SIGGRAPH
2025
The Premier Conference & Exhibition on
Computer Graphics & Interactive Techniques
Ray tracing the world of
Assassin's Creed Shadows
Ubisoft Montréal
SIGGRAPH 2025 Advances in Real-Time Rendering in Games course

SIGGRAPH 2025
Vancouver+ 10-14 August

LUC LEBLANC

Technical Lead Rendering

Ubisoft Montréal

Anvil Pipeline – Rendering

MELINO CONTE

Team Lead Rendering

Ubisoft Montréal

Anvil Pipeline – Rendering

Special thanks to all present and past contributors :

William Bussière,
Nicolas Lopez,
Sylvain Gretchko,
Eric Charpenay,
Robert Foriel
Daniel-Sabin Delion,
Snowdrop's Render Domain Team

We are programming team lead and technical lead for Anvil's Rendering team focused on lighting and ray tracing topics.
Before we start, our presentation today is only possible because of many years of work from our team and internal collaborators. Therefore, I want to first thank them for their involvement and dedication to landing the tech.

The AC games are systemic large-scale open worlds. AC Shadows is our most dynamic one yet.

Scalability

SIGGRAPH 2025 Vancouver+ 10-14 August

Time of day
- Dynamic transitions
- 11 key frames

Weather
- 14 unique states
- Dynamic transitions
- No impact on bake

Seasons
- 4 seasons
- No transitions

Not only do we have dynamic time of day,
But also 14 unique weather systems and all their transitions,
And four seasons that each impact materials, weather effects and TOD.
This brings a lot of variations that we need to account for lighting. As you can imagine, this does not scale easily with baked offline solutions.

Uniform distribution scaling

- Memory does not scale with uniform probe distribution

	Area	Probe distance	Baked key frames	Bake time	Physical Disc size
Assassin's Creed Unity	4 km^2	0.5 m	4	4 days	15 GB
Assassin's Creed Syndicate	6 km^2	1 m	9	3 days	9 GB
Assassin's Creed Origins*	256 km^2	1 m	11	156 days	468 GB
Assassin's Creed Shadows*	256 km^2	1 m	44	624 days	~1.9 TB
Assassin's Creed Shadows	256 km^2	0.5 – 2 m	33	5 days	10 GB

*extrapolated with uniform probe distribution

- 11 baked key frames for time of day
- x3-4 for seasons

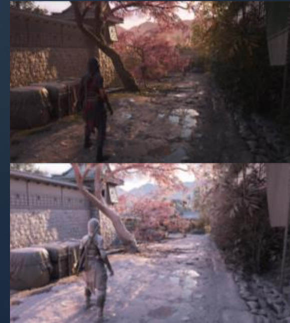
Taking a look at how big variations have become over the years, if we use uniform probe distribution from Unity, the world has taken a very different scale.
As uniform probe distribution does not scale well, we have had to make sacrifices in the baked GI version for AC Shadows, such as a unified GI for spring and summer.

Scalability

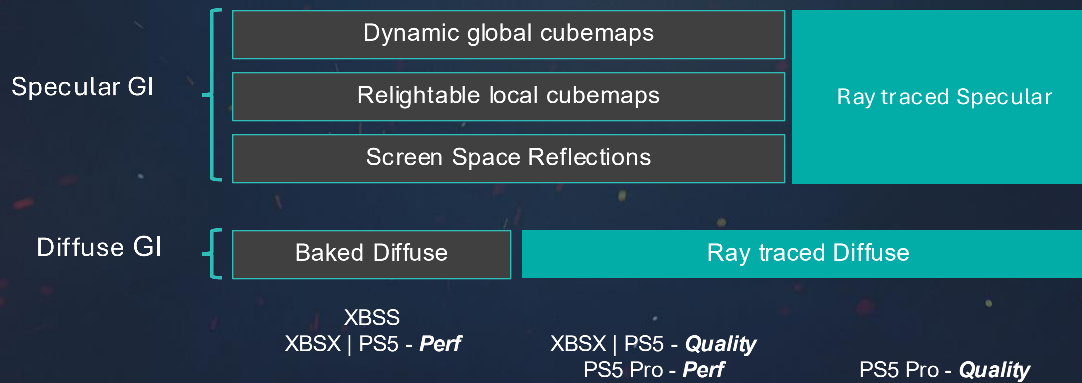
GI over the years

Open World Irradiance Volumes
Unity, 2014

Large Scale Sparse GI
Origins, 2017


Ray traced Diffuse and Specular
Shadows, 2025

We have therefore seen an evolution of global illumination systems over the years, accommodating the increasing scale. Our first step back for origins was using sparse volumes with manually added details in necessary regions.
For AC Shadows we have implemented fully dynamic ray-traced global illumination, for both diffuse and specular.



Our need for scalability is more than just world dynamism, but also platform scalability. As you can see, a single platform, can support both our shipped GI solutions.
Before we start the talk, here's a quick video showcase of environments before we dig into the details!

SIGGRAPH 2025
Vancouver+ 10-14 August

- (AC Shadows Video)

Talk structure

SIGGRAPH 2025 Vancouver+ 10-14 August

Algorithm and implementation

一 Summary
二 Ray tracing scene description
三 Probe volume
四 Per-pixel diffuse algorithm
五 Indirect specular

Challenges in AC Shadows

一 Alpha-tested materials
二 Translucency
三 Light leaks
四 Occlusion
五 Noise

Performances

Limitations and future work

We have organized our talk in two main sections. First you will deep-dive with Luc into our RTGI algorithm and it’s implementation, which we will detail as much as possible. Then, I will present some challenges specific to AC Shadows, their source, and our solutions. We will then conclude with performance, limitations of the algorithm, limitations of our implementation and our planned future work.

Algorithm and implementation

Summary

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

SIGGRAPH 2025 Vancouver+ 10-14 August

Previous work

[Lopez 2025] Rendering Assassin's Creed Shadows

[Koshlo 2024] Ray Tracing in Snowdrop: Scene Representation and Custom BVH

[Kuenlin 2024] Ray Tracing in Snowdrop: An Optimized Lighting Pipeline for Consoles

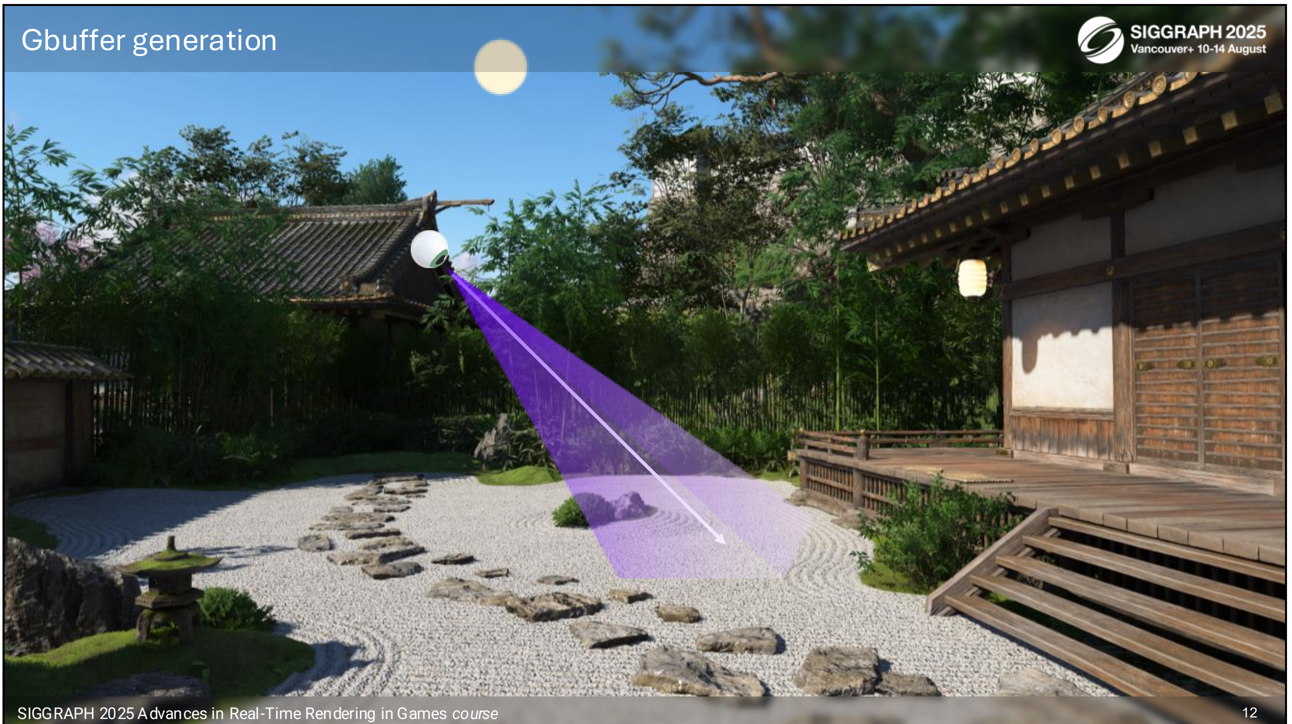
[Zhdan 2021] Reblur: A Hierarchical Recurrent Denoiser

[Majercik et al. 2021] Dynamic Diffuse Global Illumination Resampling

[Achard 2019] Exploring Raytraced Future in Metro Exodus

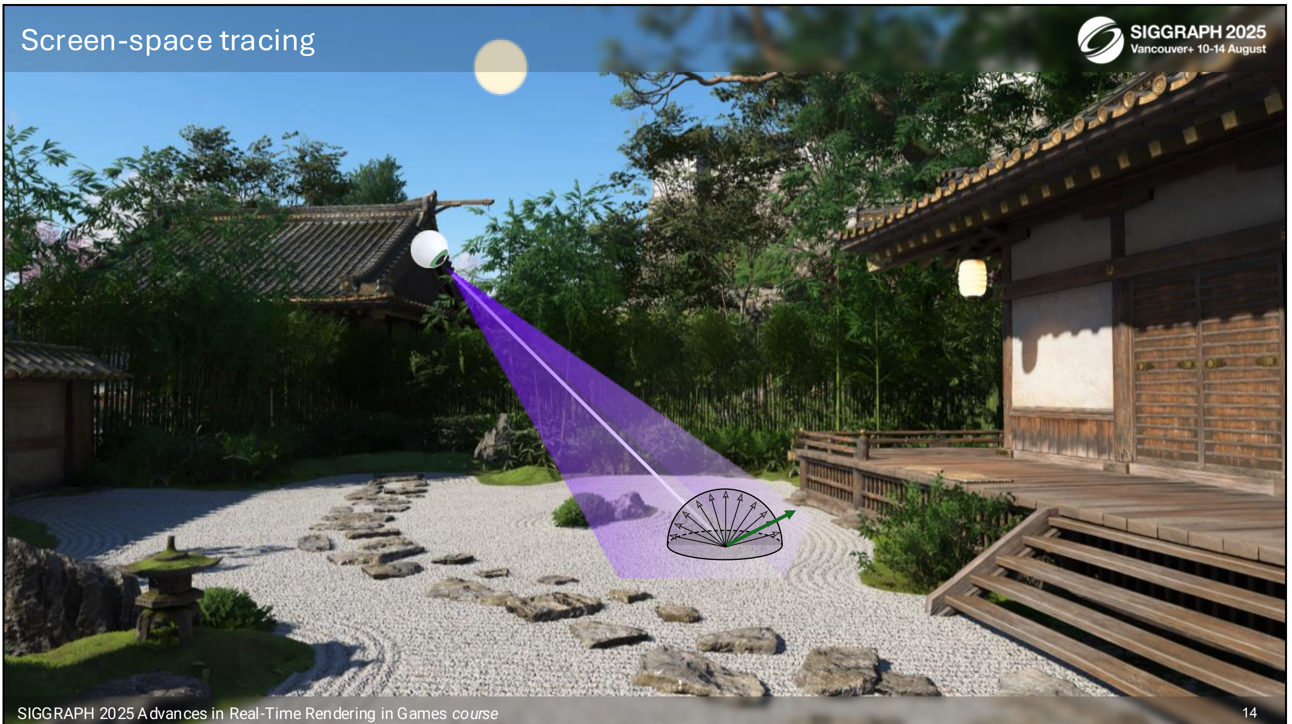
[McGuire 2019] Dynamic Diffuse Global Illumination

The work we are presenting today is based on techniques such as « Dynamic Diffuse Global Ilumination » the lastest version with a per pixel raytracing pass using probe volume as a cache for secondary hit GI.
Similar system were also presented and used in games such as Metro Exodus and games running on the Snowdrop engine, Avatar and Star Wars Outlaws.
We will look at how we have implements such a GI system to achieve the quality and the speed seen in AC Shadows, a large dynamic open world containing among other thing a lots of vegetation and running on multiple platform.



Lets start with a visual summary of the algorithm.
First we rasterize the scene from the point of view of the camera, building a gbuffer and computing the direct lighting.
For each pixel, we want to compute the indirect diffuse and specular illumination.

In order to do so, for each pixel, we select randomly a direction depending on the BRDF. We do two passes, one for diffuse lobe and one for the specular lobe.



Then we cast a ray in the selected direction in the scene.
In our system, we start with screen space ray marching for multiples reasons. First on some platform (consoles) it is faster to raymarch than raytrace.
Also, as you are going to see later, our RT (BVH) world representation is very coarse for speed reason. Not every object is contained in it, and the geometry is simpler than the rasterized version.
To get back missing object and avoid self intersection we used raymarching….

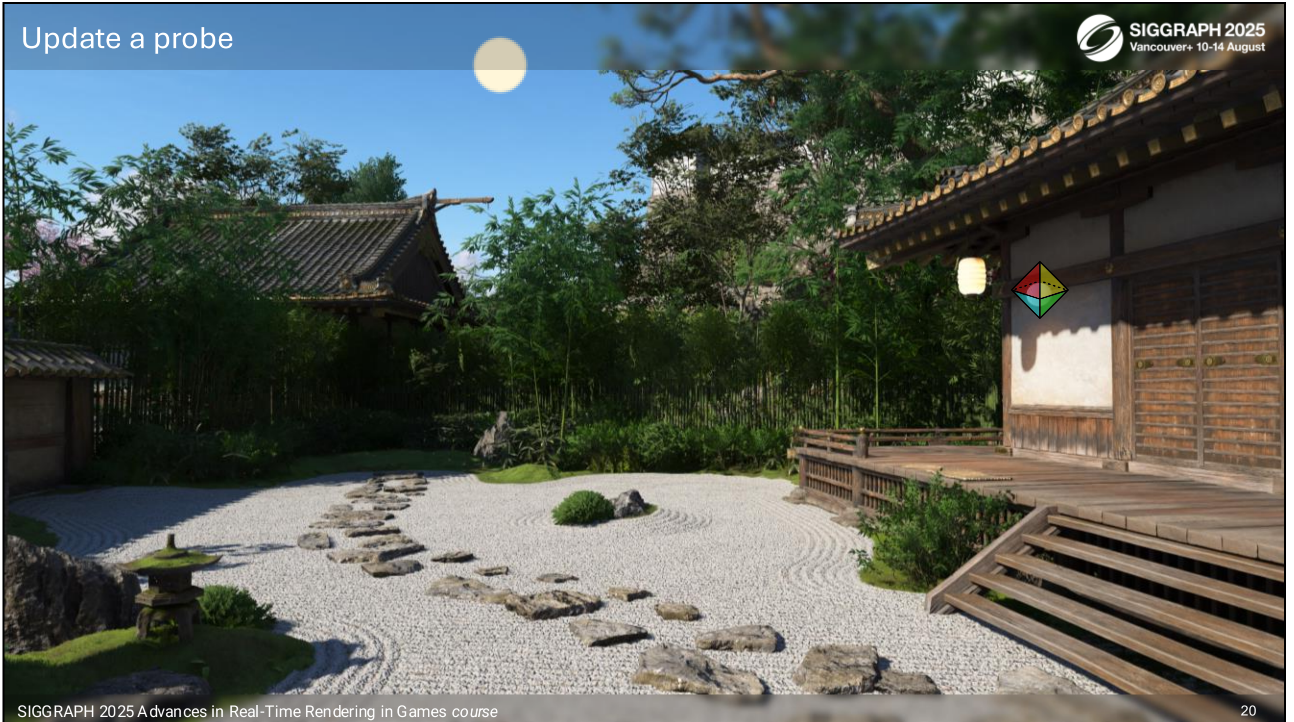
If no hit was found, we then cast the ray into the BHV scene description.

At the hit position (the secondary hit from the camera), we first compute the direct lighting coming from the sun and the local lights.

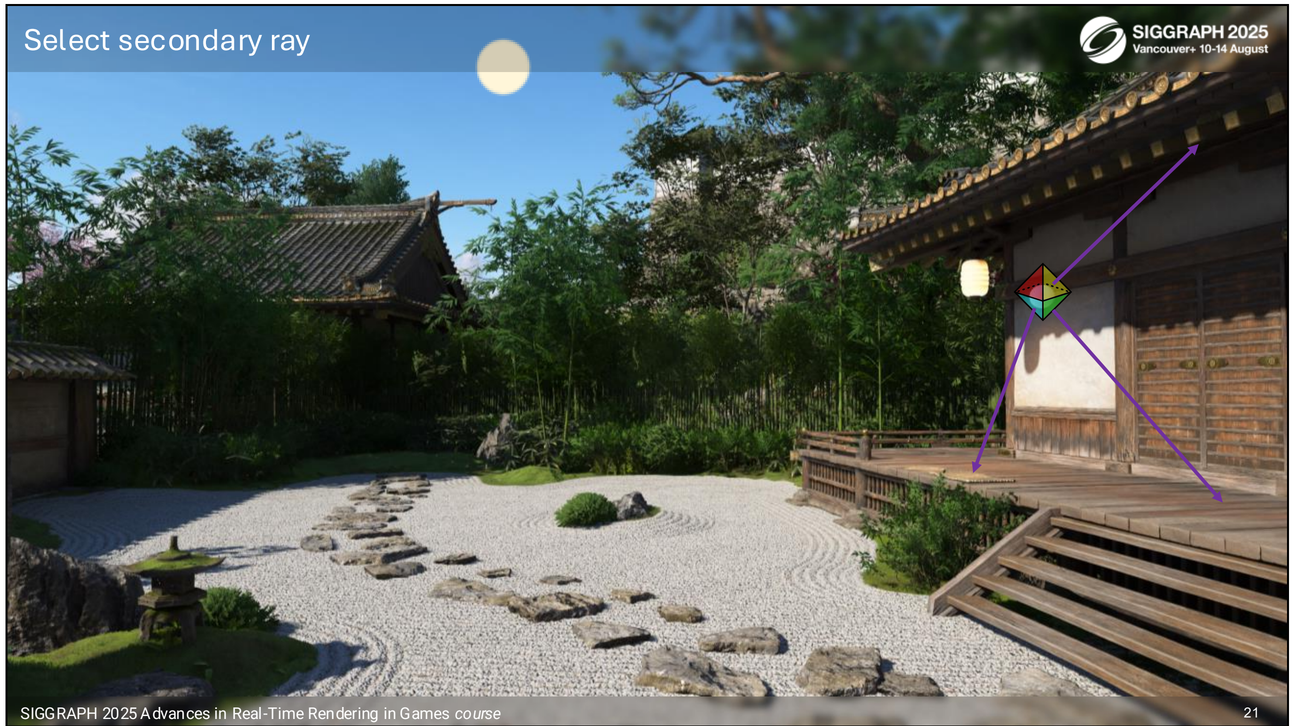
Then, we compute the indirect illumination contribution, only the diffuse part, with the help of a probe volume located all around the camera.

The 8 nearest probes are selected and interpolated to compute the indirect diffuse illumination at the hit position.
After doing so for all pixels, we end up with a noisy solution that we are going to denoise and get our indirect lighting.

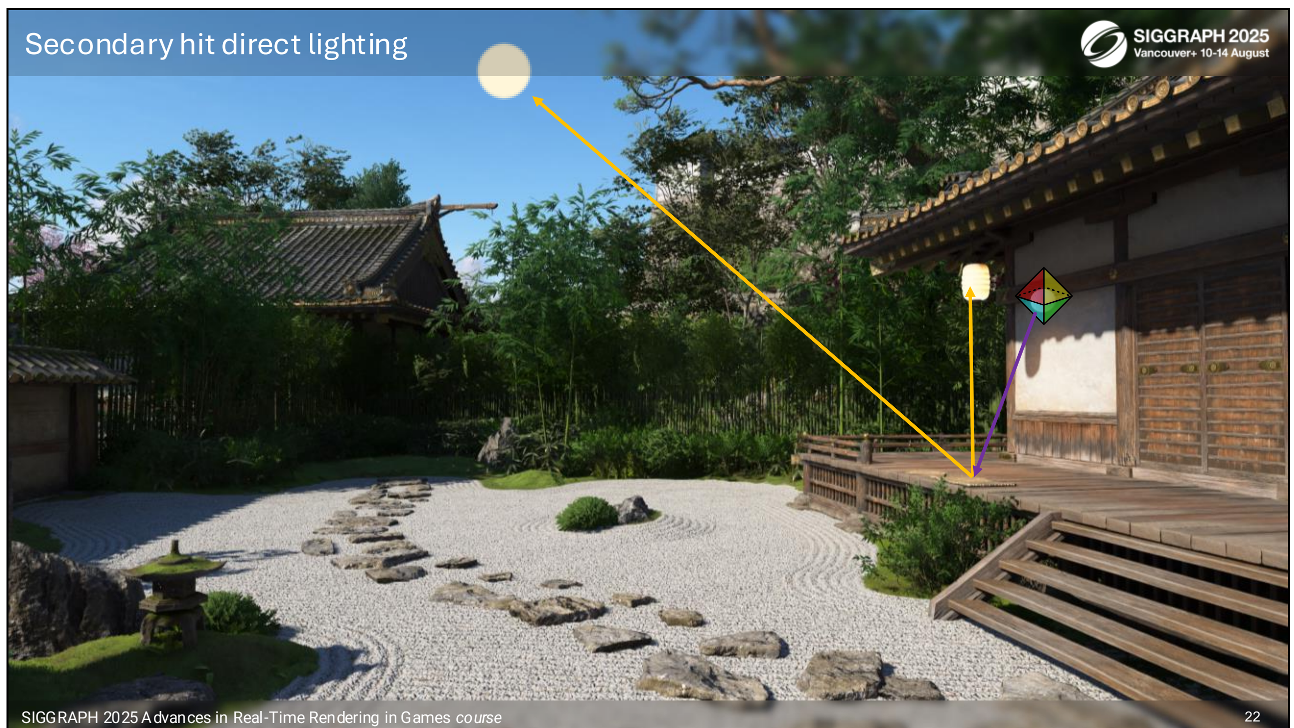
Now, to compute the indirect illumination at the secondary hit we used a probe volume that need to be built beforehand.



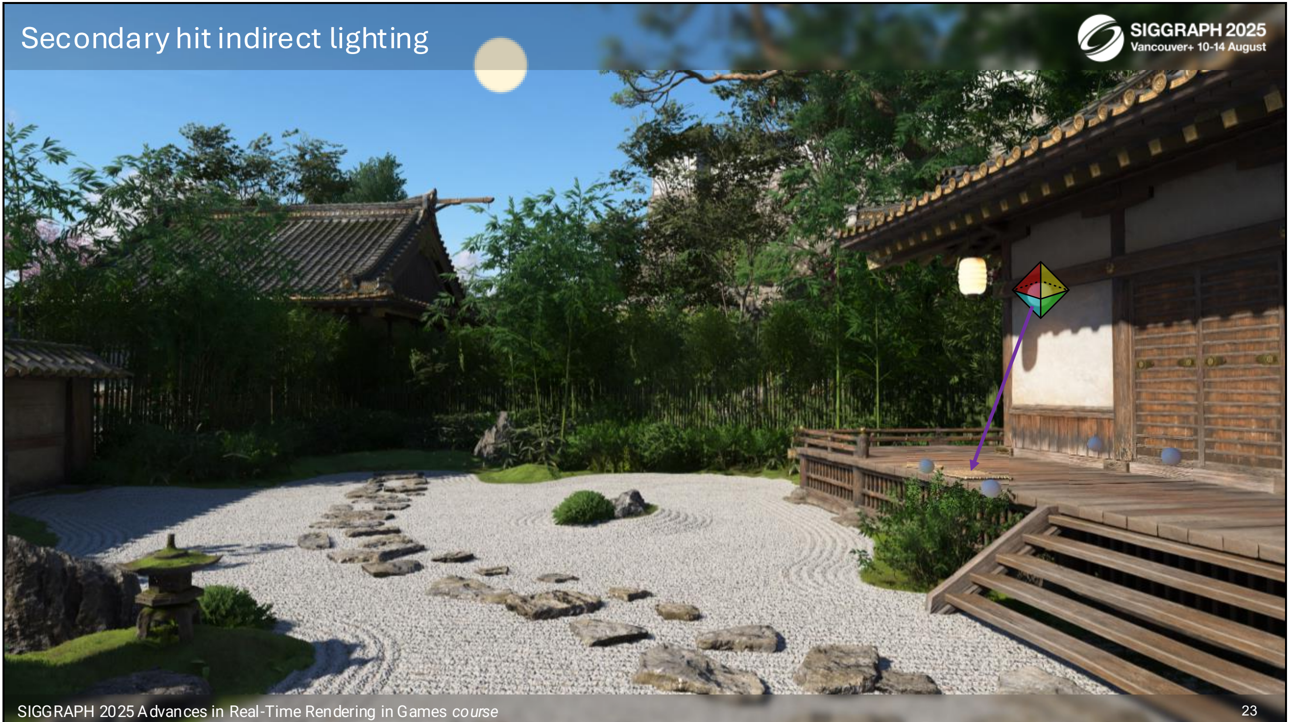
For each probes contained in the volume



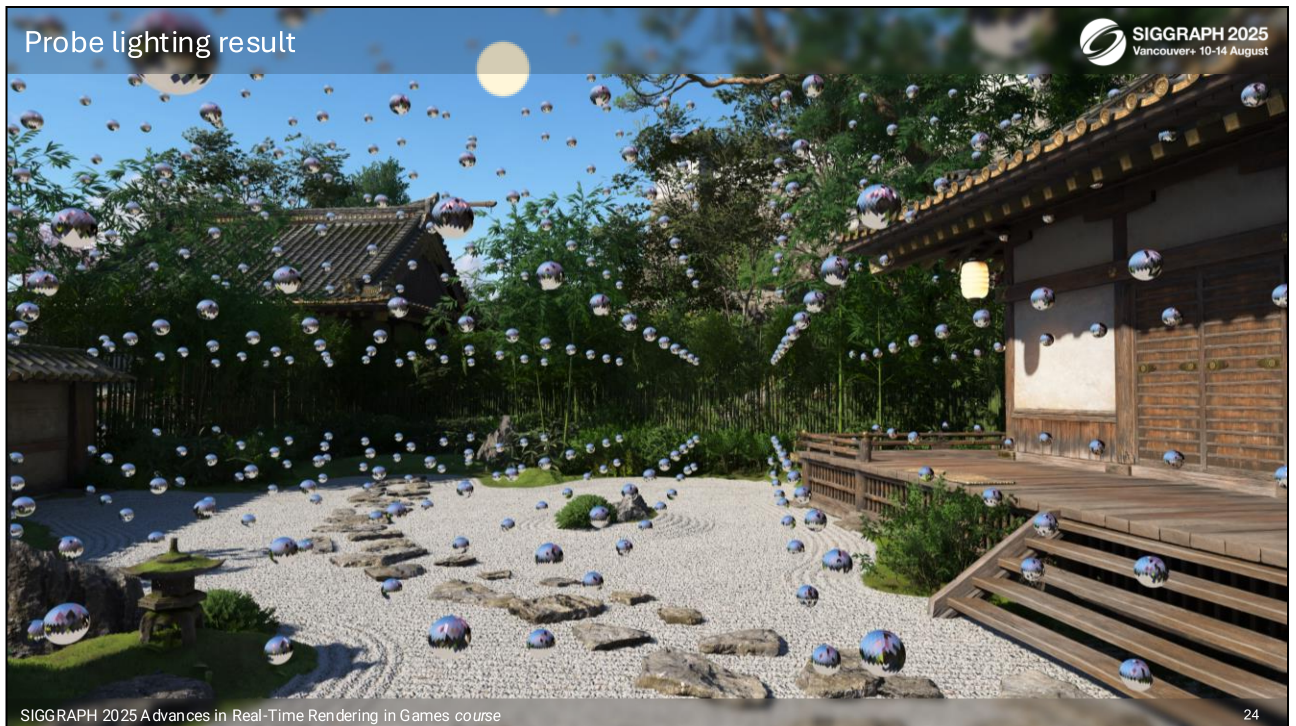
We will cast rays in all direction covering the sphere surface of the probe.



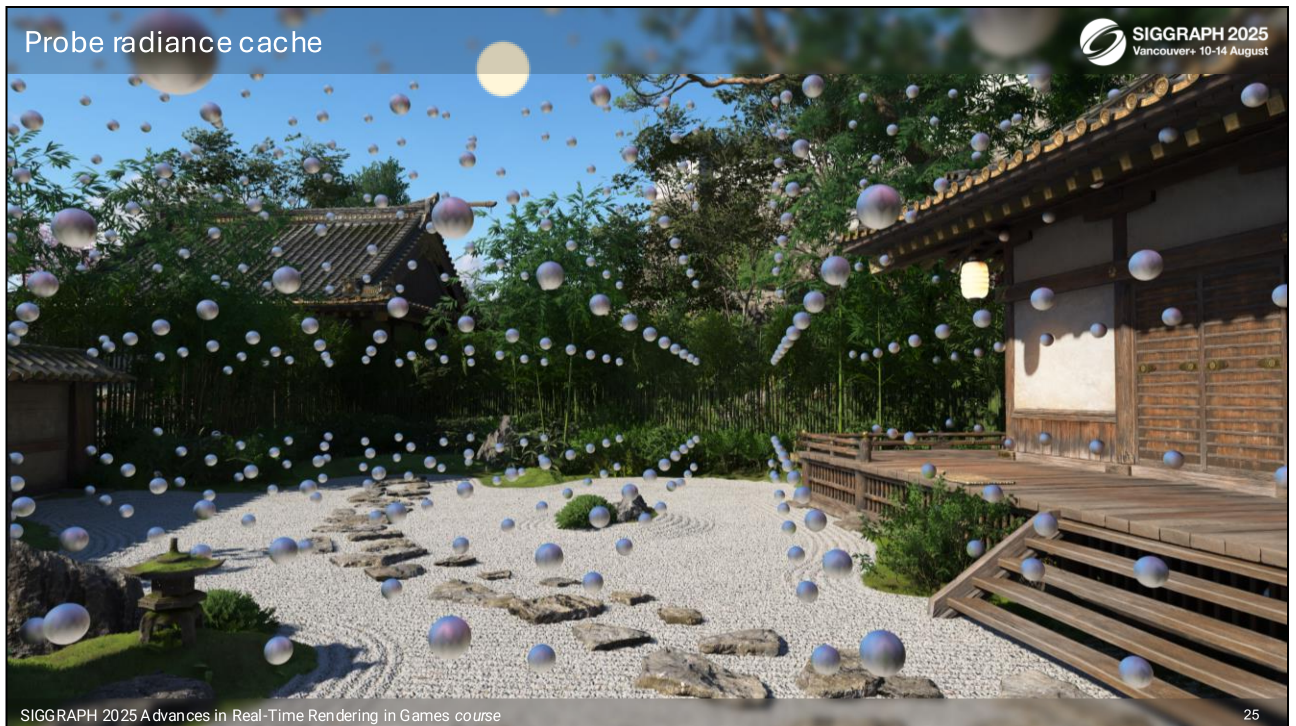
And compute the direct illumination at the hit position, again from all light sources.



The indirect illumination will also be computed using the current probe volume but with values from the previous frame. This will give us multi-bounce global illumination in the probe volume.



This is the radiance result after lighting all probes.



All probes will be convolved to create a filtered radiance cache, trading correctness for reduced noise and a more stable result. Those probes will served as fallback when not geometry is hit.

We also convolve all probes with a diffuse BRDF to produce an irradiace diffuse cache. This is used to compute indirect diffuse illumination at any point in space.

Now that we have finished looking at the overview of the algorithm, lets take a look at the raytracing scene definition.

Ray tracing scene description

SIGGRAPH 2025 Vancouver+ 10-14 August

Geometry

Speed over precision

- Low LOD level
- No vertex animation/skinning
- Cull as much as possible small geometry
- BVH Quality for culling distance

Per triangle data

- Face normal (32bits)
- Vertices UVs (3x32bits)

Terrain loaded dynamically around player

As mentionned previously, our scene description used for ray tracing is a coarse representation of the rasterized version.
We use low LODs as much as possible without incurring visible artefacts.
There is no vertex animation support, so no skinning (no character) or moving vegetation.
We cull as much as possible small geometries such as small vegetations and clutter.

To speed up getting the hit result, we bake per triangle data so that we don't need to first fetch the vertex indices and then the vertex data of each vertex.
We just fetch the triangle data and interpolate its Uvs.

=====
We do one 4x32 bits fetch to get triangle data, containing face normals and uv vertices (3x32 uv (16bits u, 16bits v)).

Materials

Unified representation

Baked texture for complex materials

- Tri-planar

Average material PBR constant attributes

- Baked or artistic driven

Season color LUT

Using the same philosophy for the materials, we used a unified description contrary to our rasterized version where each materials can have its own shader.
This unified description allow us to use inline raytracing which is generally faster on multiple platforms.

For each material we compute PBR constants. That's what is shown in this slide.

====
Accessing material info:

- instanceDataID = ray.CommittedInstanceID + ray.CommittedGeometryIndex
- materialID = instanceData[instanceDataID];

SIGGRAPH 2025
Vancouver+ 10-14 August

Materials

Bindless textures for albedo and alpha

World Space texture atlas for terrain

We also add support for textures in case of albedo and alpha parameters to increase the quality of the scene.

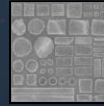
Ray tracing scene description

SIGGRAPH 2025 Vancouver+ 10-14 August

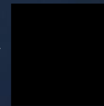
Opaque material baking

- Compute simple texture average
- Use last mip available

Albedo

Roughness


Metalness

Translucency


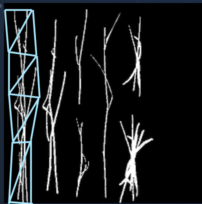
The baking process for materials is simple. For each textures of each PBR parameters we compute the average values.
In case of a complex material, like tri-planar, we first bake the result of the material into texture atlases and then use those textures as input of this process.
We also use those textures directly for the albedo and alpha.

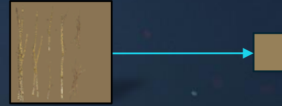
Ray tracing scene description

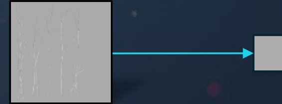
SIGGRAPH 2025 Vancouver+ 10-14 August

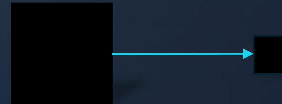
Alpha-test material baking

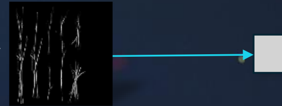
- Compute alpha weighted average
- Compute opacity by rasterizing mesh onto alpha texture











In case of alpha-test materials, we take into account the alpha of each textures when computing the constants parameters. But we also compute an opacity value for each meshes using it. To do so, we reproject each triangles of a mesh into the alpha texture and average only the contained texels. This parameter is used for raytracing.

=========

The opacity is used either for randomly selecting an opaque hit or to rescale the geometry to create a fully opaque mesh.

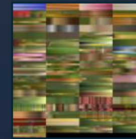
SIGGRAPH 2025
Vancouver+ 10-14 August

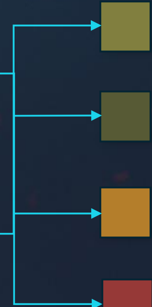
Vegetation material baking

- Global material variation index = season
- Two channels alpha texture
  - Red = leaves fallen (winter)
  - Green = leaves present

Albedo

Season LUT





In case of materials used for vegetation, we added support for color LUT. Different colors are defined according to the season.
We also support alpha texture with two channels, one with and one without leaves selecting the correct channel depending on the season.

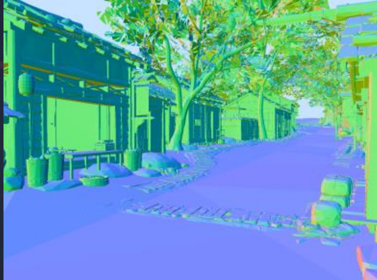
======

- We have 8 seasons values (start and end for each season)

SIGGRAPH 2025 Vancouver+ 10-14 August

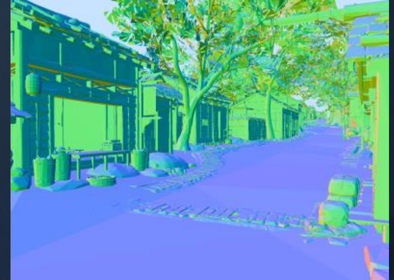
How it's built

Distance to camera



- Add/remove instance
- Update dynamic transform matrix
- Update terrain patch around camera
- Time slice culling

Solid angle



Each frame, we update our BVH representation. We add and remove instances depending on the streaming around the camera.
We update the transform matrix of moving objects. We support dynamic objects, but not vertex animation (as previously mentioned).
We also update the terrain around the camera. The terrain is different from the rest of the geometry. It is split into patches and has its own texture atlases.

Finally, we cull all instances from the point of view of the camera. We start at a fixed distance and remove instances having a smaller solid angle than a threshold.
Since we have a lot of instances, we time slice the culling to make it faster. It takes less than a second to complete the full scene.
================
We rebuilt every frame the TLAS in the async queue

The next thing we do after updating our BVH is updating our probe volume data.

Volume definition

Cascades

- 5 cascades 16x16x8 (10k probes)
- Following camera with forward 3d offset
- Toroidal addressing
- 2 meters – 32 meters spacing

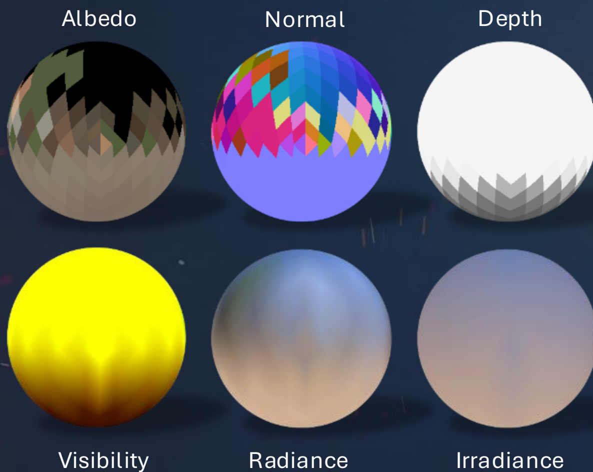
Probes Meta Data

- Worldspace offset
- Sun and sky average luminance
- Local lights average luminance
- Lighting scale
- Flags (indoor, disabled, half resolution)

We use a standard definition of multiple cascaded 3d uniform grid following the camera. In our case, containing a bit more then 10k probes.
Each probe has buffer data, but also some meta data such as world offset allowing probe to move inside its defined region.
=========

- All size are parametrizable for cascade and probes resolutions
- The largest grid covers 512 meters (32*16)
- Each probe have a fixed position inside the texture atlas since we use toroidal addressing (index = (position/probeSize) % numGridElements)

Probe data buffers



- Octahedral encoding in atlas

Buffer	Content	Precision	Resolution
GBuffer	Material (albedo, emissive, roughness, metalness, translucency, ...)	64 bits	20x20 texels
	Normals + sun shadows	32 bits	
	Depth	16 bits	
Visibility	Avg. depth + Avg. depth^2	2 * 8 bits	12x12 texels (10 + 1 border)
Radiance	RGBA	16 bits	12x12 texels (10 + 1 border)
Irradiance	RGBA	16 bits	7x7 texels (5 + 1 border)

Probe data is stored in a texture atlas using octahedral encoding. We keep a compressed gbuffer from which we will compute a temporary buffer containing the lighting (radiance, not shown here).
From this data, we will produce by convolution a visibility, radiance and irradiance probe buffers used as a caching system.
========

- Total memory for probe volume: 73 MB, details in performance slide
- Radiance + irradiance (each probe has its own scale for lighting)
- Keeping the gbuffer is useful to reduce the cost of relighting probes without re-casting rays. In AC Shadows, player can turn on/off lights, so this help reducing the cost of updates.

SIGGRAPH 2025 Vancouver+ 10-14 August

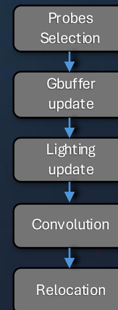
Probe update pipeline

Costly to update all probes
Full updates only in limited scenarios

- Loading scene
- Start of cutscene

Runtime optimization

- Partial update
- Gbuffer caching
- Quarter resolution update (up to 40% speed up in tracing)
- Rescale lighting



Updating all probes every frame is too costly, and we use different optimizations to reduce the cost.
First, we are going to update only a subset of all probes every frame. We also support three type of updates:
-full update, with gbuffer creation, lighting computation, convolution and relocation, at full resolution
-also, full update, but at quarter resolution. With this, we reduce the raytracing cost up to 40%.
-Lighting and convolution only, that will reuse the probe cached gbuffer

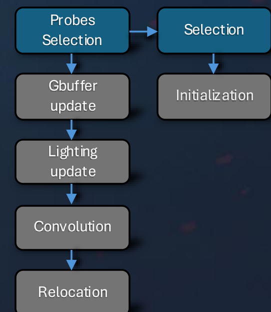
=======

- We support full updates in some special cases, like after loading a region or teleporting to a region.
- Rescaling the lighting of probe not being updated reduce visual artefacts seen when changing rapidly the lighting and also reduce the need to do a full update.

Probe selection

512/1024 probes per frame according to priority

1. New probes from volume position
   - Full update
   - Quarter resolution
   - ~0-50% (generally less than 12.5%)
2. Update probes Gbuffer
   - Full update & full resolution
   - Round robin in each cascade starting from biggest
   - ~16% of the remaining budget
3. Update probes lighting only
   - Update lighting at full resolution
   - Round robin in each cascade from biggest



We start the update by first selecting a subset of probes. In AC Shadow we have a budget of 1k probes or half of that for 60 fps mode or lower performance plateform (Xbox series S).
We use a greedy algorithm in 3 step to select the probes to update.

In the first step, we select newly created probes. Those are generally the probe on the boundary of a 3d grid that move to follow the camera. Since the number of new probes can vary a lot depending on the camera movement,
we are going to compute these probes in quarter resolution to reduce and stabilize the cost of gbuffer updates.

In the second step, we select in each cascade a certain number of probes, approximately 16% of the remaining budget using round robin. Those probes are going to be fully updated at full resolution.

Finally, in the third step, we again select probes in each cascades in a round robin ways using the rest of the budget. Those probes are going to be relight and convolve only. When the camera is not moving too fast, this represent around 80-84% of the budget.

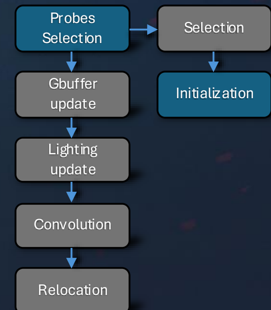
=============

- We can disable a cascade if the camera moves too much (too many probes to

- update)
- Each probe is updated around 3 times per seconds (lighting) and has a full update every 2 seconds
- For the round robin update, each cascade has two indices (one for gbuffer update, one for lighting update) keeping the last updated probe index, so that the next frame the selection can start from the next probe.

Probe selection

- Classify indoor/outdoor
- Test against terrain
  - Under-terrain cave support with indoor flag
  - Move up if terrain is in probe range
  - Deactivate if too deep



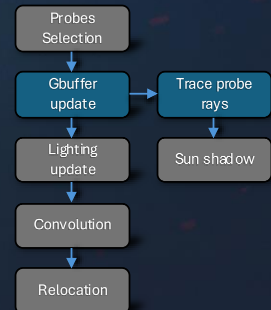
The first step done on all the selected probes is to update some of their meta data. We are going to classify the probes to be indoor or outdoor. For probes under the terrain, those just under will be moved above and the rest will be disabled.

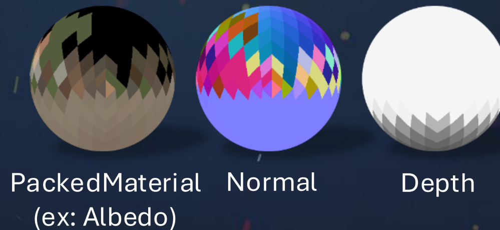
=====

- Done on all updated probes
- Probe under terrain but contained inside an interior volume won't be disabled. Those are either basement or cave.

Gbuffer update

- Full and quarter resolution
- Trace center of texel
- Maximum distance based on cascade
- Sample ray tracing material for closest hit
- Keep back facing count
  - Disable probe if too many





PackedMaterial (ex: Albedo) Normal Depth

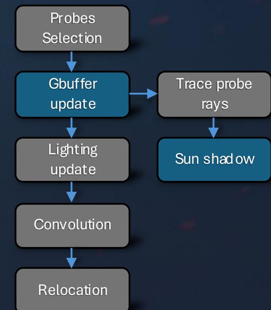
Then for probes selected for full update, either full or quarter resolution, we will generate their gbuffer by tracing rays at the center of their texel. There is no jittering nor temporal accumulation, This does not work well with the gbuffer caching. The maximum distance of the rays will depend on the size of their cascade. The bigger the cascade the longer the range will be. At this stage, we keep track of the number back-facing triangle hit. If we are not able to relocate the probe if necessary and it contained too many back-faces, we will disable the probe.

SIGGRAPH 2025 Vancouver+ 10-14 August

Gbuffer update

Compute sun shadow since it moves slowly

- Test secondary shadow map
- Shadow ray cast if outside
- Add cloud shadows

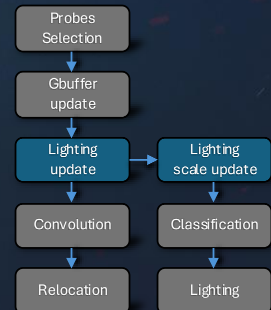


As we are updating the gbuffer, we are going to compute the sun shadow. It is not totally precise, as we can do multiple lighting update before updating the gbuffer and the sun position can move during that time.
With our probe selection algorithm, probes are fully updated around each 2 seconds, the approximation is good enough since normally the sun does not move that much in that time period.
For shadow computation we first try to use a shadow map positioned around the camera, and if the hit position is outside, we will cast a shadow ray if the face is oriented toward the sun. We will combine the result with cloud shadows.

Lighting update

To keep non-selected probes balanced in time-of-day transitions

- Compute new sky and sun lighting estimate
- Compute previous and new average luminance estimate
- Update lighting scale




```
float3 estimatedSunAndSkyLighting   = (sunLight.LightDir.z + 1) * 0.5 * sunLight.LightColor * C_MIDDLE_GREY * (1 - metadata.SkyVisDC);
estimatedSunAndSkyLighting          += EvaluateSkyLightingSH(probeWSPosition, float3(0, 0, 1));
float estimatedSunAndSkyAvgLuminance = ColorToLuminance(estimatedSunAndSkyLighting);

float currEstimatedAvgLuminance = estimatedSunAndSkyAvgLuminance * metadata.SkyVisDC + metadata.LocalLightsAvgLuminance;
float prevEstimatedAvgLuminance = metadata.SunAndSkyAvgLuminance * metadata.SkyVisDC + metadata.LocalLightsAvgLuminance;

LightingInvScale *= currEstimatedAvgLuminance / prevEstimatedAvgLuminance;
```


To support the case of fast lighting change from the sun. The first step of the lighting computation is to rescale the radiance and irradiance of all probes that have not been selected. This is done by estimating the new average illumination of the probe looking at the new sun position and sky intensity and comparing it against previous estimate. Only the lighting scale of the probe is updated and not all texels of the cache, as we are encoding each probe in their own lighting scale. This is done as an optimization, but also to increase the precision.

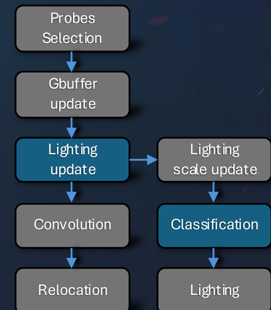
====

The computation of this estimate use meta data information for the probe such as sky visibility, average sun and sky lighting and average local lighting.

Lighting update

Prepare texel lists for compute dispatchs

- Split by hit/miss result
- Encode atlas pixel coordinate in 32 bits: $x|(y << 16)$
- Reduce cost by ~20%



After that step, we are going to compute the lighting for all selected probes, the ones we just traced and also those we did not, using the cached gbuffer.
We split the computation in two passes, one for texel that have hit geometry, and one for the miss. Doing this reduce the lighting cost by 20%, considering the classification pass.

=========

- Look at hit distance to determine hit/miss. Miss is encoded as a special distance value (MISS_DISTANCE)

Probe volume

SIGGRAPH 2025 Vancouver+ 10-14 August

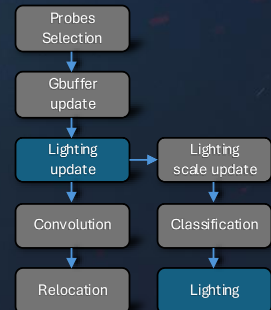
Lighting update

Prepare texel lists for compute dispatchs

- Split by hit/miss result
- Encode atlas pixel coordinate in 32 bits: $x|(y << 16)$

Hit

- Direct: Sun + local lights
- Indirect: irradiance probes (recursive multi-bounce)



In the first pass, we compute the direct lighting from all lights and also the indirect diffuse illumination using the probe volume irradiance cache from the previous frame.

Lighting update

Prepare texel lists for compute dispatchs

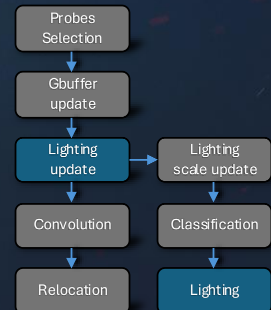
- Split by hit/miss result
- Encode atlas pixel coordinate in 32 bits: $x|(y << 16)$

Hit

- Direct: Sun + local lights
- Indirect: irradiance probes (recursive multi-bounce)

Miss

- Radiance probe
- Sky radiance



In the second pass, for the texels that did not hit any geometry, we are first going to try to use the probe radiance cache if the ray end position is contained in the probe volume.
Otherwise, If the position fall outside the probe volume, a sky approximation will be used.

====================
The sky approximation is a small 8x8 texture update every frame.

SIGGRAPH 2025 Vancouver+ 10-14 August

Lighting update

Prepare texel lists for compute dispatchs

- Split by hit/miss result
- Encode atlas pixel coordinate in 32 bits: $x|(y << 16)$

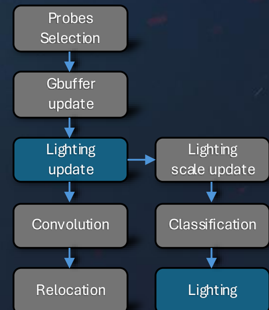
Hit

- Direct: Sun + local lights
- Indirect: irradiance probes (recursive multi-bounce)

Miss

- Radiance probe
- Sky radiance

Then, output radiance + local light luminance



At the end, two outputs will be kept in the buffer. The total radiance, but also the local light luminance. This will be used to help rescaling the probe lighting.

SIGGRAPH 2025 Vancouver+ 10-14 August

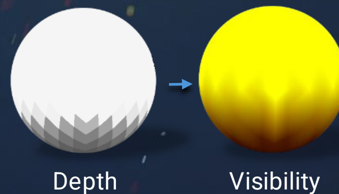
Convolution

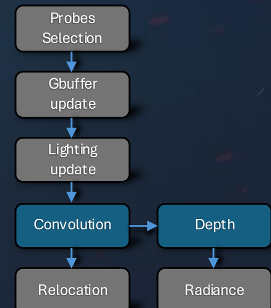
Variance shadow map
- Average depth
- Average depth^2

Rescale depth and clamp to probe voxel region

$$depth = \min\left(\frac{depth}{(probeSpacing * \sqrt{3})}, 1\right)$$

Output result with duplicate border





The next step is to convolve all selected probes. We first compute a variance shadow map from the depth buffer. This will be used to compute weight for probes when doing interpolation. One interesting point is that we need rescale and clamp values between [0, 1] since we are encoding it in 8 bits. But we do it before computing the average and average^2 value. This help reduce leaks in some cases (interior corner).

=========

- While writing the output, texels on the border are replicated. To create a safe border for interpolation. A 5x5 probe become a 7x7. See extra slide for the code.

Probe volume

SIGGRAPH 2025 Vancouver+ 10-14 August

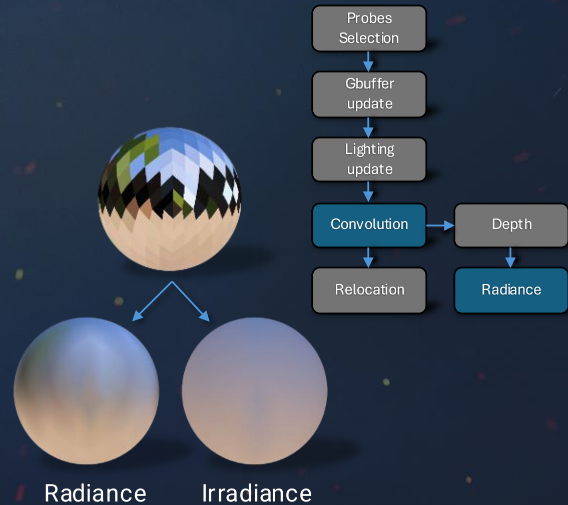
Convolution

Compute

- Sky visibility
- Average local lights luminance
- Average luminance
- Filtered radiance and irradiance

Lighting scale $= \frac{1}{avgLuminance}$

- Scale radiance and irradiance
- Use to adjust non-rendered probes



We also convolve the radiance result to produce a filtered radiance and an irradiance probe. During that process, we compute multiple meta data such as the sky visibility, the average local lights luminance, and the average luminance that will be used as the lighting scaling factor.

=============

- Scaling radiance help keep precision anywhere (outside in sun, dark cave or interior) and any time (day and night). From one extreme to the other the exposure can go from around -16 to +16.

Convolution

Dispatch

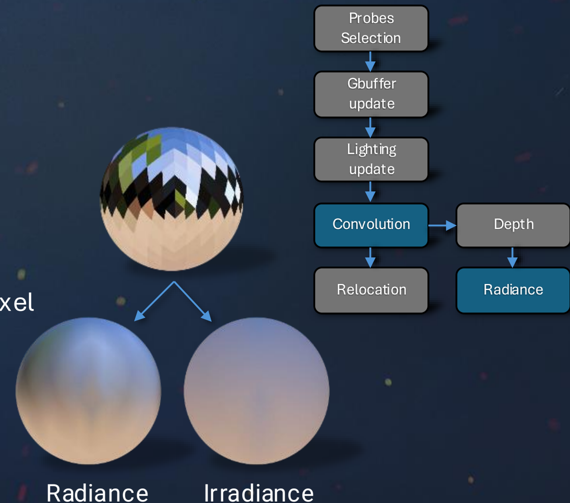
- (radiance + irradiance pixels)/64,1, probes
- numthread(64,1,1)

Load data in LDS

- Use bilinear filtering to reduce 4x samples numbers
- Decrease convolution time by 4
- Batch loading, then processing by 64 samples

Each thread compute full convolution for output texel

- Cosine function for irradiance
- Small lobe for radiance



To optimize the compute shader used for doing the convolution, where each thread computes the full convolution of the input probe for an output texel, we first load the radiance data into the LDS. While doing so, we use bilinear filtering to reduce by 4 the number of samples to read and convolve. This give us a 4x speed up in exchange of a small decrease in quality, since we are not using the correct weight for the texels.

==========

- In the compute shader, each thread group is working on a different probe doing at the same time the radiance and irradiance convolution. Each thread is doing one pixel of the output, either for the radiance or the irradiance probe.
- Each thread of the thread group first participate into loading the radiance into the LDS (local data store). This is done by batch of 64 texels. Then each thread convolve the 64 texels contained into the LDS with their respective BRDF (for their direction). And this continue until all texel of the input probe as been processed.
- In AC Shadows we used 10x10 radiance cache and 5x5 irradiance cache. This give us 100+25=125 threads for each probe. We round this to the next multiple of 64 that is 128.
- For radiance filtering we use a clamped cosine: pow(saturate((cos-0.75)/(1-0.75)), 4)

Relocation

- Done on probes with full update
- Compute min/max distance
- Move probe outside or away of geometry
- Disable probe if not possible

The last step of the probe volume update is to move probes that are inside geometry (seeing mostly back-faces) or too near geometry in order to have a better radiance representation. The probe will be disabled if we are not able to move it outside of geometry.

================

- A probe is considered inside geometry if the number of back-face triangle hit is above a threshold.
- To find a better position, the whole probe gbuffer depth is looked at. If the minimum depth is too small the probe is moved in the direction of the maximum depth.

Now that we have computed and updated our probe volume cache, we can compute our per pixel indirect illumination. We will look first at the diffuse indirect illumination pass.

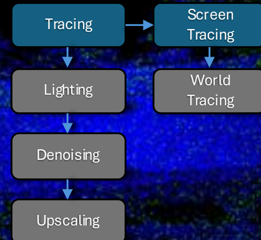
Per-pixel indirect diffuse algorithm

Screen tracing

1 ray per pixel at quarter resolution
Cosine distribution
Trace cascade probe range * 2

Screen ray march until hit or outside
- Valid if ray z – z buffer < threshold
    - Add pixel to hit result

Add pixel to miss result if end of ray



For the diffuse lobe, we are going to cast a ray for each pixel in quarter resolution. We support also full resolution but the increase in quality is not enough to justify the 4x difference in cost.

The ray direction, is randomly chosen from a cosine distribution and the maximum distance depends on the ray origin position. The farther the position is from the camera, the farther the ray will be cast.

We start by raymarching in screen space and if it hit a valid geometry, we add the pixel to the hit list and we keep the hit position. Those are the green pixels on screen. If we travel the maximum distance without hitting anything, then we add the pixel to the miss list. Those are the blue pixels. In all other cases, we are going to cast the ray into the BVH. In this image it represent 49% of the pixels.

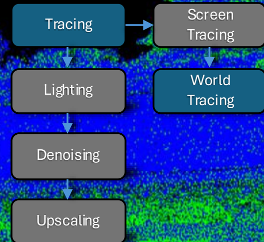
==========
- The cases that needs to cast into the BVH are:
    - ray falling outside of screen before going its max distance.
    - Ray hitting a geometry on screen, but the ray position is too far behind the depth buffer value. In this case we backtrack to the last visible position on screen before casting into the BVH.
    - Material is translucent and we are casting the ray behind the screen geometry
- 13% skipped (87% screen RT), 38% Screen RT only, 49% World RT

Per-pixel indirect diffuse algorithm

SIGGRAPH 2025 Vancouver+ 10-14 August

World tracing

- Only if no screen hit
- Backtrack to last position
- Trace ray in BVH
- Add pixel to hit or miss list
- Output packed material, normal, hit distance into a screen GBuffer



For rays that needs to be cast into the BVH, we use the last valid position of the raymarching as start position.
On hitting geometry, we add it to the hit list. Again, those are the green pixels on screen. We build a gbuffer containing the needed information for the lighting: the material description and the hit position.

Again, if we don't hit any geometry, we are adding the pixel to the miss list. The blue pixels.

============
We also use a minimum start distance to reduce self intersection cases. We use t = 0.05 m.

We only keep the hit distance and not the hit position since we can regenerate it by generating the same ray direction using the same seed.

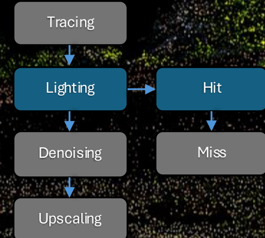
Per-pixel indirect diffuse algorithm

SIGGRAPH 2025 Vancouver+ 10-14 August

Hit lighting

Direct: sun + local lights
- Simplified shading
- Shadow: cascade, secondary, ray or fully shadowed

Indirect: irradiance probe
- 8 probes with weights



After the ray casting is done, we can compute the lighting from the indirect gbuffer. Same as for the probe volume, we do it in two passes. We start for the hit case where we compute the direct lighting for all the lights. Where it differs a bit from the probes is how we handle sun shadow.
We start by looking at the cascade shadow maps if it is valid. If not, we look if we can used the secondary shadow maps. And if the hit position fall outside its range, then we are assuming the result is fully in shadow.
We also compute the indirect lighting by using the irradiance probe volume.

========

- The CSM is considered valid only for hit position visible on screen. For hits contained in frustum but hidden on screen, the shadow value can be wrong since the CSM is optimized for speed and does not contain all occluders.
- When outside shadow map, better full shadow than full light for leaks

Per-pixel indirect diffuse algorithm

Miss lighting

Try radiance probes
- Similar interpolation as irradiance

Sky approximation
- If outside probe volume

We then do the lighting computation for the pixels that did not hit any geometry. Exactly as we did for the probes, we first try to used the filtered radiance cache if the end of ray position is contained inside the volume definition. If not, we fallback to the sky approximation texture.

The shown image is the final lighting for a frame.

Per-pixel indirect diffuse algorithm

SIGGRAPH 2025 Vancouver+ 10-14 August

Lighting

Scale radiance with last frame's exposure

Encode in spherical harmonic

- YCoCg (order 1 for Y, order 0 for CoCg)
- 64 bits (6*8bits + 16 magnitude)

Output hit distance

- 16 bits
- Use for RTAO and denoising (reprojection + filtering size)

Contrary to the probe volume, we are not going to output only the radiance. We are going to encode it in spherical harmonics. That way, we are going to reconstruct the full spherical radiance per pixel during the denoising.

We also output the hit distance that will also be denoised to produce RTAO.

=====

- The radiance will be scaled by the last frame exposure to increase the precision.
- Order 0 is constant 1 channel, order 1 is linear 3 channel giving a total of 6 channel to encode in YCoCg. Those 6 channels are divided by the maximum magnitude and encoded in 8 bit. The maximum magnitude is kept in 16 bits giving us a total of 64 bits encoding.

SIGGRAPH 2025 Vancouver+ 10-14 August

Denoising

Modular Snowdrop Denoiser (MSD)

- Spatial filtering
- Temporal filtering

Similar to NVIDIA's NRD

After the lighting is done, we need to run a denoiser to reconstruct the full indirect lighting. I am not going into details here since it could be a talk on its own.

We use a modified version of the Snowdrop engine's denoiser based on Nvidia denoiser. We will talk a bit later about some of those modifications.

This denoiser is split in multiple passes doing spatial and temporal filtering

Per-pixel indirect diffuse algorithm

SIGGRAPH 2025
Vancouver+ 10-14 August

Upscaling

Bilateral upscaling
- Depth
- Normal + material

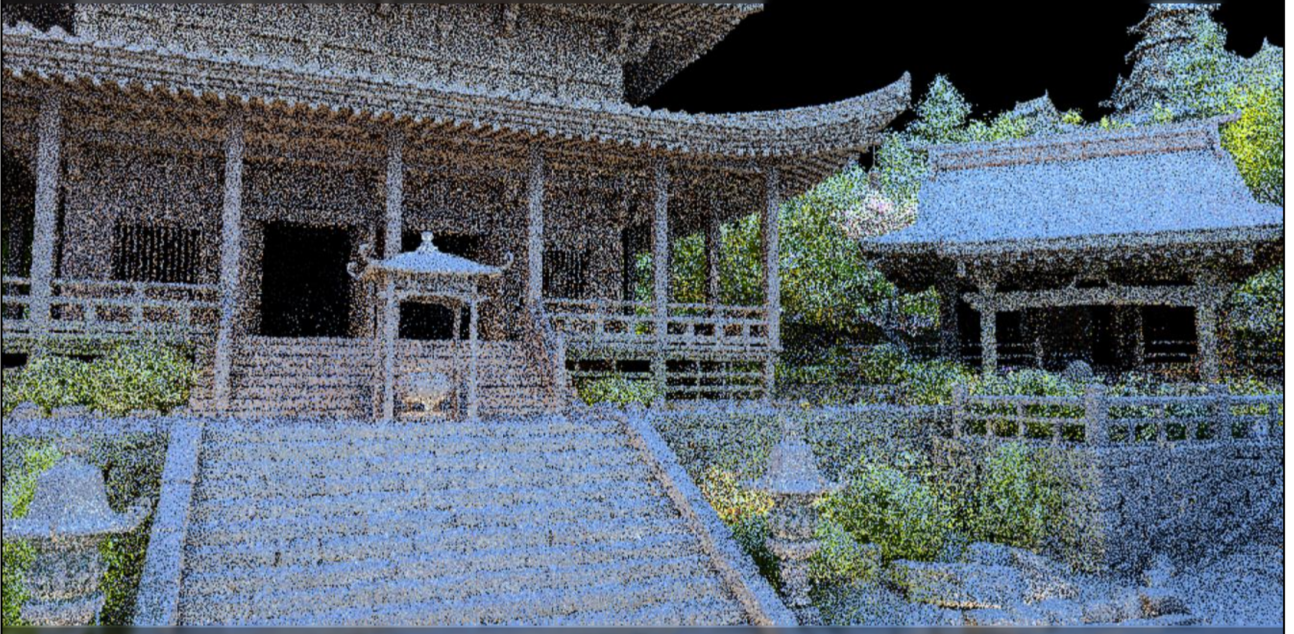
Spherical harmonics (SH)
- Evaluate irradiance per pixel with hi-res normal

Tracing
Lighting
Denoising
Upscaling

Finally, we upscale our quarter resolution result to full resolution.
We upscale spherical harmonics that we evaluate per pixel at full resolution with high resolution normal.

We use a standard bilateral upscale.
=====

- float4 denoisingDepth = GatherDepth(uv);
- float4 depthWeights = 1.f / (0.1 + abs(denoisingDepth - fullResDepth));
- weights *= depthWeights * depthWeights;



I’m going to show you the level of details we get when using SH.
First, here is the input radiance.

And this is the filtered irradiance result.

Now, with SH, we have much more details and better directionality of the lighting

Now, let's look at the indirect specular illumination.

Per-pixel indirect specular

SIGGRAPH 2025 Vancouver+ 10-14 August

Timeline development

- Post-launch plan included
  - Two months to implement
  - One month for QA
- Focused on visual issues
- Close to no optimization time
- Algorithm similar to per-pixel diffuse
- Shipped in half resolution

First, a word on the context of the development. This feature was first planned to be added after the release of AC Shadows but with the announcement of the delay, it was decided to add it before the launch. A total of 3 months was allocated for the complete development, with our first iteration done after three weeks.
As such, the focus was more towards making it work and looking good than to fully optimize it (due to the time constraint). Also, no modification on the data side could be done, we had to make it work without any changes.

The algorithm is very similar to the diffuse pass but with added specialization for specular lobe and changes to increase the visual quality, mostly due to the simplification and optimization done for the diffuse computation that did not work well for highly reflective surface.

SIGGRAPH 2025 Vancouver+ 10-14 August

Tracing

Ray direction

- BRDF Importance sampling: GGX visible normal distribution function
- Water filtering (roughness and normal)

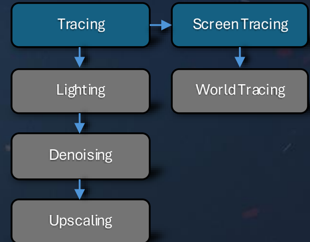
Ray length

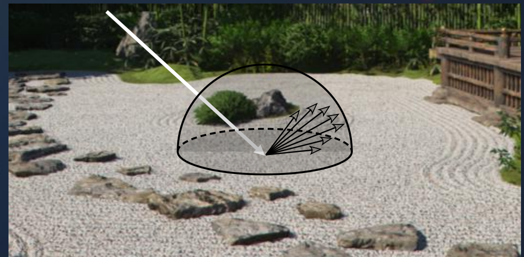
- Depends on roughness: $maxDist\ ((1 - roughness)^2\ +\ 0.1)$

Ray skipping

- Replace by diffuse SH evaluation when possible
- Roughness > threshold
- Dot(RayDirection, ReflectionDirection) < threshold

More precise ray marching





The per pixel pass supports quarter, half and full resolution. We end up shipping using half resolution that have the better quality performance ratio.
First, the ray direction is generated by using importance sampling on a GGX visible normal distribution function. Then, the ray length is adjusted depending on the surface roughness. The more mirror like the surface is, the farther we need to cast the ray.

We optimized this pass by not casting and lighting rays when possible. We already have computed a per-pixel spherical radiance approximation encoded in SH during the diffuse pass. We reuse this when possible by directly evaluating the SH for the generated ray direction.
We do it for high-roughness surfaces and rays that are far from the mirror specular direction since their contribution are going to be low. This optimization give us 20%-30% in reduction on the casting and lighting side.

=================================
Half resolution is half in width and full in height.
We don’t do diffuse evaluation of water surface but we do for specular and adapt the roughness and normal to reduce aliasing (take into account world space projection size of pixel (z +1 in x & y)
Raymarching: linear step + binary search + secant

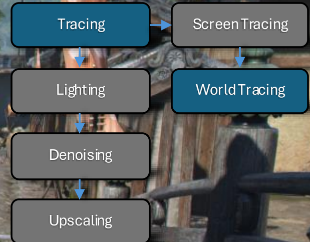
The opaque approximation that we use for alpha-tested vegetation works great for computing the diffuse illumination but is very visible when used to compute strong reflective surface as you can see in this image.

=============

Some detail about this opaque approximation are given in another section.

Tracing

- Removed opaque approximations
- Fallback to alpha-test with a max of 4 any-hit tests



We had to switch to the evaluation of the alpha-test texture limiting to 4 any hit shader call.

===========================

Since diffuse and specular tracing share the same BVH and the opaque approximation generate a different geometry, we had to switch to a different approximation for diffuse ray casting when enabling specular evaluation.

Per-pixel indirect specular

Tracing

Reproject hit on screen and use gbuffer material when possible

Tracing → Screen Tracing
Screen Tracing → World Tracing
Tracing → Lighting → Denoising → Upscaling

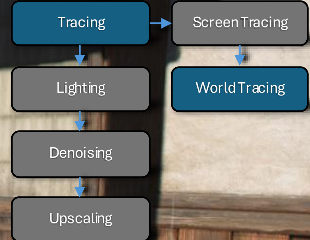
We also see the BVH material approximation in highly reflective surface, but it is not visible with high roughness or when computing diffuse illumination.

Per-pixel indirect specular

SIGGRAPH 2025 Vancouver+ 10-14 August

Tracing

Reproject hit on screen and use gbuffer material when possible



To increase the quality, we always reproject the hit point on screen to see if we can get a better material definition from the gbuffer.

========================
 But it need to not be too different, else the difference will be too much eye catching when going from one to the other.
We also cast shadow ray, if necessary, on high end platform.

Per-pixel indirect specular

SIGGRAPH 2025 Vancouver+ 10-14 August

Secondary hit lighting

Direct Lighting

- Add specular lighting to evaluation
- Use shadow ray evaluation when necessary for sun

On the shading side, we increase the quality by doing the complete evaluation, adding the specular part. We also used the result of shadow ray for the sun shadow evaluation on higher end PC platform.

======================

- We don't reproject the lighting (only the material) to reduce difference for on screen hit or not. The specular is also incorrect when reusing last frame result (in some case the difference is big).

Lighting on miss

Indirect lighting

- Try using only the nearest radiance probe first. Results in faster evaluation due to less probes fetched.

Sky approximation and missing objects

- Use far cubemap

Around half the cost of the lighting evaluation is the probe interpolation. It needs to fetch, compute weights and interpolate 8 probes. An optimization that can be done is to only use the nearest probe at the requested position if the weight is high. Otherwise, we fallback to the standard interpolation. Doing so, more then halves the interpolation time without any visual difference.

On the visual side, again, on highly reflective surface such as water, we can see more easily the limits of some of our approximations. We can see missing far geometries not contained in our BVH.
To get back some of those missing data, we replaced the sky texture approximation by a dynamic fallback cubemap containing the terrain, sky, clouds and far entities.

=======================
This cubemap is 128x128 and update one face per frame.

Per-pixel indirect specular

SIGGRAPH 2025 Vancouver+ 10-14 August

Lighting on miss

Indirect lighting

- Try using only the nearest radiance probe first. Results in faster evaluation due to less probes fetched.

Sky approximation and missing objects

- Use far cubemap
- Adjust the screen space tracing for water

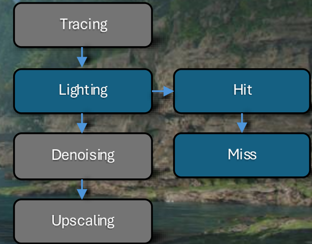
We can see the result here with the mountain and trees in the reflection.
Also, the screen space tracing was adjusted to get more far object not contained in the BVH.

===================

- reprojection used: float3 warpedDirection = normalize((rayOrigin + rayDirection * fakeHitDistance) - cubemapPosition);
- For specular reflection, the screen space tracing is reduced in length to reduce visible artefacts. (This reduce the step size). But on water surface for ray, starting near the end of the BVH definition, the length of the ray marching limit is increased to get back more far geometries.

Volumetric lighting

- Evaluate fog
- Simplified version with only one step integration
- Don’t evaluate near fog



When doing the indirect diffuse computation, the fog evaluation is skipped as an optimization. We add it back for the specular pass but only a simplified version. Here is the different without on the left and with it on the right. The horizon is much smoother taking fog into account.

Lighting

Use radiance demodulation

- See NRD documentation

Better denoising results on grazing angle

Current result

- $radiance$
- $Denoise(radianceDeMod) * preIntegratedBRDF$

The output radiance is much more noisy and difficult to filter compared to the diffuse pass. We have implemented different techniques to reduce this. The first thing we did is to use demodulation of the BRDF as explained in the Nvidia NRD documentation.
In order to do so, we can filter the radiance without computing the surface BRDF and add it after denoising (split sum approximation). This is what we normally do when using SSR and local cubemap.
We can see the result in this image and notice some error at grazing angle where the lighting is too bright.

Per-pixel indirect specular

SIGGRAPH 2025 Vancouver+ 10-14 August

Lighting

Use radiance demodulation

- See NRD documentation

Better denoising results on grazing angle

Current result

- $radiance * \frac{BRDF}{preIntegratedBRDF}$
- $Denoise(radianceDeMod) * preIntegratedBRDF$

A better way to do it, is to compute the full BRDF (and its PDF) while computing the radiance but to also dividing the result by the pre integrated BRDF before the denoising. And then, multiply back by the pre integrated BRDF. This is what is shown in this image, and we can see that we have better result since the BRDF is correctly applied.

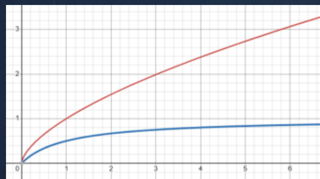
===========

- Demodulation: see nvidia nrd: https://github.com/NVIDIA-RTX/NRD/blob/master/README.md
- If you apply correctly brdf/pdf it become noisier, so demodulation reduce the noise from the brdf evaluation

SIGGRAPH 2025
Vancouver+ 10-14 August

Lighting

- Radiance luminance compression
  - $radiance * pow(\max(radiance.x, radiance.y, radiance.z), expn - 1.0)$
  - Inverse: $radiance * pow(\max(radiance.x, radiance.y, radiance.z), 1.0/expn - 1.0)$



$y = x^{0.625}$
$y = x/(1+x)$

We also compressed the space in which we do the denoising. It is not physically based but some cases, mainly interior scenes reflecting bright exterior, can be really noisy. We use a power function to do so.

==========

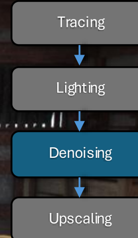
- compressedRadiance = radiance * pow(maxRadiance, exp - 1.0); // exp = 0.625
- Inverse = compradiance * pow(maxCompRadiance, 1.0/exp – 1.0);
- Often used is the Reinhard function x/(x+1) but the compression of the high value was reducing too much the reflection.

Per-pixel indirect specular

SIGGRAPH 2025
Vancouver+ 10-14 August

Denoising

- New pass specialized for specular
- Different reprojection
- Firefly removal



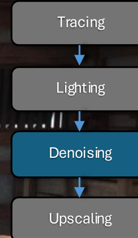
Finally, we specialized the denoiser to better handle specular BRDF and add firefly removal during one of the spatial filtering pass.

Per-pixel indirect specular

SIGGRAPH 2025 Vancouver+ 10-14 August

Denoising

- New pass specialized for specular
- Different reprojection
- Firefly removal



This is the result of the firefly removal.

====================
For the firefly removal, we clamp the luminance of a pixel if its value is too high compared to the neighbor average intensity.

So moving on to challenges in production.

Alpha sampling

- Use for reference
- Most (too) expensive

Alpha-tested materials are known to be costly due to the need to load and test the alpha textures when traversing the BVH. This is also problematic since many leaves are condensed in clusters, leading to divergent times for each of our ray tracing compute groups.

Our reference cost throughout the next slides is this result, not limiting the any-hit calls and fetching each texture until either a hit is found or the leaves missed.
Its well known to be too expensive for real-time in dense forests.

Alpha sampling

- Use for reference
- Most (too) expensive

Limit any-hit calls

- Force opaque after N hits
- Biases result
- 85% cost* (n=2)

*relative to reference on a Nvidia GPU

Limiting the number of any-hit calls can help us reduce divergence in compute groups but adds bias through added occlusions. This is of course the first implementation as it's easy to implement.

Alpha sampling

- Use for reference
- Most (too) expensive

Limit any-hit calls

- Force opaque after N hits
- Biases result
- 85% cost* (n=2)
- 90% cost* (n=4)

*relative to reference on a Nvidia GPU

Letting 4 tests is a good floor to keep a closer look to reference without being unbound.
Spectular RTGI uses this option as the cost of the pass was less of an issue on the targeted hardware.

Alpha-tested materials

SIGGRAPH 2025 Vancouver+ 10-14 August

Dithering

- Use average opacity
- Blue noise per ray
- Modify per any-hit invocation
- No performance gain*

*relative to reference on a Nvidia GPU

An other solution we tested was dithering the alpha-test through noise. You do so by using a precomputed average opacity and could be useful in ray-tracing engines that do not access textures from the BLAS. This did not bring any performance gains for us, even sometimes being more costly than the real test.

Alpha-tested materials

SIGGRAPH 2025 Vancouver+ 10-14 August

Dithering

- Use average opacity
- Blue noise per ray
- Modify per any-hit invocation
- No performance gain*

Limit any-hit calls

- 82% cost* (n=2)

*relative to reference on a Nvidia GPU

This can be improved by limiting the number of any-hit calls again, helping performance but adding occlusion.

Alpha-tested materials

SIGGRAPH 2025
Vancouver+ 10-14 August

Opaque approximation

- Scale triangles according to average opacity
- Fully opaque intersection
- Close to n=2
- 60% cost*

*relative to reference on a Nvidia GPU

The solution used in the end is an opaque approximation. We scaled the triangles according to average opacity and force them to be opaque, leaving us with better performance for the trace and close to our reference in result. We did not analyze the introduced bias, but found our result to be very close to the alpha-test reference, at almost half cost.

Here it is compared to our reference result.

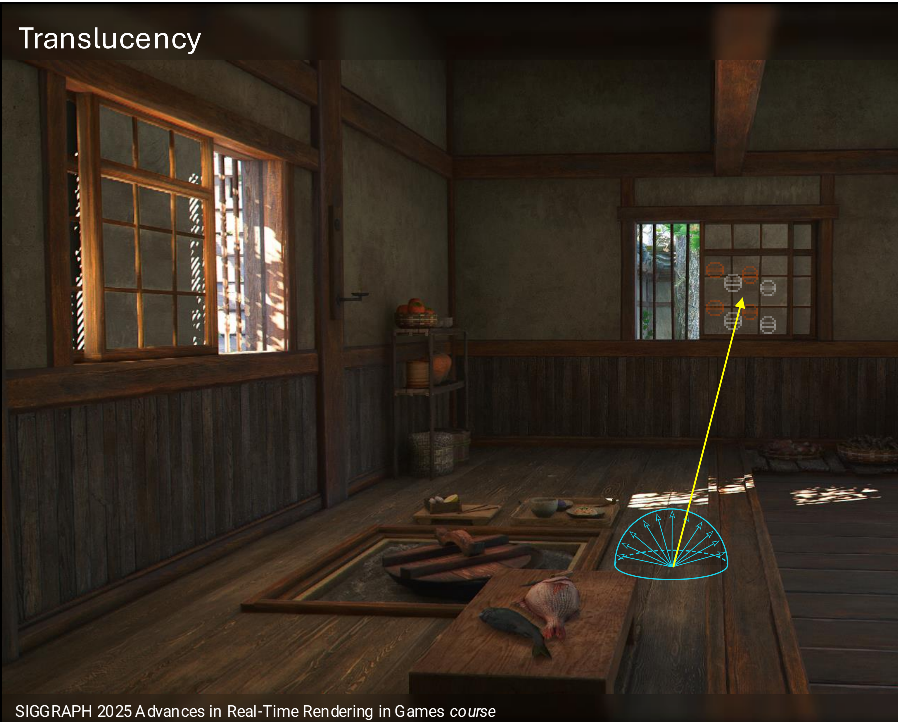
This is with alpha test texture, limiting the any hit shader to 2 calls.

And finally, this is our fully opaque approximation.

Moving on, AC Shadows contains a lot of translucent materials. Either through doors and window panels made of paper, or tree leaves and greenery.

Here I will present each step we added to support translucency. As you can probably see, or not see, this image is very dark. Small opening limit the number of secondary hits finding direct lighting. We are missing most indirect lighting from the windows and door.

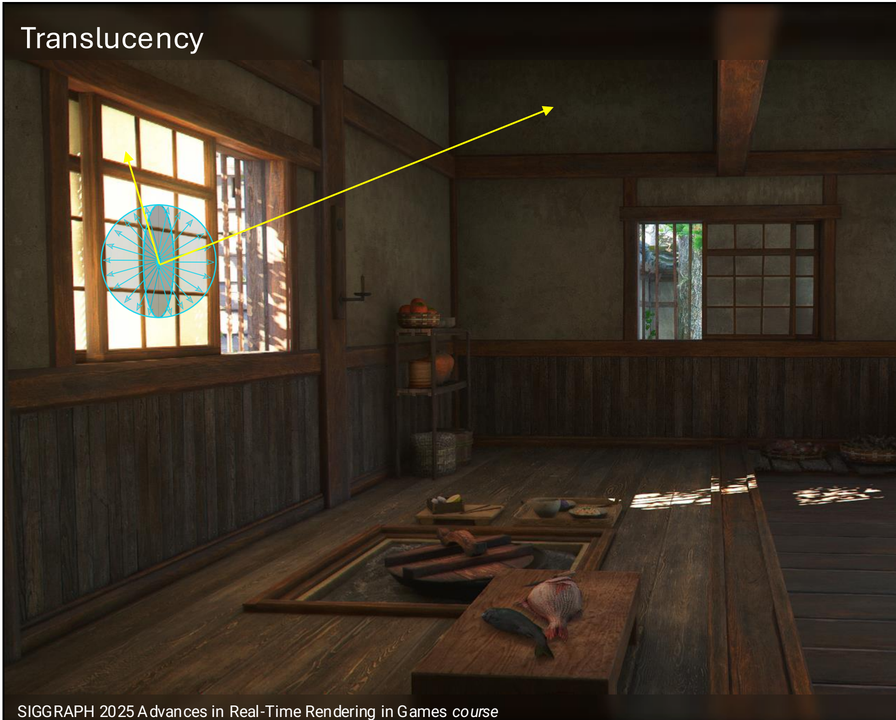
To improve direct lighting on the secondary hit, we let it accept back-facing normals. Good to note we support only thin translucency here, so the back face normal is the inverted normal vector. Translucency factor and albedo also affect that lighting contribution.



For indirect lighting on the secondary hit, we evaluate each faces' irradiance by calculating weights according to the normal, splitting our eight probes between both faces and evaluating irradiance separately before summing them together.

Translucency

SIGGRAPH 2025
Vancouver+ 10-14 August



Per-pixel indirect

- Randomly select one side with translucency probability

- Divide radiance by pdf
  Back-face probability = t/(1+t)
  pdf = 1/(1-probability)
- Flip backface direction for SH encoding

When our primary ray is on the translucent surface. We trace our secondary ray by randomly selecting one side, with our probability tuned to the translucency. If the back face is chosen, we fully skip screen space tracing.
We rebalance the result according to our new pdf which must contain the parameters of the material for the back-face rays.
Finally, we always encode a front facing ray direction and so flip the ray if it was thrown behind the surface.

- Backface Prob = t / (1+t)
- Trans = t*albedo for standard material and more complex for foliage
- (trans/albedo)/prob or 1/(1-prob)

This is the final result taking into account thin translucent materials.

Light leaks in real-time GI solutions is a well-known issue. Here we split our two main leak sources, direct and indirect illumination.

Direct lighting

Shadow fetches outside cascaded shadow map (CSM) coverage is assumed in shadow. Even then :

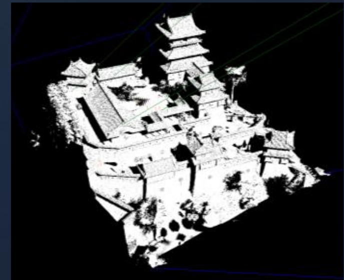
- ***CSM has optimized occluders***
  Objects can be culled from CSM in regions that are occluded on screen.
- ***CSM has limited range***
  Objects too far away do not have shadow information.
- ***Wall are too thin***
  Leaks when bias is not strong enough for combination of sun orientation and wall thickness.

For direct lighting, the main issue is missing shadow informations. Some leaks happen even in CSM range.

SIGGRAPH 2025 Vancouver+ 10-14 August

Secondary shadow map

- Objects can be culled from CSM in regions that are occluded on screen
- Low-resolution shadow map with all occluders

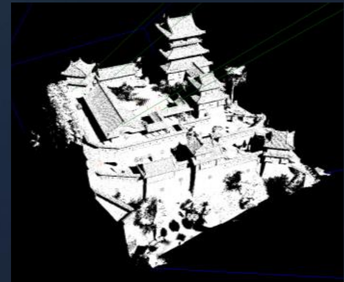


Our secondary shadow map helps to bridge the missing information from the CSM.

SIGGRAPH 2025
Vancouver+ 10-14 August

Secondary shadow map

- 1k x 1k resolution
- 128 meters coverage, 64 from camera
- Split in 4x4 tiles
- Time-sliced update over 16 frames



We update the secondary shadow map every 16 frames as we time-slice its update.

Light leaks

SIGGRAPH 2025 Vancouver+ 10-14 August

Secondary shadow map

We reduce secondary shadow map leaks with conservative filtering

- Take the min value of gather4 for probes

```
secShadow = min4(Gather(...));
```

- Apply a filtering bias for hit lighting

```
bias =  0.5f;
secShadow = saturate((secShadow - bias)/(1-bias));
```


We also add extra bias during probe lighting pass. The filtering bias here is the same we use for our CSM in deferred lighting

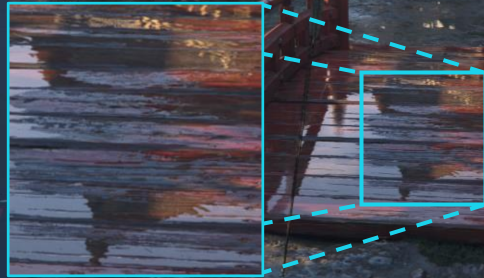
===============

We use a more aggressive filtering for the probe computation to reduce even more leaks.

Light leaks

Outside shadow maps

When outside both the CSM and secondary shadow map, consider shadowed to reduce light leaks in probes.



Less apparent in diffuse, but some areas fall outside both our shadow maps, resulting in wrong reflections.

Outside shadow maps

When outside both the CSM and secondary shadow map, consider shadowed to reduce light leaks in probes.

Shadow ray cast

- Only on high quality modes on PC
- Cast during world tracing pass

On high end platform, for specular reflection we add shadow ray casting when falling outside of the secondary shadow map.

====================

- Added cost for shadow ray casting is 0.1 ms on pc, 0.3-0.4ms on ps5 pro

Indirect lighting

By fetching the 8 probes around our secondary hit we potentially inject outdoor lighting inside.

- ***The probe VSM's have a too small resolution***
  The visibility test is not precise enough for thin wall, angled wall or corners.
- ***Outdoor lighting is many factors stronger than indoor***
  Even reducing outdoor probes' weights does not compensate the difference.
- ***Occlusions can be missed if too small***
  The unique point of view of the probe cannot always contain the full complexity of occlusion inside it's cell.

Our biggest artefact that we have not yet totally fixed is light that is not supposed to be included is added into the system.
When computing indirect lighting on a secondary hit position with probe evaluation, we evaluate all eight surrounding probes.
With secondary hits on walls separating interiors and exteriors, some probes bring in exterior lighting.

Probe filtering

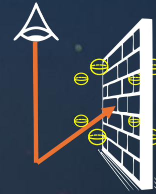
Offset query position

- Only done for GI (irradiance probe)
- Scaled by probe cascade

View direction offset

Sampling direction offset

Inverse ray direction offset



To help probe selection and weighting, we offset our sampling position.
The last offset we use, introduced by the DDGI Resampling paper, moves the position backwards along the ray. We limit that final offset to half the traced distance on our side. Our most common issue with inverse ray direction is that many rays tend to run almost parallel to the walls and so doesn’t distance our sampling position enough.

SIGGRAPH 2025
Vancouver+ 10-14 August

Probe filtering

Weighting each probe according to secondary hit

1. Directional weight (wrap shading)

```
float wrapShading *= (dot(sampDir, probeDir) + 1) * 0.5f;
wrapShading = (wrapShading * wrapShading);
```

2. Visibility (Variance shadow map)
3. Reduce small weights

```
if (w < 0.2)
    w *= (w * w) * (1.f / (0.2 * 0.2));
```

4. Multiply by trilinear interpolation weight

With the offset position, we build the weight of each probe.
First, we evaluate the directional weight. Using a pure dot product might rule out all the probes so we use wrap shading. [Sloan et al. 11] “Wrap Shading”
For the visibility we evaluate the probe VSM.
Then we crush weights smaller than 0.2
And finally multiply by trilinear weight

Light leaks

SIGGRAPH 2025
Vancouver+ 10-14 August

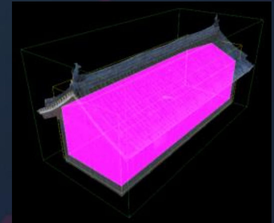
Outdoor probe leaks

- Visibility test is not precise enough
- Outdoor probes are many factors brighter than indoor
- Sampling offsets can push the position closer to walls or outside.

Still, these weight are not enough to completely remove outdoor lighting for indoor as the lighting factor difference is too large.

Indoor volumes

List of planes already available from baked GI.



We reuse the list of planes from our baked GI solution.

Light leaks

SIGGRAPH 2025 Vancouver+ 10-14 August

Indoor volumes

- Classify and compare probes and secondary hit sampling position.
- Classify both faces separately for translucency
- Used in caves to keep probes valid under terrain
- Add GI blockers for caves

With these planes, we classify our secondary hits and our probes.

Not shown here is GI Blocker mesh. They help by killing light of all rays hitting them. Using them brings in false occlusion but compensates leaks for very dark interiors. They are used exclusively in caves.

=========================
Each probe is classified to be either interior or exterior. When using the probe volume, we first classify the hit position and then only use probes having the same classification.

SIGGRAPH 2025
Vancouver+ 10-14 August

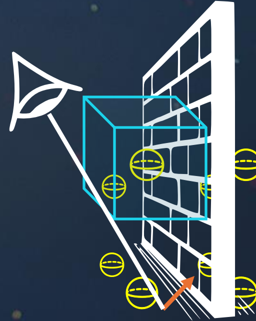
Indirect occlusion (Probe AO)

- We are missing occlusion inside the probe range since probes only have a single viewpoint for their cell
- Add occlusion when intersection is shorter than probe cell size
- Only for irradiance probes

Since our probes have a single point of view when evaluating their buffer and irradiance, we are missing micro details from inside their range.

Indirect occlusion (Probe AO)

$$relativeHitDist = \overrightarrow{hitDistance} \,/\, \overrightarrow{GridSpacings}$$
$$irradiance \mathrel{*}= saturate(relativeHitDist * scale + bias)$$



By comparing the traced distance of our ray to the probe’s range, we assume that any distance shorter than the range means some micro-visibility was missed during probe evaluation.
We simulate the missing occlusions by calculating that ratio and applying it over a predetermined range.

In this section we will go over the techniques used for occlusion

Starting from our diffuse GI result using spherical harmonics and without any added occlusion. We can notice missing occlusions and micro occlusions.

We first add the previously explained probe AO that takes into account ray lengths shorter than our probe spacings.

We then add a RTAO, a per-pixel AO that comes from our denoiser, although more an artistic occlusion, it helps bring in ambient occlusion similar to SSAO using our raytraced rays.

We then do a post-effect pass of GTAO. (Practical Real-Time Strategies for Accurate Indirect Occlusion, Jimenez & Wu 2016)
Our GTAO is parametrized to bring back microdetails lost in filtering and from missing objects in our BVH.

No added occlusion.

With occlusion.

Final result with occlusion

For noise, we adapted our denoising solutions to the specific challenges we faced during development.

Denoising

- Spatial filtering bleeds light on opaque surfaces
- Separate mask for filtering (material ID: translucency, character, vegetation, ...)

To remove bleeding of translucent material on opaque material during the denoising, we added a mask containing the material type and use it to remove neighbor samples during spatial filtering passes.

================

The material ID is also used to make special adjustement on the weights and how the samples are fetches.

SIGGRAPH 2025 Vancouver+ 10-14 August

Per-pixel infinite ray

- Noise noticeable in interiors with high frequency lighting due to small openings
- Even more noticeable in disoccluded region with no history

If we dont limit range of rays, its very noisy.

Probe radiance cache

- Use shorter rays with fallback on radiance cache
- Reduces high frequency details
- Better performance
- Cache is filtered and temporally stable
- Trade-off between precision and noise

We instead trace shorter rays and fallback on the radiance cache. This helps mostly for performance and reducing noise but is a tradeoff for precision as it leaks more lighting.

============
We also lose a bit of details since our nearest probes are spaced at 2m from each others.

SIGGRAPH 2025
Vancouver+ 10-14 August

Disocclusion

- Tried using "probe GI" for disoccluded pixels on characters and replace their result. Too much artefacts.
- Added an extra spatial filtering pass using a Poisson filter with 8/16 samples for disoccluded pixels

Disocclusion was an issue with our 2D motion vectors. We added a filtering pass for disoccluded pixels, mostly for character movement.

===========

Disocclusion is always a problem since we can't use temporal filtering. Adding an extra spatial denoising pass for those pixels help reduce the noise. In our engine, since our motion vector does not contain any depth information, it is difficult to validate the reprojected pixel and add more disocclusion on moving entities, leading to increased noise.

SIGGRAPH 2025
Vancouver+ 10-14 August

Vegetation disocclusion

- Remove normal contribution to weights
- Relax temporal reprojection criteria for moving vegetation
- Expand temporal reprojection search for vegetation using 16 pixels instead of 4

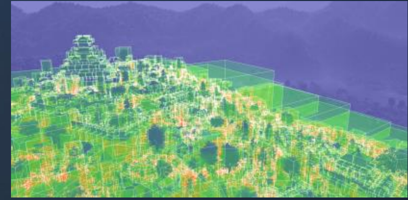
Tall grass blades in the wind have high normal variance and disocclusion. We adapted our denoiser for them.

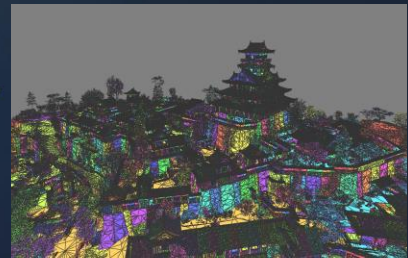
Performance

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

Statistics

- Typical urban scene
  - 2k+ BLAS
  - 30K+ instances
  - 300 MB BLAS Data
  - 20 MB TLAS Data





On the BVH side, during runtime, we have around 2 thousand BLASes loaded and around 30 thousand TLAS instances. Taking up 320 megabytes total in memory. This is a typical urban scene,

==============

Those numbers were taking on an Xbox Series X.

Performance

SIGGRAPH 2025 Vancouver+ 10-14 August

Probe memory

	Pixels per probe	Bytes per pixel	Total size in bytes
G-buffer	20x20	14 bytes	55MB
Visibility	10x10 (12x12 with borders)	2 bytes	2.8MB
Radiance cache	10x10 (12x12 with borders)	8 bytes	11MB
Irradiance	5x5 (7x7 with borders)	8 bytes	3.8MB

- 10240 probes
- 73MB in total

The largest buffer is the probe G-buffer which is 55MB. We are using 14 bytes per pixel to represent all the different material attributes.
You can also see here that other buffers require a one pixel border, that's because we need to sample them with bilinear filtering.

The total size for probes buffers is 73MB for 10k probes.

SIGGRAPH 2025
Vancouver+ 10-14 August

Timings

Performance view point in Osaka for timings

Probe update timings

Async / Graphic	PlayStation 5 1024 probes	PlayStation 5 Pro 1024 probes	Xbox Series X 1024 probes	Xbox Series S 512 probes	RTX 4070 512 probes
G-buffer update	0.35 ms	0.2ms	0.26ms	0.32ms	0.12ms
Sun shadows	0.20 ms	0.13ms	0.12ms	0.18ms	0.30ms
Lighting update	0.49 ms	0.26ms	0.35ms	0.46ms	0.44ms
Lighting convolution	0.06ms	0.04ms	0.07ms	0.07ms	0.06ms
Probe relocation	0.11 ms	0.09ms	0.12ms	0.09ms	0.05ms
Total (No Async)	(0.91) **1.24 ms**	(0.60) **0.66ms**	(0.78) **0.92ms**	(0.88) **1.12ms**	(0.42) **0.97ms**

- 16% of updated probes are traced

In teal, passes running in async queue, and green on the graphics queue. In orange, timings for a single queue graphics pipeline.

GPU timings for the probe update on all different platforms:

- Profiled in quality mode at 30fps on consoles.
- 16% of probes are traced in the G-buffer update pass.
- Halved update frequency on XBSS to keep same cost.
- We also update 512 probes per frame on PC but that’s because we’re running at a higher framerate so we don’t need more than that.

Performance

SIGGRAPH 2025
Vancouver+ 10-14 August

Diffuse GI timings

Async / Graphic	PlayStation 5 1440p	PlayStation 5 Pro 1440p	Xbox Series X 1440p	Xbox Series S 900p	RTX 4070 1440p
SS tracing	0.61ms	0.29ms	0.49ms	0.38ms	0.27ms
WS tracing	1.13ms	0.49ms	0.88ms	1.04ms	*0.55ms
Lighting	3.34ms	1.23ms	2.03ms	2.52ms	0.97ms
Denoising	2.04ms	1.24ms	1.58ms	1.28ms	0.86ms
Total (No-Async)	(4.58) **7.12ms**	(2.48) **3.25ms**	(3.75) **4.98ms**	(3.19) **5.22ms**	(1.90) **2.65ms**

- Traced at quarter resolution

In teal, passes running in async queue, and green on the graphics queue. In orange, timings for a single queue graphics pipeline.

- Here are the GPU timing for the per-pixel diffuse GI which is traced at quarter resolution.
- This was profiled with a fixed internal render resolution of 1440p except on XBSS which is 900p here to keep a similar cost as the XBSX.

Performance

SIGGRAPH 2025 Vancouver+ 10-14 August

Specular GI timings

Async / Graphic	PlayStation 5 Pro 1440p	RTX 4070 1440p
Tracing	0.99ms	0.70ms
Lighting	0.80ms	0.92ms
Denoising	1.98ms	1.6ms
Total	**3.77ms**	**3.22ms**

- Traced at half resolution

Finally, here are the GPU timings for RT specular traced at half resolution.

- Specular RT is only available on the PS5 Pro in quality mode at 30fps and on PC.
- It is quite costly on the PS5 Pro in half resolution but the quality in quarter resolution was not good enough and we were still reaching our target framerate and resolution so we kept it at half resolution.

Conclusion

SIGGRAPH 2025
Vancouver+ 10-14 August

Limitations of our implementation

一 **Shipping two GI systems is time-consuming**
- Tailoring two looks for different contrast and GI results

二 **Ray tracing materials still requires manual labor**
- Complex shaders cannot always be easily simplified

三 **Lack of AO on objects not in BVH**
- Limited by tracing performance

四 **Ray tracing world quality is fixed**
- Per platform setting, no runtime selection or LODs

五 **2D motion vectors create more disocclusion**

1. Reviewing all content in two different looks. Specific data created for both solutions.
2. Manual conversion for our simplified RT material. Long manual labor.
3. Missing many assets in BVH cause issues.
4. Same asset quality at the limit of our BVH or at the player's feet. Something we hope changing can help performance in the future.
5. We hope to use 3D vectors in our next title to help disocclusions.

Limitations of the current algorithm

一 **Probe grid and buffer resolutions limit details**
- Need more probe resolution and smaller grid spacing, but too expensive with uniform probe distribution in the cascades

二 **Probe interpolation always leaks**
- No good way to resolve leak-proof
- Visibility is complex to resolve

三 **Screen tracing optimization creates issues**
- BVH geometry differs from raster, creates self-intersection disparity

四 **Complex cases are still noisy**
- We had to remove indoor volumes from basements
- Removing probe radiance fallbacks introduces more noise
- Modern frames are upscaled and filtered/denoised again

1. Uniform probe distribution in our cascade does not scale well if we want larger or more precise coverage. Spacings are too sparse, and probe buffers to small. Costly to scale currently.
2. Leaks are bound to happen with this algorithm, visibility is a big factor in this.
3. Difference in world creates occlusion issues, offsetting is not a soluton as our worst found case was 50cm a differences (We need to put constraint on our simplified geometry).
4. Noise patterns appeared with some frame upscalers, hard to have a noisy GI results upscaled so much.

Our Future Work

一 **Reduce production support load**
- More automatization of our BVH construction

二 **Scale to lower platforms**
- More optimizations, hopefully get rid of baked technique

三 **Reduce noise and leaks**
- Implement importance sampling
- Investigate generating probes on surfaces and sparse cascades

四 **Better and faster denoising**
- Our current denoising costs almost half the pass

1. Reduce complexity of supporting RTGI. Automation of material simplification.
2. XBSS RTGI was not planned and was not possible due to memory constraint for AC Shadows. We are working towards making sure our next title will support RTGI for all platforms.
3. Many solutions to look into to reduce leaks. Most will add cost that will need to be balanced out.
4. Take more time looking into denoising, we have not spent a lot of time on this.

Questions?

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

- We have much more details and less smearing of lighting values

- We have much more details and less smearing of lighting values

- We have much more details and less smearing of lighting values

- We have much more details and less smearing of lighting values

- We have much more details and less smearing of lighting values

SIGGRAPH 2025 Vancouver+ 10-14 August

Convolution output

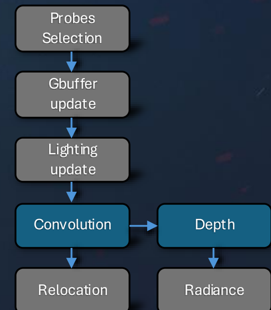
```
void WriteOctahedronWithBorder(RWTexture2DArray<float4> dest, uint3 destOffset, uint2 octTexel, uint2 octSize, float4 value)
{
    uint3 destTexel = uint3(destOffset.xy + 1 + octTexel, destOffset.z);
    dest[destTexel] = value;

    bool isEdgeX = octTexel.x == 0 || octTexel.x == octSize.x - 1;
    bool isEdgeY = octTexel.y == 0 || octTexel.y == octSize.y - 1;

    if (isEdgeX)
    {
        destTexel.x = destOffset.x + ((octTexel.x == 0) ? 0 : (octSize.x + 1));
        destTexel.y = destOffset.y + octSize.y - octTexel.y;
        dest[destTexel] = value;
    }

    if (isEdgeY)
    {
        destTexel.x = destOffset.x + octSize.x - octTexel.x;
        destTexel.y = destOffset.y + ((octTexel.y == 0) ? 0 : (octSize.y + 1));
        dest[destTexel] = value;
    }

    if (isEdgeX && isEdgeY)
    {
        destTexel.x = destOffset.x + ((octTexel.x == 0) ? (octSize.x + 1) : 0);
        destTexel.y = destOffset.y + ((octTexel.y == 0) ? (octSize.y + 1) : 0);
        dest[destTexel] = value;
    }
}
```



- This is the code used when writing the result of the convolution for probe texel. It add a boundary texel to enable the use of hardware bilinear filtering.