

Motivation
for a new lighting algorithm

SIGGRAPH 2025
Vancouver+ 10-14 August



UGC Challenges

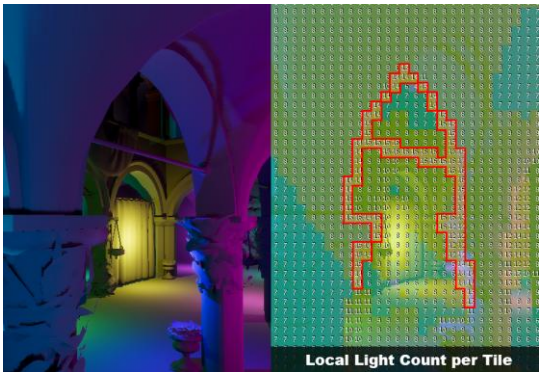
- Casual creators
  - Unpredictable dynamic light clumping
  - Unoptimal manual light placement

We must also prepare that lights may be randomly spawned and moved. For example, every pink orb here is a light created whenever an enemy dies. We can't afford large framerate drops or visually disturbing artifacts because of unpredictable light clumping.

Or, our creators might just decide to add 50 shadowed lights to a room, because why not? The creators kitbash scenes from our asset library prefabs that can include lights and quickly populate game worlds with a large number of lights.

Motivation
for a new lighting algorithm

SIGGRAPH 2025
Vancouver+ 10-14 August



Sponza model © 2010 Frank Meinl, Crytek
Downloaded from Morgan McGuire's Computer Graphics Archive

Existing Lighting Solutions

- Tiled Deferred lighting
  - **Pros**
    - Coherent memory access & lighting evaluation
  - **Cons**
    - Scales only linearly with light count per tile
    - Large tile light count even in simple scenes
    - Costly light tile culling

To explore existing solutions, we considered Tiled Deferred lighting used by many modern renderers.

The benefit of this solution is local memory access and wave coherent lighting evaluation, which is good for high GPU performance.

However, it scales only linearly with the number of lights per tile and assumes that artists light scenes carefully limiting their region of influence to keep the performance in check.

Even in quite simple scenes the light count per tile can become high, especially at object silhouettes as highlighted here. When dynamic lights are added to the mix, the lighting cost can be very unpredictable causing performance issues especially on weaker devices. Some engines limit the number of lights per tile but exceeding this limit results in flickering tile artifacts.

The light culling for the tiles is also quite costly, especially when done precisely with exact tile frustum checks.

Motivation
for a new lighting algorithm

SIGGRAPH 2025
Vancouver+ 10-14 August

Existing Lighting Solutions

- ReSTIR by Bitterli et al.
  - **Pros**
    - Fixed cost (at the expense of noise)
  - **Cons**
    - Cache unfriendly ⇒ increased bandwidth usage
    - Per pixel data & code divergences

We had also a look at ReSTIR by Bitterli et al. for a stochastic lighting solution.

The benefit of stochastic lighting is the fixed cost regardless of the scene lighting complexity, at the expense of some added noise. You can have many lights influencing a single pixel without blowing the lighting budget, like in this night shot I just took here in Vancouver. ReSTIR performs light sampling per pixel and in addition to uniform scene light sampling it resamples lights from the previous frame local pixel neighborhood.

The uniform sampling accesses the scene lights randomly which is quite cache unfriendly and increases the memory bandwidth usage. Because the result from the uniform sampling is very noisy, ReSTIR performs another spatiotemporal resampling step from the history buffer, which is another cache unfriendly and bandwidth-heavy operation. Doing such bandwidth intensive sampling per pixel, makes this algorithm quite expensive on mobile.

Also, because of the per pixel sampling, light data and types can vary randomly per pixel. This causes incoherent light data loads and diverging code execution paths during lighting evaluation, which hurts performance. We weren't aware of any existing lighting solution that matched our needs and were designed low-end mobile in mind.

This led us to develop our Stochastic Tile-Based lighting algorithm tailored specifically to work within the mobile limits.

SIGGRAPH 2025
Vancouver+ 10-14 August

Overview

of the stochastic lighting algorithm

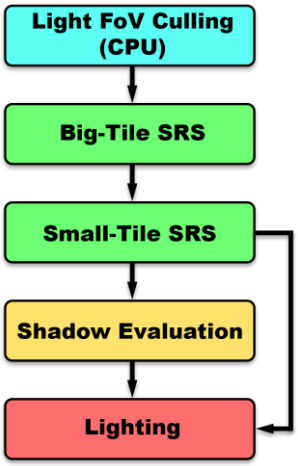
SIGGRAPH 2025 Advances in Real-Time Rendering in Games *course*

Overview
of the stochastic lighting algorithm

SIGGRAPH 2025
Vancouver+ 10-14 August

Tile-Based Solution

- Optimizes bandwidth and reduces divergences
- Can be fully implemented with pixel shaders
  - To benefit from Tile-Based Rendering & Frame Buffer Compression on mobile
- High-quality PDF to minimize noise
- Two-stage sampling process
  - With Stratified Reservoir Sampling (SRS)



The Stochastic Tile-Based lighting algorithm performs light sampling per tile, rather than per pixel.

This means that for each tile we store a single set of light samples shared across all its pixels, significantly reducing memory usage, bandwidth, and wave divergence during lighting evaluation. Much like traditional Tiled Deferred lighting, this allows threads in a GPU wave to take the same execution path and benefit from coherent data loads.

We also designed the algorithm so that all the passes can be implemented efficiently with pixel shaders to benefit from tile-based rendering architecture and frame buffer compression on mobile. That said, compute shader implementations can further improve performance, depending on the hardware architecture.

To minimize noise and denoising cost, we wanted the light sampling to be guided by a high-quality importance function that closely approximates influence of each light to a tile. For instance, if a tile is fully in shadow from a light, that light should ideally not be picked for lighting evaluation.

However, computing a precise PDF for every tile and light in the scene would be far

too expensive, so we developed a two-stage sampling process using Stratified Reservoir Sampling.

The algorithm start on CPU side with camera field-of-view culling of the scene lights.

The list of culled lights is passed to GPU for the first “Big-Tile Sampling” stage, where the screen is divided into larger tiles. For each big-tile, we use Stratified Reservoir Sampling to select a subset of lights based on a simple low-cost PDF that roughly approximates light’s importance for the tile region.

Then, in the “Small-Tile Resampling” stage, we resample few lights for small-tiles by performing another round of SRS from corresponding big-tile reservoirs. This time we use a higher-quality PDF that includes more accurate light visibility and BRDF terms.

The light sampling is followed by deferred shadow pass, where we evaluate shadow terms for small-tile light samples and for all the tile pixels. The shadow evaluation is decoupled from the lighting shader to optimize performance.

Finally, during the lighting pass, we read light samples and shadow terms for each pixel, evaluate the BRDF, and apply proper light weighting for the final lighting result.

SIGGRAPH 2025
Vancouver+ 10-14 August

Big-Tile Sampling
with SRS

SIGGRAPH 2025 Advances in Real-Time Rendering in Games *course*

Big-Tile Sampling
with SRS

SIGGRAPH 2025
Vancouver+ 10-14 August

Big-Tile Layout

- 128x128px tiles
- 16 light samples per tile reservoir
- Stratified Reservoir Sampling (SRS)
  - "Without replacement" for no light duplicates
  - Efficient to implement with pixel shaders

The light sampling starts by sampling a subset of scene lights per big-tile to reduce the number of light candidates considered during the subsequent small-tile resampling stage.

Each big-tile covers a 128x128px region of the image as shown with the checkerboard pattern.

For every big-tile, we stochastically select 16 distinct lights from the scene and write them into that tile's reservoir.

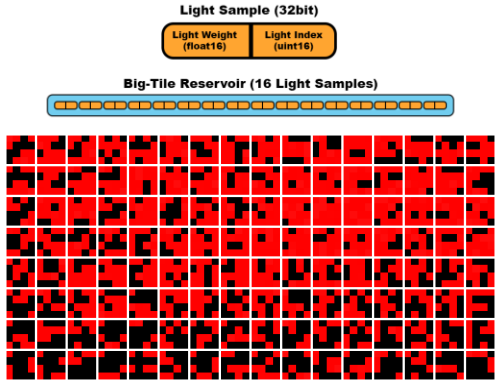
For the sampling, we use Stratified Reservoir Sampling.

SRS is "without replacement" sampling algorithm that ensures we don't have light duplicates in big-tile reservoirs and that we use the reservoir space efficiently, which is important to maintain a diverse and representative set of lights for each big-tile.

SRS can be also implemented efficiently with pixel shaders because each reservoir slot is independently sampled.

Big-Tile Sampling
with SRS

SIGGRAPH 2025
Vancouver+ 10-14 August



Big-Tile Storage

- 32bit light samples
  - Light Index → FoV-culled light data buffer
  - Light Weight = $\frac{1}{PDF}$
- 16 samples stored in 4x4px regions
  - To improve cache coherence (for read & write)

Each resulting light sample is stored in a 32bit format as shown in the image. For pixel shader implementation, the big-tile sampling shader is evaluated for each big-tile reservoir slot that exports the 32bit light sample to the render target.

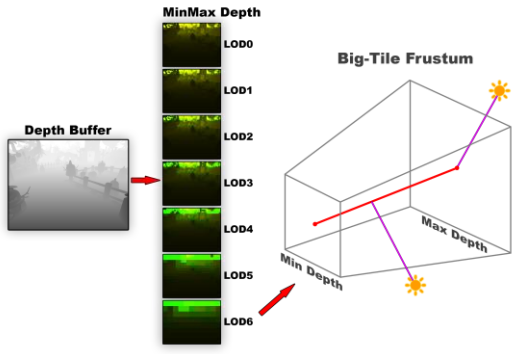
The light index refers to the buffer containing the camera FoV-culled lights.

The light weight is the inverse of the PDF.

The reservoir samples are stored in 4x4px regions in the render target for improved cache coherence. For example 1080p frame buffer requires tiny 60x36px render target for the storage.

Big-Tile Sampling
with SRS

SIGGRAPH 2025
Vancouver+ 10-14 August



Big-Tile PDF

- Use camera MinMax depth for big-tile frustum
- Construct tile central line segment from LOD6
- Calculate light distance d from the segment

$$PDF = \frac{\Phi_v}{d^2}$$

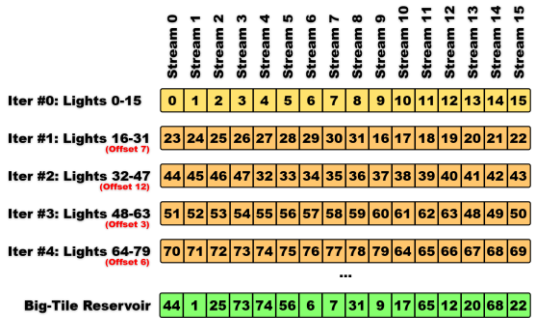
To compute the PDF for the big-tile sampling, we start by fetching the depth bounds of the tile from hierarchical MinMax depth buffer, that was generated after the G-Buffer pass.

For 128px big-tiles we use LOD6 of the MinMax depth buffer which has the same pixel footprint as the big-tile. Next, we construct the tile central line segment between the tile depth bounds, shown red in the image.

During SRS we use the same PDF for all the light types. For this we evaluate omni-light scalar illuminance at the nearest point on the line segment. This provides a rough estimate of each light's influence on the tile, and because the PDF is positive everywhere, it doesn't introduce sampling bias. The PDF is fast to evaluate but assumes that the tile pixels are roughly evenly distributed between the tile depth bounds, which may not be true. With more accurate PDF the reservoir quality could be improved, but this simple PDF has been sufficient for us for now.

Big-Tile Sampling
with SRS

SIGGRAPH 2025
Vancouver+ 10-14 August



Big-Tile SRS

- Split lights to 16 streams
- Run 1-sample A-Chao WRS for each stream
- Start streams with matching light index
- Random index offset % 16 per SRS iteration
- PRNG w/ big-tile seed for same offset seq.
- Result: 16 unique importance sampled lights

```
RNGState rng = initRNG(bigTileIdx, frameIdx);
uint selLightIdx = 0;
float selLightPDF = 0.0f;
float totalPDF = 0.0f;
for(int lightIdx = btSlotIdx; lightIdx < numLights; lightIdx = nextLightIdx(lightIdx, rng))
{
    float lightPDF = evaluateLightPDF_BigTile(lightIdx);
    totalPDF += lightPDF;
    if(randUreal1(rng) * totalPDF <= lightPDF)
    {
        selLightIdx = lightIdx;
        selLightPDF = lightPDF;
    }
}
storeReservoirSample(selLightIdx, selLightPDF ? totalPDF / selLightPDF : 0.0f);
```


To perform SRS, we divide the lights into 16 separate streams, one for each big-tile reservoir slot

and run 1-sample A-Chao Weighted Reservoir Sampling for each stream. This ensures we don't get light duplicates in the reservoir because each slot samples a unique set of lights.

We start the stream iteration with the scene light index matching the reservoir slot index. This initial ordering is done to ensure that first lights are placed in specific slots to optimize performance and to reduce noise in scenes with only few lights.

Then in the next SRS iteration we jump forward to the next group of 16 lights, generate a random offset for the iteration, and offset stream indices with modulo 16 to randomize the streams. This randomization is done to avoid heavy-weight lights competing of the same reservoir slot across big-tiles and frames.

We keep repeating this process until we have iterated through all the lights.

To ensure the same random offset sequence for each slot with a pixel shader implementation, we initialize Pseudo Random Number Generator with the seed of the

big-tile index.

At the end of this sampling pass, each big-tile reservoir has 16 unique and unbiased light candidates picked based on their approximate influence on the big-tile footprint.

Here’s a code snippet for the big-tile slot sampling using SRS based on A-Chao WRS. We iterate through scene lights for a big-tile reservoir slot using the described random offset sequence. For each iteration we evaluate the PDF for each light and use the PDF to probabilistically pick the light. Once completed we store the light and its weight to the reservoir.

Big-Tile Sampling
with SRS

SIGGRAPH 2025
Vancouver+ 10-14 August

Here's a comparison how the big-tile SRS reduces noise versus uniform sampling. On the left we have an image where we pick small-tile samples from 16 uniformly sampled scene lights. On the right we have an image with lights resampled from the big-tile reservoirs of 16 samples, which has significantly less noise.

SIGGRAPH 2025
Vancouver+ 10-14 August

Small-Tile Resampling

with SRS

SIGGRAPH 2025 Advances in Real-Time Rendering in Games *course*

Small-Tile Resampling
with SRS

Small-Tile Pixel Layout

- Resample 1-4 lights from big-tile reservoirs
- Use SRS for no light duplicates
- Independent of the scene's lighting complexity
- Each small-tile covers 256 pixels
- Tiling artifacts with regular 16x16px layout □

After the initial big-tile light sampling, we resample 1-4 lights for small-tiles from the big-tile reservoirs. These samples will be used for the final lighting evaluation.

For the resampling we'll have another round of Stratified Reservoir Sampling to ensure no light duplicates in small-tile reservoirs.

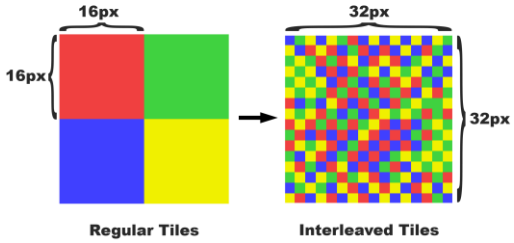
This resampling step is independent of the scene's lighting complexity because the resampling is done from the fixed-size big-tile reservoirs.

Each small-tile covers 256 pixels, and for resampling we pick a big-tile reservoir based on the location of the small-tile center point.

However, if we lay out the small-tile pixels in a regular 16x16px grid, it leads to visible tiling artifacts due to sampling correlation as shown in the image □

Small-Tile Resampling
with SRS

SIGGRAPH 2025
Vancouver+ 10-14 August



Small-Tile Sample Interleaving

- Interleave 4 neighbor tiles on 32x32px footprint
  - 64 2x2px quad samples per tile
  - Gaussian-Poisson sample distribution
- Repeat the sample patterns over the image

To break these tiling artifacts, we interleave the small-tile pixels with 4 neighbor tiles over a larger 32x32px footprint.

Specifically, we distribute 64 2x2px quads for each four tiles. Here we have the 64 precomputed quad sample distributions for the four tiles shown in red, green, blue and yellow.

For the sample distribution we use Gaussian-Poisson distribution to concentrate a bit more samples to the tile centers while avoiding placing samples from the same tile right next to each other.

These sampling patterns are then repeated over the entire rendered image.

Small-Tile Resampling
with SRS

SIGGRAPH 2025
Vancouver+ 10-14 August

Small-Tile Sample Interleaving

- Reduces small-tile tiling artifacts □
- Big-tile tiling artifacts remain □

This interleaved sampling pattern reduces the small-tile tiling artifacts.

However, we can still see square big-tile lighting artifacts, because the big-tile reservoirs are point sampled at the small-tile centers.

Small-Tile Resampling
with SRS

Small-Tile Sample Interleaving

- Sample big-tiles with stochastic bilinear filter
  - Random offset small-tile sampling center

To fix this, instead of point sampling the big-tiles, we use stochastic bilinear filtering between neighboring big-tiles.

We implement this by randomly offsetting small-tile centers when selecting the big-tile for resampling. The result is a smoother transition between big-tiles and eliminates the tiling artifact. We can still see some interleaved small-tiles, but this image is now much better suited for spatiotemporal denoising.

Small-Tile Resampling
with SRS

SIGGRAPH 2025
Vancouver+ 10-14 August

Small-Tile PDF

- Resample using more refined PDF
- Stochastic PDF evaluation
  - Pick N random small-tile quad sample points
  - Average reflected luminance at the points for PDF

$$PDF = \frac{1}{N}\sum_{N} E_v \, f_v(\omega_i, \omega_o)(V \cdot Cos^{+}\theta)$$

N = *Number of Small-Tile sample points*
E_v = *Unshadowed illuminance (luminance-weighted)*
f_v = *BRDF (luminance-weighted)*
V = *Shadow term*

For the small-tile resampling we use a more refined PDF that better estimates light's importance to the tile.

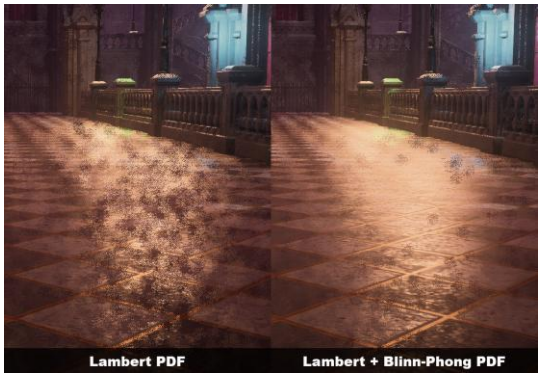
We use stochastic PDF evaluation by picking 4 random small-tile quad sample points out of the 64.

Then average reflected luminance at the points using light's shadowed illuminance and a BRDF. We calculate the product of the standard luminous-efficiency weighted light illuminance and BRDF functions for the PDF, which means that materials and lights are assumed to be monochromatic. It would be better to perform the weighting after the RGB illuminance and BRDF product so that for example a red light illuminating a blue surface would result in lower PDF, but we compromise currently for performance at the expense of some added noise.

Here's an example image showing how the shadow term influences noise. If a tile is completely shadowed or all its surface normals point away from the light we don't pick the light for the lighting evaluation, which reduces noise.

Small-Tile Resampling
with SRS

SIGGRAPH 2025
Vancouver+ 10-14 August



Small-Tile PDF

- Closer BRDF approx. reduces noise
  - Using Lambert + Blinn-Phong for lower cost
- Average material parameters at the samples
  - Luminous-efficiency weighted to lower ALU & VGPRs

Initially we used Lambert BRDF for the PDF but replaced it with a specular BRDF to reduce specular noise.

We found that Lambert + Blinn-Phong is good enough but cheaper approximation for the PDF to reduce noise of the final Lambert + GGX BRDF. Also worth noting that we support currently only one lit BRDF, and it might be good to implement different PDF approximations for multiple BRDFs to optimize noise.

To reduce the ALU usage and VGPR pressure, we also first average luminous-efficiency weighted material parameters at the tile sample points and then use the same parameters in reflected luminance calculation of the points.

Small-Tile Resampling
with SRS

Small-Tile PDF Bias

- For unbiased resampling:

$$PDF > 0 \text{ whenever } L_o > 0$$

- Stochastic PDF evaluation not unbiased
- Add small offset ε to $(V \cdot Cos^{+}\theta)$-term

$$PDF = \frac{1}{N}\sum_{N} E_v \, f_v(\omega_i, \omega_o)(V \cdot Cos^{+}\theta + \varepsilon)$$

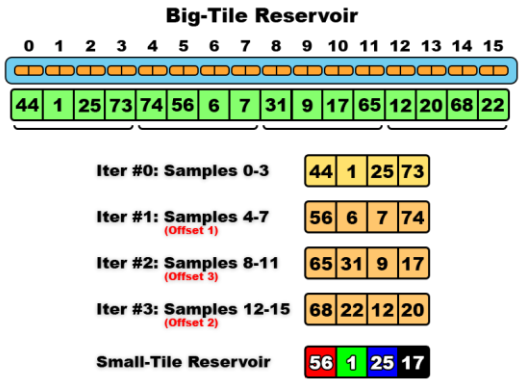
Ideally, our resampling stays unbiased by ensuring the PDF is non-zero everywhere the tile's reflected luminance is non-zero.

In practice we don't fully meet that requirement: For example, it's possible that all the four PDF sample points fall in shadow even though some tile pixels are lit. This adds a biased lighting edge noise as shown in the areas highlighted with the red arrows. In the TAA image at the bottom this biased noise shows as slight darkening in the area.

The bias is most obvious in simply lit areas and to mitigate the issue, we add a small offset to the shadowed cosine term, before it's multiplied into the PDF. After denoising the bias is mostly gone and unnoticeable.

Small-Tile Resampling
with SRS

SIGGRAPH 2025
Vancouver+ 10-14 August



Small-Tile SRS

- Resample N lights w/ big-tile SRS-strategy
  - Split big-tile reservoir to N streams
- Fixed resampling cost independent of N
- Store in a 1-4 channel texture (32bits/channel)

```
RNGState rng = initRNG(smallTileIdx, frameIdx);
uint selLightIdx = 0;
float selLightPDF = 0.0f;
float totalPDF = 0.0f;
for(int btSlotIdx = stSlotIdx; btSlotIdx < 16; btSlotIdx = nextSlotIdx(btSlotIdx, rng))
{
    ReservoirSample btSample = fetchBigTileSample(btSlotIdx);
    float lightPDF = evaluateLightPDF_SmallTile(btSample.lightIdx);
    float resamplePDF = lightPDF * btSample.weight;
    totalPDF += resamplePDF;
    if(randUreal1(rng) * totalPDF <= resamplePDF)
    {
        selLightIdx = btSample.lightIdx;
        selLightPDF = lightPDF;
    }
}
storeReservoirSample(selLightIdx, selLightPDF ? totalPDF / selLightPDF : 0.0f);
```


For the small-tile resampling we apply the same SRS-strategy as in the big-tile pass.

For the small-tile reservoir of N light samples, we split the 16 big-tile sample candidates into N independent streams.

The resampling cost is independent of N, because of the stream splitting. Only the big-tile reservoir size and the small-tile PDF cost influences the total cost of the pass.

Like with the big-tile SRS, we start by picking the first N samples from the big-tile reservoir for the streams. In this example we use N=4.

For the next iteration we shuffle the next batch of 4 light samples across the streams with a random offset, to reduce the likelihood of high-importance lights competing of the same reservoir slots for nearby small-tiles.

We then repeat this process until all the 16 big-tile samples have been processed. At the end of the resampling stage, we have 4 unique high-quality light samples that will be used for the lighting evaluation.

The small-tile reservoir is exported from the pixel shader to a 1-4 channel render

target, using the same 32bit format as for the big-tile reservoirs. It’s possible to have more than 4 samples in the reservoir, but we max at 4 for performance and to fit the reservoir into one render target. For 1080p this requires 120x68px render target, or from 32KB to 128KB of memory depending on the reservoir size.

Here’s a code snippet for the small-tile slot resampling with SRS. We iterate through the big-tile reservoir samples for the slot as described. For each iteration we evaluate the light PDF and calculate resampling PDF that unbiases the big-tile sampling bias. This resampling PDF is then used to probabilistically pick the light. Once completed we store the light and its weight to the reservoir.

SIGGRAPH 2025
Vancouver+ 10-14 August

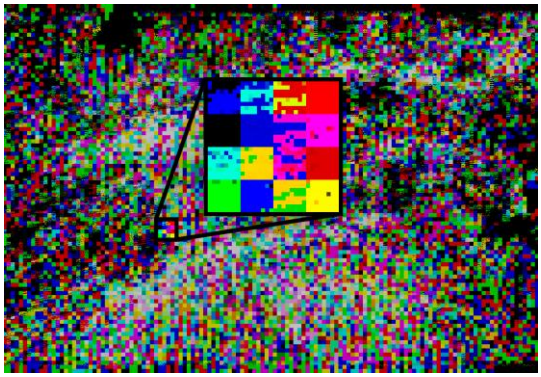
Shadow Sampling

with deferred shadows

SIGGRAPH 2025 Advances in Real-Time Rendering in Games *course*

Shadow Sampling
with Deferred Shadows

SIGGRAPH 2025
Vancouver+ 10-14 August



Deferred Shadow Pass

- Decoupled from the final lighting pass
- Evaluate shadows per 2x2px quad
- Simplifies the lighting shader
  - Less wave divergences, data reads & VGPR pressure
- Shadow terms stored in 8x8px blocks per tile
  - Reduces wave divergences
- 8bit shadow terms with 1bit temporal noise
- Evaluate with shadow maps, ray-tracing, etc.
  - No influence in the lighting shader performance

After the small-tile resampling, we evaluate the shadow terms for the light samples. We perform this in a separate pixel shader pass, decoupled from the final lighting evaluation.

This enables us to evaluate shadows at a lower resolution for better performance. Specifically, we evaluate shadows once per 2x2px small-tile quad instead of per pixel.

This not only improves shadow evaluation performance by cutting the evaluations to one quarter, but also simplifies the lighting shader. The wave divergence, data access and VGPR pressure is reduced because the light type specific shadow sampling is kept out of the shader.

Since the algorithm is implemented entirely in pixel shaders, shadow terms are stored in 8×8 pixel blocks, corresponding to the 64 quad samples of a small-tile. This organization reduces wave divergences because of how GPUs generally pack pixel shading to waves. The shadow evaluation code is largely unified for spot and omni lights, but sunlight uses cascaded shadow maps with a separate path.

To reduce memory usage and bandwidth, we use only 8bit shadow terms and add 1bit of temporal noise to eliminate banding with TAA. For 1080p this uses from 0.5MB

to 2MB of memory.

The deferred shadow pass also enables the shadow evaluation to be implemented using different techniques, such as shadow maps or ray-tracing, without influencing the final lighting shader performance.

Shadow Sampling
with Deferred Shadows

SIGGRAPH 2025
Vancouver+ 10-14 August

Shadow Map Atlas

- Dynamically managed, 16bpp
- Square pow-of-2 spot & omni shadow maps
  - Standard projected shadow maps for spot
  - Octahedral shadow maps for omni
  - Resized based on distance from the camera

Because of our target platforms, currently we use shadow maps for shadow evaluation, stored in a persistent, dynamically managed 16bpp atlas.

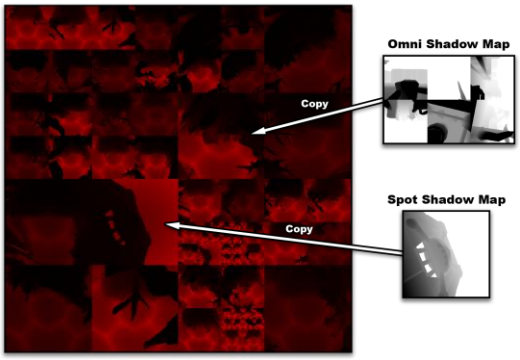
The atlas contains both spot and omni shadow maps, which are square and power-of-two in size.

For spotlights, we use standard projected shadow maps, while for omni lights we use octahedral-mapped shadow maps.

We also resize the shadow maps based on lights distance from the camera, to balance the shadow quality with the limited atlas storage.

Shadow Sampling
with Deferred Shadows

SIGGRAPH 2025
Vancouver+ 10-14 August



Shadow Map Atlas Update

- Render to temp depth buffer, then copy to atlas
- Upon copy: linearize, bias and add 2px border
  - For omni also map cube faces to octahedral space
- Static shadow maps (static geo)
  - Updated only upon light or shadow res. changes
- Dynamic shadow maps (static + dynamic geo)
  - Updated constantly
- Load balanced shadow map updates

When updating a shadow map for a light, we first render geometry within the light's range to a temp depth buffer, and then copy the data into the atlas.

For spotlights, we render a regular depth buffer from the light's point of view.

Then upon the copy we linearize and bias the depth values.

For omni lights, we render a cubemap to the depth buffer

and during the copy, we map the cube faces to octahedral space in addition to depth linearization and biasing. We also add 2px border upon both copies to support PCF filtering.

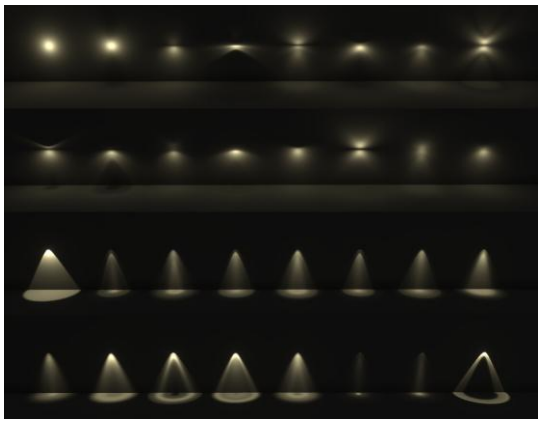
We support both static and dynamic shadow maps. Static maps are updated only when light's transform or shadow resolution changes. Static maps include only static geometry and are meant for decorative lighting.

Dynamic maps are more expensive because they are constantly updated, and include both static and dynamic geometry.

To avoid performance spikes when many shadow maps require updates, we load balance the updates with a priority queue system to spread the shadow map rendering across multiple frames.

Shadow Sampling
with Deferred Shadows

SIGGRAPH 2025
Vancouver+ 10-14 August



Shadow Filtering

- 4-tap “Hardware” PCF w/ stochastic PCF
  - Only 1 gather per shadow term evaluation
- Apply IES profile + angular falloff
  - Applied also for small-tile PDF
- Improve further with:
  - PCSS, detail shadows, ray-traced shadows

For shadow filtering, we use 4-tap “hardware” PCF with stochastic PCF to produce softer, more natural-looking shadows.

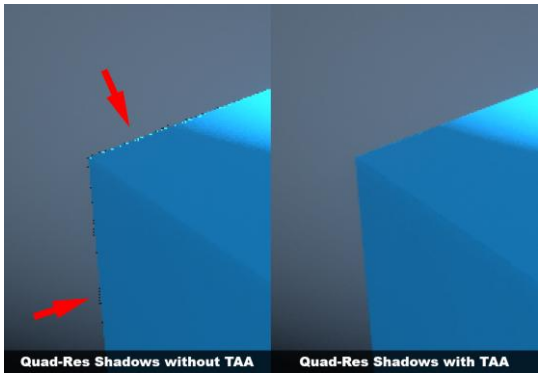
This requires only 1 gather operation from the shadow map per shadow term evaluation.

At this stage, we apply also selected IES light profile and angular spot falloff to the shadow term, allowing light-specific shape falloff.

We apply IES profiles also during small-tile PDF evaluation to reduce noise.

We plan to improve shadow quality further with PCSS for contact hardening shadows, screen-space ray-marched detail shadows, and ray-traced shadows, which can be enabled on higher-end devices.

Shadow Sampling
with Deferred Shadows



Quad-Res Shadow Evaluation Issues

- Potential artifacts at object silhouettes
- TAA mostly mitigates the issue
- Full-res shadows possible at additional cost

Sharing one shadow term per 2x2px quad works well in most cases, but it can introduce artifacts at object silhouettes where quad pixels should have quite different shadow term values.

However, randomized tile sampling and TAA mostly mitigates this issue. While it can still introduce subtle lighting bias at the edges, the artifacts are quite minor.

If necessary, we can also fall back to full-resolution shadow evaluation to improve the quality at an additional cost.

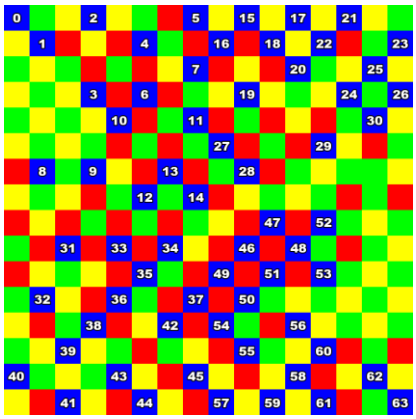
SIGGRAPH 2025
Vancouver+ 10-14 August

Lighting Evaluation

SIGGRAPH 2025 Advances in Real-Time Rendering in Games *course*

Lighting Evaluation
with Deferred Shadows

SIGGRAPH 2025
Vancouver+ 10-14 August



Lighting Evaluation

- Fetch light samples and shadow terms
- Use 16x16 8bpp LUT for inverse tile mapping
  - Maps screen quad coords to tile & sample indices
  - Compute shader impl. doesn't require the LUT
- Accumulate the BRDF for the samples
  - Weigh by shadow term and sample weight
- Fixed cost independent of the scene lighting

In the lighting evaluation stage, we first fetch small-tile light samples and shadow terms for the pixel.

Since we want this stage to also run in a pixel shader, we need a way to map screen pixels back to small-tile and sample indices.

To achieve this, we use a 16x16px inverse LUT, that links screen-space quad coordinates to the interleaved tile and sample indices. With this information, we can then fetch both the light samples and shadow terms for each pixel.

With compute shaders the inverse mapping isn't necessary, and we could directly evaluate the light samples for the tile pixels and write the lighting result to the lighting buffer with scattered writes.

Finally, we accumulate the Lambert + GGX BRDF contribution for each light sample, weighted by both the sample weight and its shadow term, and write the result to the lighting buffer.

Because we store a fixed 1 to 4 light samples per tile, this pass has also a fixed cost independent of the scene lighting complexity. The lighting evaluation is also quite

unified across all our light types because of the decoupled shadow evaluation, which reduces the divergence with the pixel shader implementation. With compute shaders we could have coherent lighting evaluation of tile pixels packed to waves, further reducing the divergence.

SIGGRAPH 2025
Vancouver+ 10-14 August

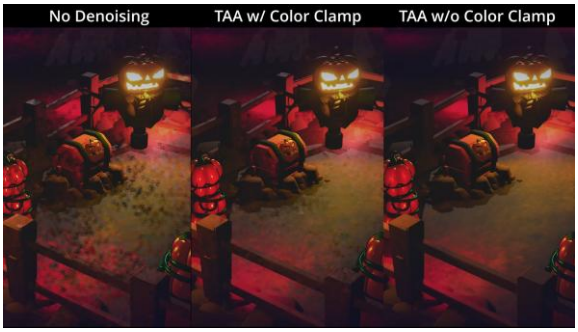
Denoising

work in progress

SIGGRAPH 2025 Advances in Real-Time Rendering in Games *course*

Denoising
work in progress

SIGGRAPH 2025
Vancouver+ 10-14 August



Denoising with TAA

- History color clamp has issues with noise
  - Flashing lighting splotch artifacts
- WIP: Noise-compatible ghosting suppression

To keep the denoising cost low, we are planning to improve temporal anti-aliasing to work as our denoiser and avoid the need for a separate denoising pass, which would consume more bandwidth.

However, the main challenge with the standard TAA is the 3x3px neighborhood history color clamping designed to reduce ghosting. In our case, this causes flashing lighting splotches shown in the middle, because the pixel neigborhoods may not render key lights for a frame, limiting the color range.

So, we need to either make the color clamping more compatible with the noisy input, find alternative ghosting suppression techniques, or provide input data that better suits TAA, or with a combination of these approaches. This is still an active area of R&D for us.

SIGGRAPH 2025
Vancouver+ 10-14 August

Visual Results

SIGGRAPH 2025 Advances in Real-Time Rendering in Games *course*

Visual Results
1spp vs 4spp

SIGGRAPH 2025
Vancouver+ 10-14 August

For visual results, here is a comparison between 1 and 4 samples-per-pixel lighting. In the noisy image on the left, the 1spp image has more noise, as expected. In the TAA image on the right, we can also see that the 4spp result is able to reproduce some of the finer lighting details, like sharp specular reflections in detailed geometry. Both converge to quite similar and acceptable results though without major differences.

Visual Results
1spp vs 4spp

SIGGRAPH 2025
Vancouver+ 10-14 August

For visual results, here is a comparison between 1 and 4 samples-per-pixel lighting. In the noisy image on the left, the 1spp image has more noise, as expected. In the TAA image on the right, we can also see that the 4spp result is able to reproduce some of the finer lighting details, like sharp specular reflections in detailed geometry. Both converge to quite similar and acceptable results though without major differences.

Visual Results
4-64 Big-Tile samples

Here's a visual comparison how big-tile reservoir size influences the lighting results. Decreasing the number of big-tile samples darkens the scene, especially in the distance. The darkening is lighting bias caused by capping the light weights to reduce fireflies. Otherwise, increasing the big-tile reservoir reduces noise and improves lighting details because the small-tile resampling can pull more relevant light samples from a richer set.

Visual Results
4-64 Big-Tile samples

Visual Results
4-64 Big-Tile samples

Visual Results
4-64 Big-Tile samples

Visual Results
4-64 Big-Tile samples

SIGGRAPH 2025
Vancouver+ 10-14 August

SIGGRAPH 2025
Vancouver+ 10-14 August

Performance

on mobile and PC

SIGGRAPH 2025 Advances in Real-Time Rendering in Games *course*

Performance
on mobile and PC

SIGGRAPH 2025
Vancouver+ 10-14 August

Testing Hardware

Device	Resolution	ALU / Bandwidth	Power
$99 Budget Android (Adreno 610)	745 × 360	243 GFLOP/s, 14.9 GB/s	≈3 W
$1000 High-end Android (Adreno 830)	2256x1080	3.4 TFLOP/s, 85 GB/s	≈5 W
Laptop (RTX 4070)	3840 × 2259	15.6 TFLOP/s, 256 GB/s	≈115 W

For the performance profiling, we are using the following hardware setup. It covers one of our weakest Android target phones, a high-end Android phone and a more powerful high-end laptop. Adreno 610 is found in $99 budget phones today and matches roughly Xbox360 and PS3 for ALU power and memory bandwidth, but runs at 3W. Adreno 830 is found in $1000 phones and matches around 10 year old consoles and desktop GPUs, but runs at 5W. We also use lower rendering resolutions on weaker devices.

Performance
on mobile and PC

SIGGRAPH 2025
Vancouver+ 10-14 August

First, we'll have an apples-to-apples visual test, comparing our old sun-only lighting with the Stochastic Tile-Based lighting. Sun-only lighting simply evaluates shadowed sunlight for every pixel without any kind of light classification. Visually, the sun-only lighting and our STB lighting look almost identical. The main difference is that STB lighting evaluates sun shadows for 2x2px quads instead of pixels, so they appear slightly softer.

Performance
on mobile and PC

SIGGRAPH 2025
Vancouver+ 10-14 August

	Adreno 610 (745x360)	Adreno 830 (2256x1080)	RTX 4070 Laptop (3840x2259)
Sun-only Lighting	1.51ms	0.16ms	0.11ms
STB Lighting (1spp)	2.17ms	0.51ms	0.43ms
Big-Tile SRS	0.01ms	0.01ms	0.00ms
Small-Tile SRS	0.34ms	0.18ms	0.16ms
Shadows	0.49ms	0.08ms	0.14ms
Lighting	1.33ms	0.24ms	0.13ms
STB vs Sun-only	+0.66ms/44%	+0.35ms/219%	+0.32ms/291%

Our lighting passes evaluate also specular and diffuse IBL, SSAO upscaling, and so forth, but these numbers include only the direct lighting cost.

With 1spp stochastic lighting, the cost on the Adreno 610 increases only by 0.66ms, which is quite modest given it now also handles arbitrary local lights.

On Adreno 830 and RTX 4070 the cost increase is around 0.35ms, but the relative cost increase is higher. These platforms could potentially benefit from the compute shader implementations and merging passes to bridge the gap.

Performance
on mobile and PC

SIGGRAPH 2025
Vancouver+ 10-14 August

We also profiled how the STB lighting performance scales with samples-per-pixel with 50 local lights in camera FoV, and how it compares to the sun-only lighting performance. The sun-only lighting on the left differs visually of course because it renders only the faint directional moonlight.

Performance
on mobile and PC

SIGGRAPH 2025
Vancouver+ 10-14 August

	Adreno 610 (745x360)	Adreno 830 (2256x1080)	RTX 4070 Laptop (3840x2259)
Sun-only Lighting	1.76ms	0.17ms	0.12ms
STB Lighting (1spp)	3.25ms	0.68ms	0.58ms
Big-Tile SRS	0.01ms	0.01ms	0.00ms
Small-Tile SRS	1.05ms	0.29ms	0.24ms
Shadows	0.57ms	0.10ms	0.15ms
Lighting	1.62ms	0.28ms	0.19ms
STB Lighting (4spp)	7.71ms	1.34ms	1.08ms
Big-Tile SRS	0.01ms	0.01ms	0.00ms
Small-Tile SRS	1.46ms	0.34ms	0.30ms
Shadows	1.47ms	0.19ms	0.16ms
Lighting	4.77ms	0.80ms	0.62ms
STB 1spp vs Sun-only	+1.49ms/85%	+0.51ms/300%	+0.46ms/383%
STB 4spp vs 1spp	+4.46ms/137%	+0.66ms/97%	+0.50ms/86%

In this scene the sun-only lighting is a bit more costly, but 1spp adds an extra 1.5ms on budget phone and around 0.5ms on the high-end phone and the laptop.

Notably the Small-Tile SRS pass is now 3x as expensive as in the previous scene on the budget phone because it needs to resample from the full big-tile reservoirs. An easy way to reduce the cost would be to reduce big-tile reservoir size, at the cost of some added noise.

4spp lighting gets quite expensive on the budget phone but is more reasonable on the high-end phone and the laptop, especially at those resolutions.

The cost of big-tile SRS pass is still tiny, but could be improved to fixed cost with uniform sampling if it becomes an issue with larger light counts. I tested the performance in a separate scene with 500 lights and the Big-Tile sampling cost was still only 0.02ms on all the platforms.

SIGGRAPH 2025
Vancouver+ 10-14 August

Closing Thoughts

and the future

SIGGRAPH 2025 Advances in Real-Time Rendering in Games *course*

Closing Thoughts
and the future

SIGGRAPH 2025
Vancouver+ 10-14 August

Some Future Work

- Compute shader implementations & pass merging
- Continue improving TAA for denoising
- Improve big-tile PDF
- Improve small-tile PDF
- Parameter tuning
- Semi-transparent surface & volumetric lighting

We presented a new stochastic lighting algorithm that moved light sampling from per-pixel to per-tile, drastically cutting bandwidth and divergence, and making fully dynamic complex lighting a viable option even on budget mobile phones. The development is still work in progress, but the results are encouraging for both quality and performance that we plan to keep improving in the future. Some of our future work includes:

Implementing compute shader variants of the current pixel shader passes and profiling on different devices. There are good ALU and bandwidth optimization opportunities that might translate to mobile devices as well. With compute shaders we could also merge the last three passes; small-tile resampling, shadows and lighting, to a single pass which should give significant performance boost on PC & consoles and possibly also on some mobile devices.

We'll also continue the work on TAA to improve denoising and lower the level of noise generated by the lighting algorithm. While the lighting cost remains fixed, the level of noise increases as more lights are illuminating a region. Our biggest challenge currently is to make TAA to work well with noisy output produced by the lighting algorithm. We are still hopeful that we can denoise the result with a modified TAA instead of having to implement a separate spatiotemporal denoising pass. It might be

also possible to reduce noise with pervious frame small-tile reservoir resampling similarly to ReSTIR, but this requires more R&D.

The current big-tile PDF estimates lights influence on the tile using the central line segment between the tile depth bounds. If the depth range is large and the tile pixels are mostly near the near and far planes, lights in the middle of the range are picked too frequently increasing noise. It would be better to weigh the PDF proportionally to number of pixels at certain depth for higher quality big-tile reservoir samples. Another potential improvement is to set PDF to 0 if spotlight cone doesn't intersect the big-tile frustum. The current big-tile sampling is very efficient so we could spend some more cycles to improve the big-tile sample quality to better handle a wider range of scenarios.

The small-tile PDF can also be improved for both performance and quality. Our stochastic PDF evaluation requires currently 4 shadowed illuminance and BRDF evaluations which is rather expensive and results in some bias at lighting boundaries. We might be able to reduce this to one conservative shadow evaluation using tile bounds and hierarchical shadow map, and to have single conservative BRDF evaluation by generating a low-res version of the G-buffer. We don't either use any tone-mapping in the PDF evaluation, which should also help to bring down the noise.

There are also potential low-hanging fruits to improve the quality and performance by tuning the existing algorithm parameters. For example, big-tile reservoir size, big-tile pixel size, sampling patterns, and so forth can be likely improved.

The lighting algorithm supports currently only fully opaque surfaces rendered in the G-Buffer. To add support for rasterized semi-transparent surfaces the light sampling and reservoirs should be extended to volumes that can be efficiently sampled during the rasterization. The algorithm can be also extended to support volumetric lighting for example by performing small-tile resampling upon volumetric ray marching.

Acknowledgements

Thanks!

Natalya Tatarchuk
Sébastien Lagarde
Sebastian Aaltonen
Janne Gröndahl
HypeHype
and my wife Laura

So, that wraps up my presentation and thank you for listening! I would like to thank Natalya Tatarchuk, Sébastien Lagarde, Sebastian Aaltonen and Janne Gröndahl for reviewing and helping to prepare the presentation, and HypeHype for sponsoring the trip. And finally, my wife Laura for her patience and support.

SIGGRAPH 2025
Vancouver+ 10-14 August

Q&A

questions, thoughts, ideas?

SIGGRAPH 2025 Advances in Real-Time Rendering in Games *course*

Appendix
Fast GLSL PRNG

SIGGRAPH 2025
Vancouver+ 10-14 August

```
struct RNGState
{
    uint seed;
};

RNGState initRNG(uint seed)
{
    RNGState state;
    state.seed = seed;
    return state;
}

RNGState initRNG(uvec2 pos, uint seed)
{
    seed += pos.x * 283 + pos.y * 2097169;
    RNGState state;
    state.seed = seed;
    return state;
}

// return float random number in range [0, 1)
float randUnitA(inout RNGState state)
{
    state.seed *= 2747636423u;
    state.seed += 2654435769u;
    return uintBitsToFloat((state.seed >> 9) | 0x3F800000) - 1.0f;
}

// return float random number in range [-1, 1)
float randRealA(inout RNGState state)
{
    state.seed *= 2747636423u;
    state.seed += 2654435769u;
    return uintBitsToFloat((state.seed >> 9) | 0x40000000) - 3.0f;
}

// return uint random number in range [0, 65535]
uint randUint16(inout RNGState state)
{
    state.seed *= 2747636423u;
    state.seed += 2654435769u;
    return state.seed >> 16;
}
```