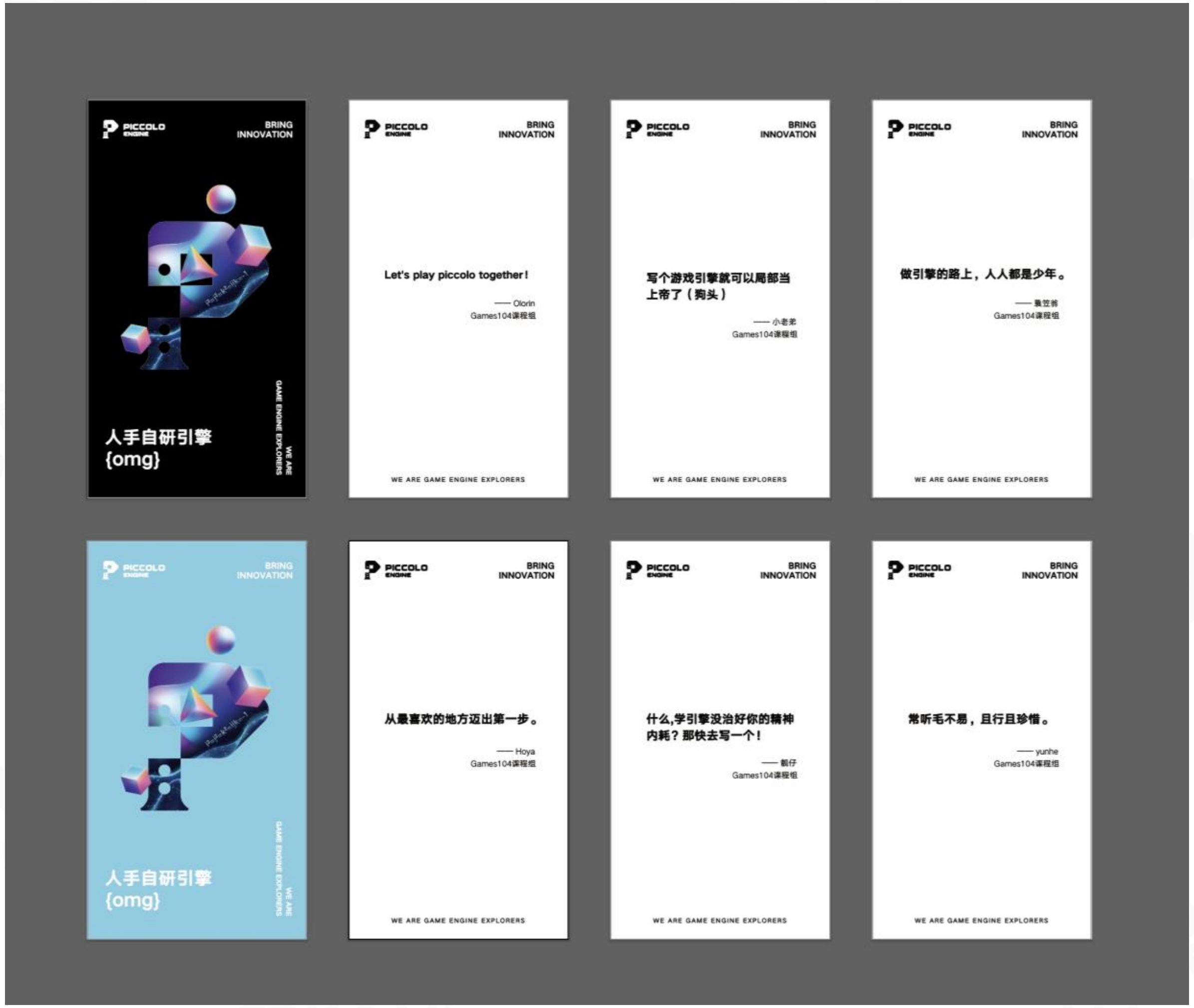
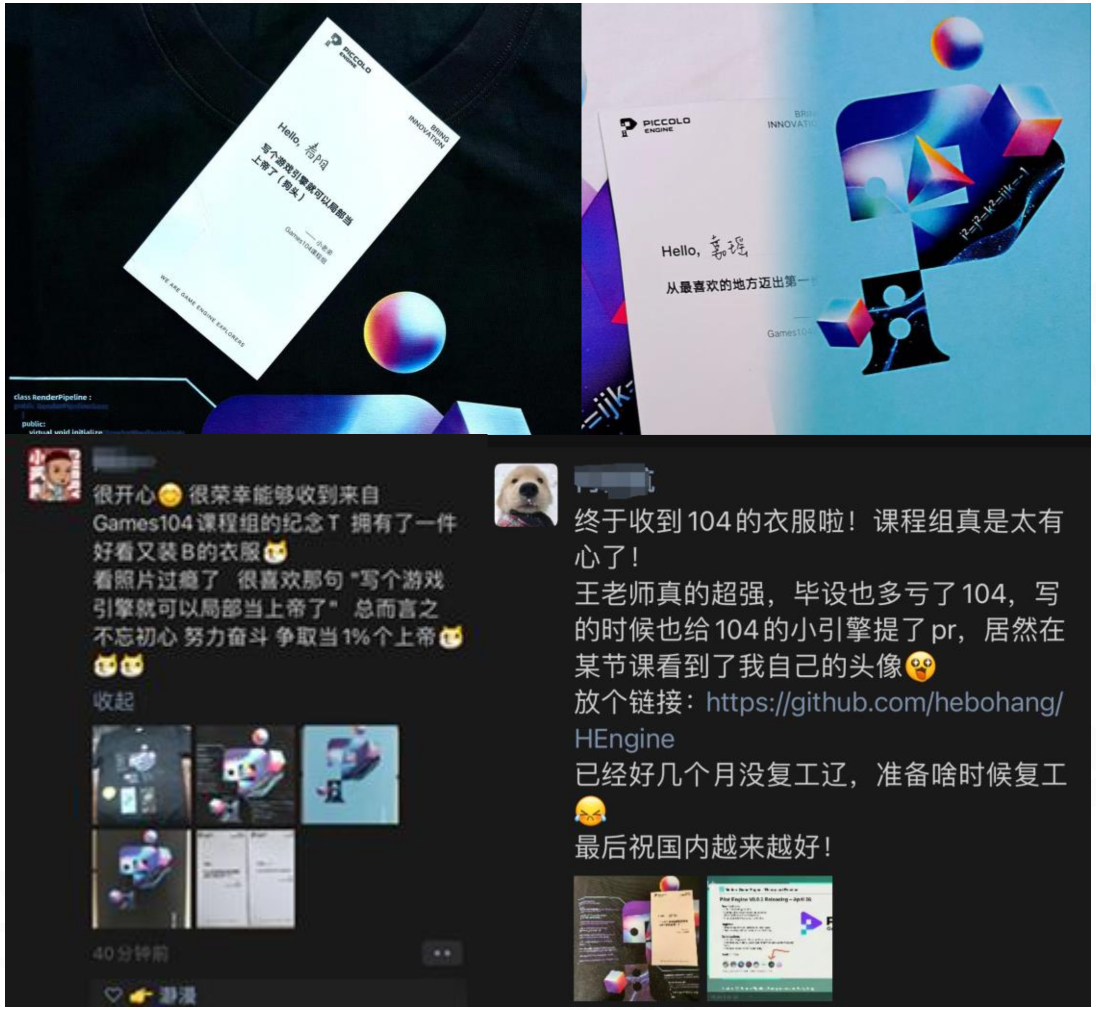
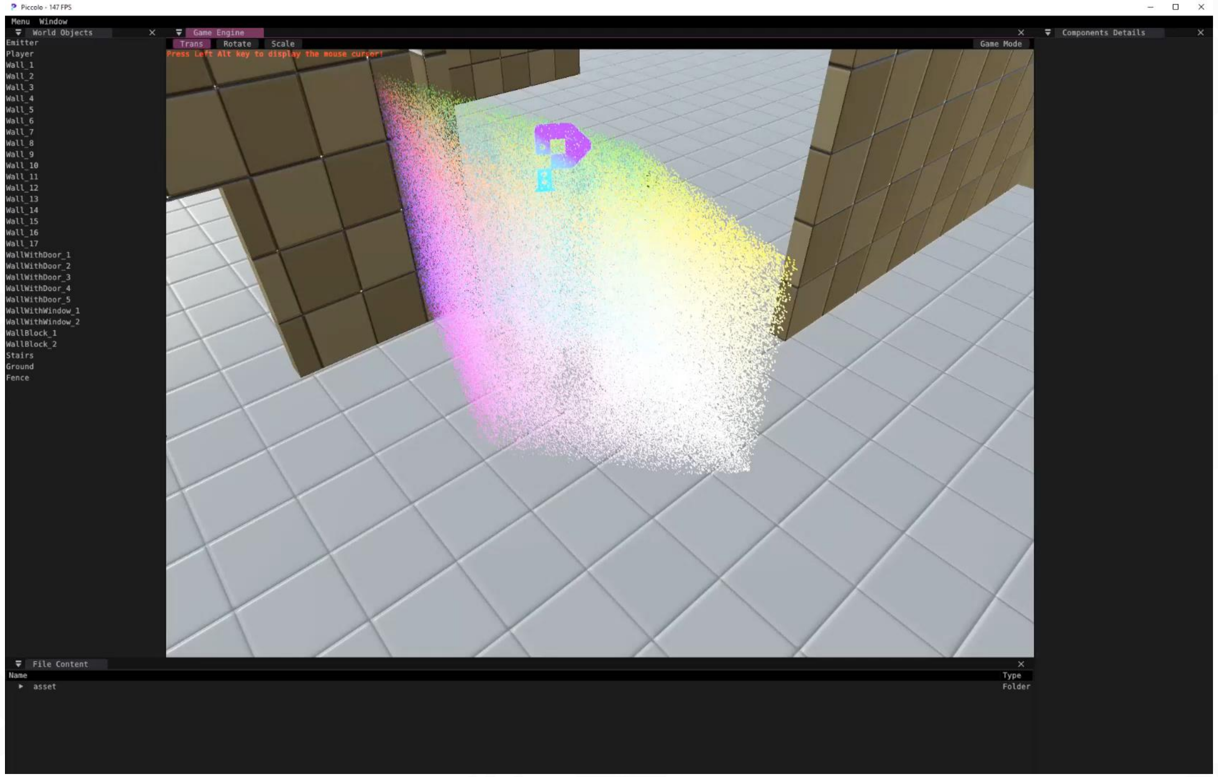
Our Best Wishes to Every Explorer





Piccolo Engine v0.0.8 Released – 12 September

- GPU-based Particle System!

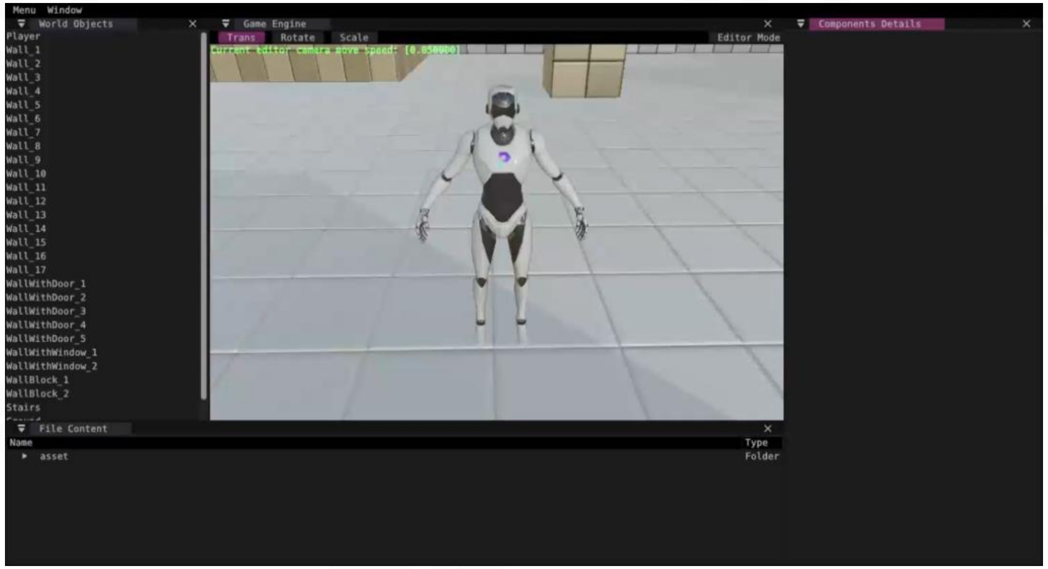


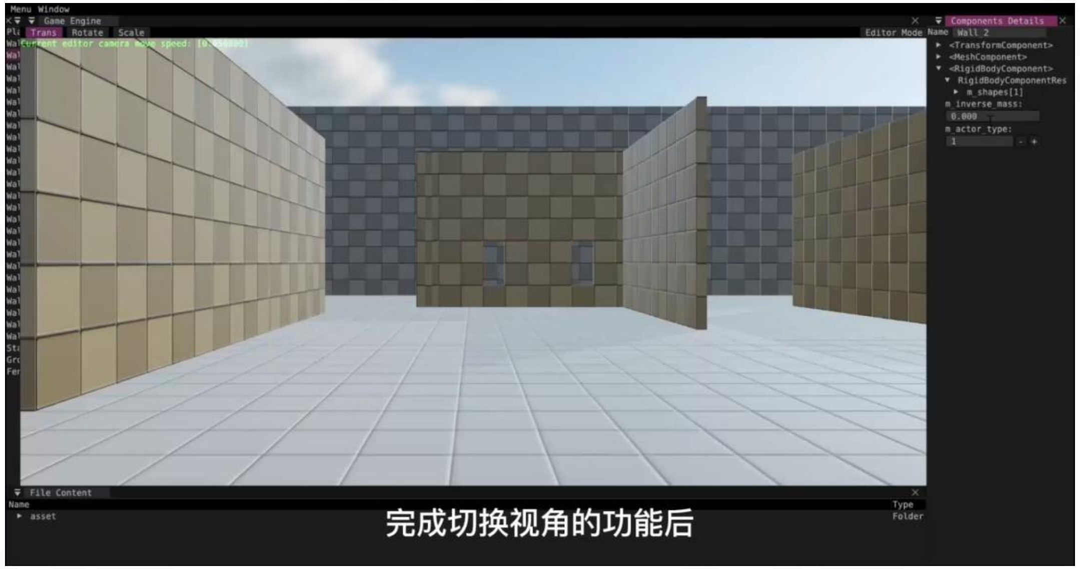
- Piccolo Code Explained - 10 October

关注公众号，回复【入群】，加入我们的社区

PICCOLO
Game engine

Homework Showcase (1/2)



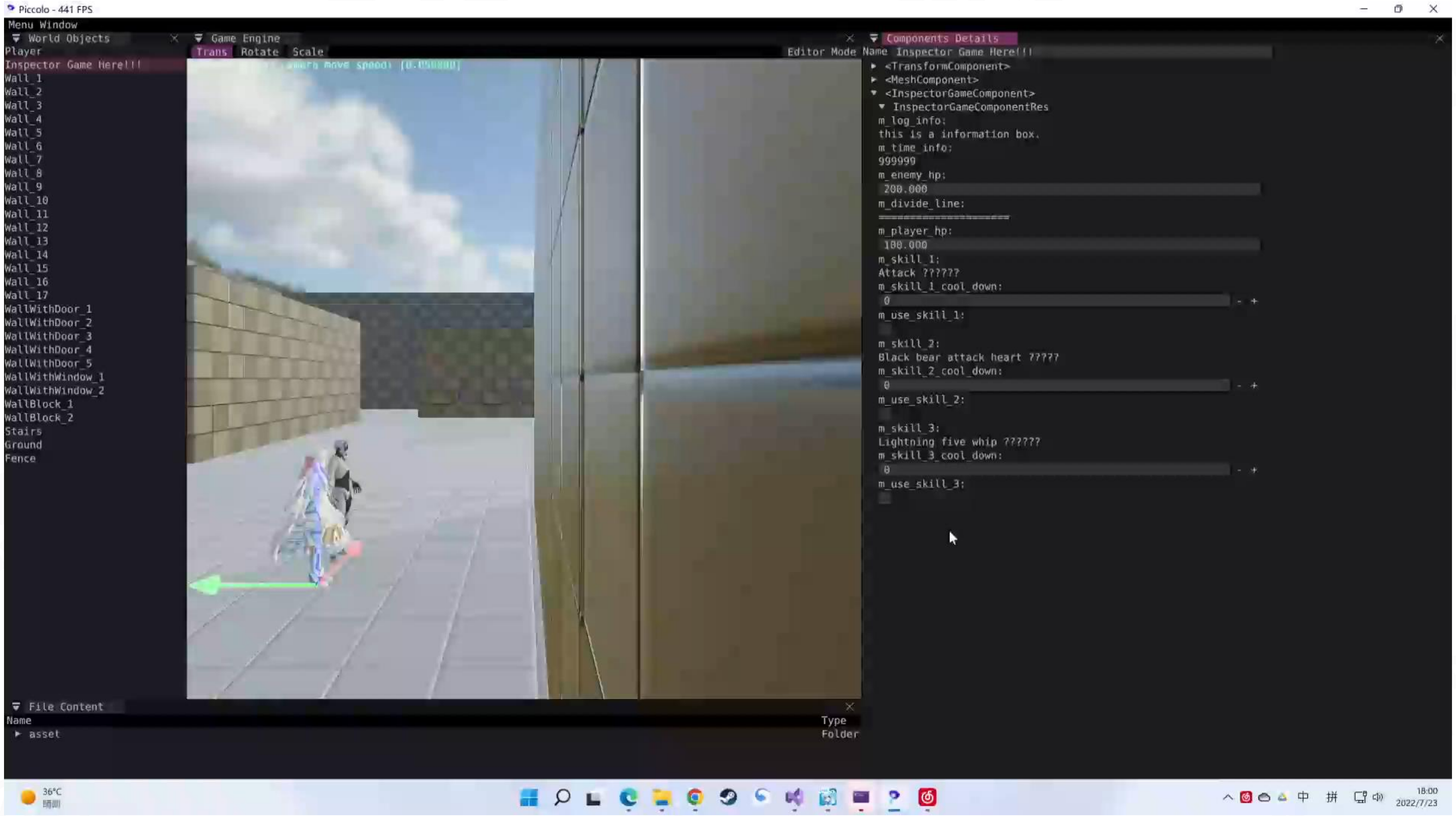


1. Added color type reflection UI
2. Added changing mesh base color function

Added camera mode change function

Homework Showcase (2/2)

A mini game!



Q&A

- Q1: How does ECS handle destroy of entities?
- Q2: How can we measure Cache miss?
- Q3: How should we provide tools for designers to design functions under DOP architecture?

Lecture 21

Dynamic Global Illumination and Lumen

Advanced Topics

WANG XI

GAMES 104

2022

Global Illumination(GI)

The Rendering Equation

James Kajiya,”The Rendering Equation.”

SIGGRAPH 1986.

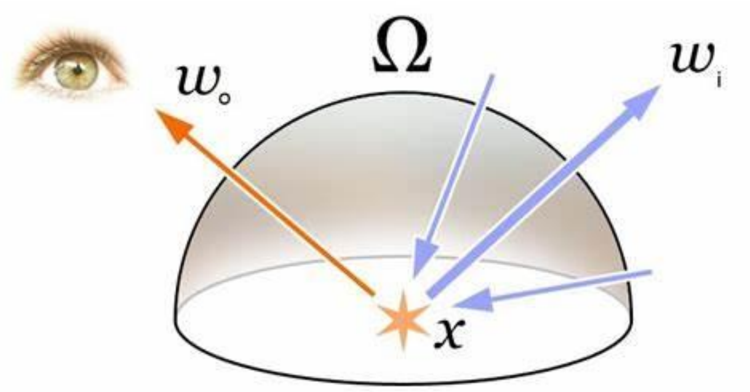
Energy equilibrium:

$$L_o(\boldsymbol{x},\omega_o) = L_e(\boldsymbol{x},\omega_o) + \int_{H^2} f_r(\boldsymbol{x},\omega_o,\omega_i) L_i(\boldsymbol{x},\omega_i)\cos\theta_i d\omega_i$$

outgoing emitted reflected

Radiance and Irradiance

The Rendering Equation



outgoing/observed radiance

emitted radiance (e.g., light source)

angle between incoming direction and normal

incoming radiance

$$L_o(\mathbf{x}.\omega_o) = L_e(\mathbf{x},\omega_o) + \int_{H^2} f_r(\mathbf{x},\omega_o,\omega_i) L_i(\mathbf{x},\omega_i) \cos\theta_i \, d\omega_i$$

point of interest

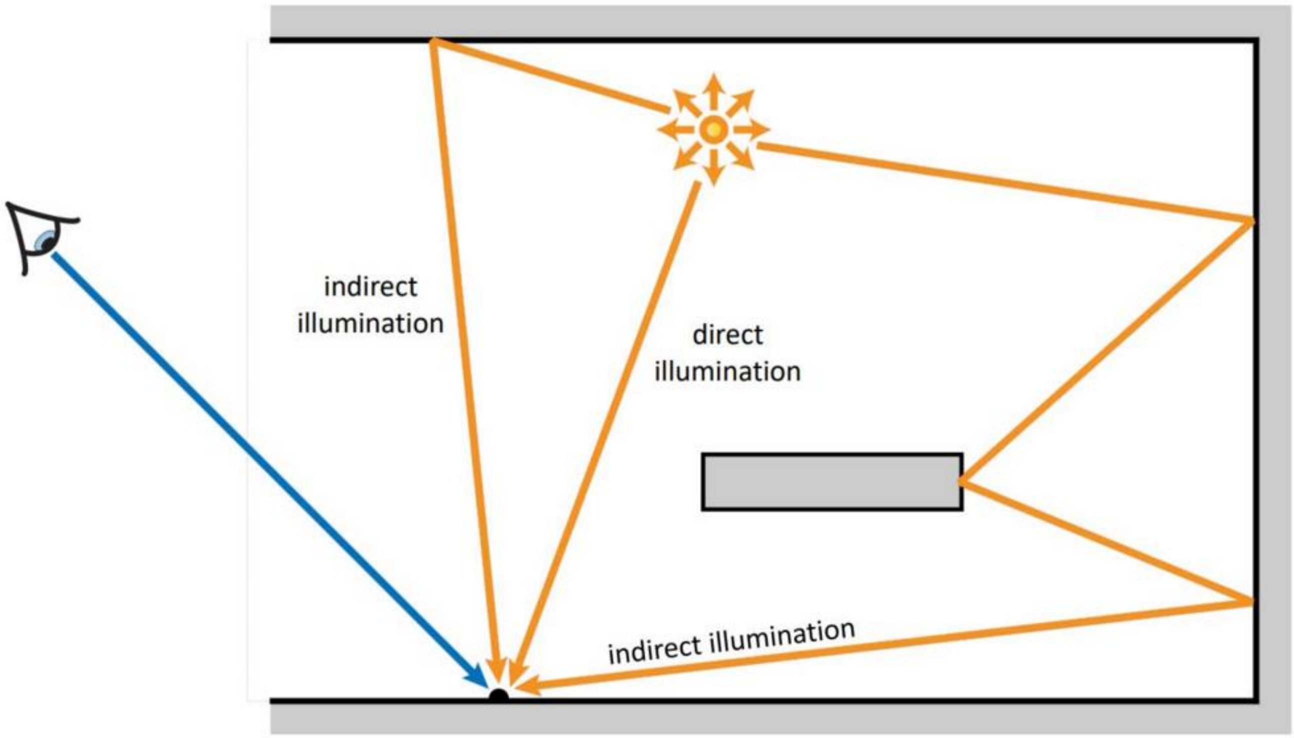
direction of interest

all directions in hemisphere

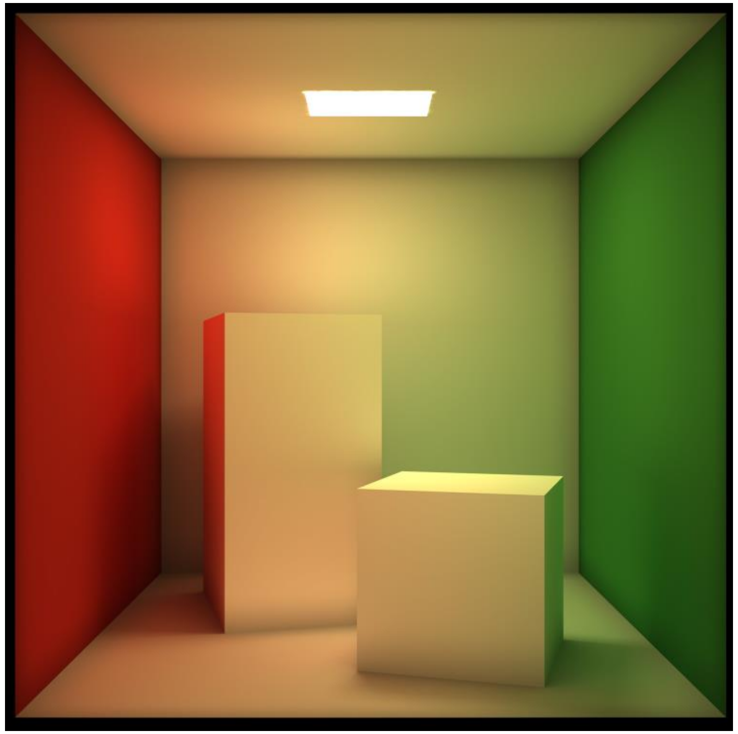
scattering function

incoming direction

Global Illumination: Billions of Light Source



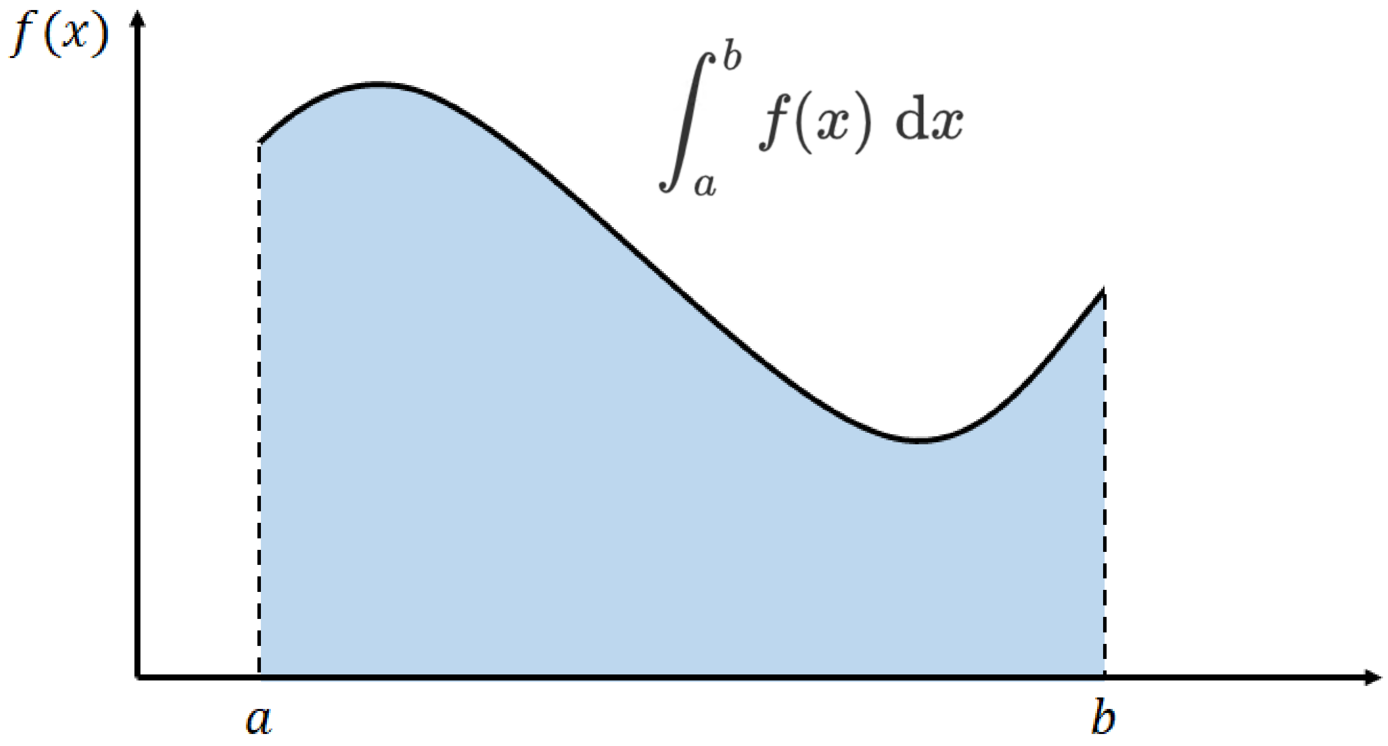
Direct vs. Indirect Illumination



Global Illumination (GI)

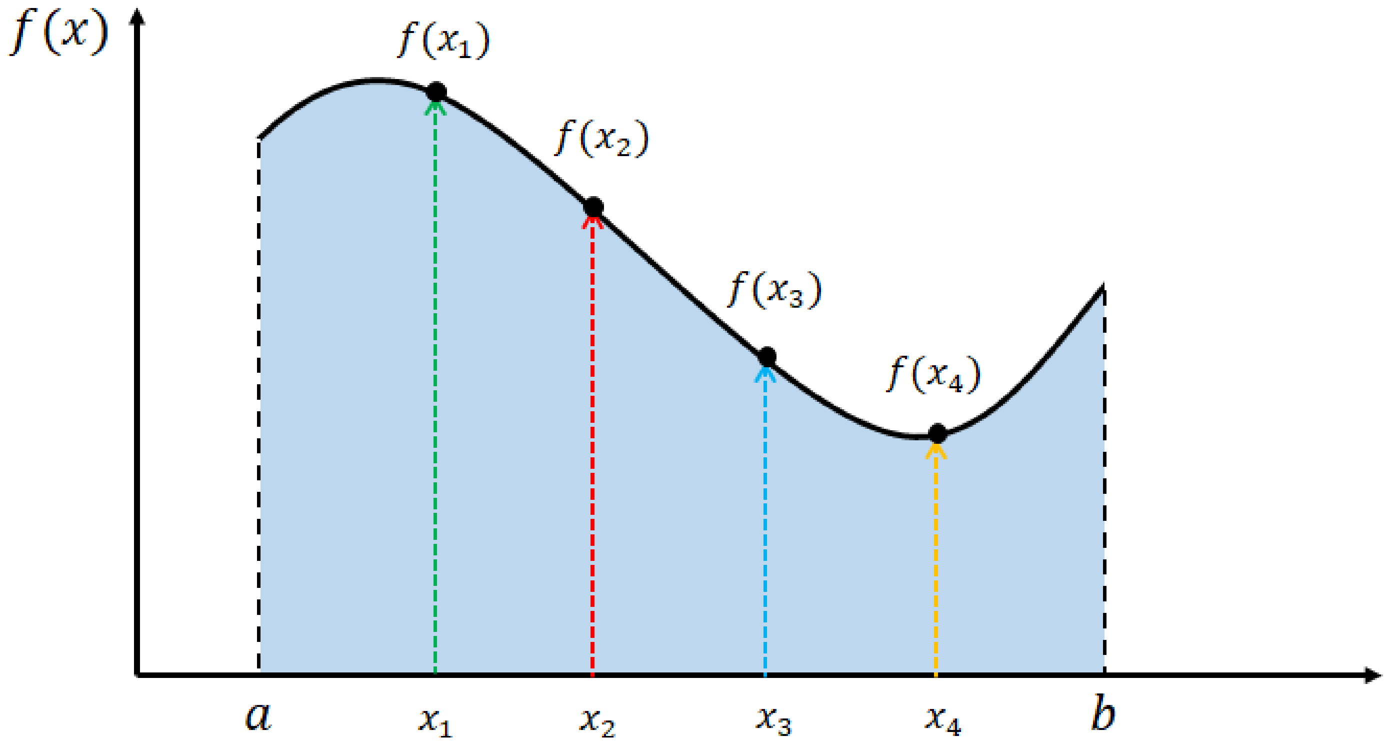
Global Illumination is Matter for Gaming

Monte Carlo Integration



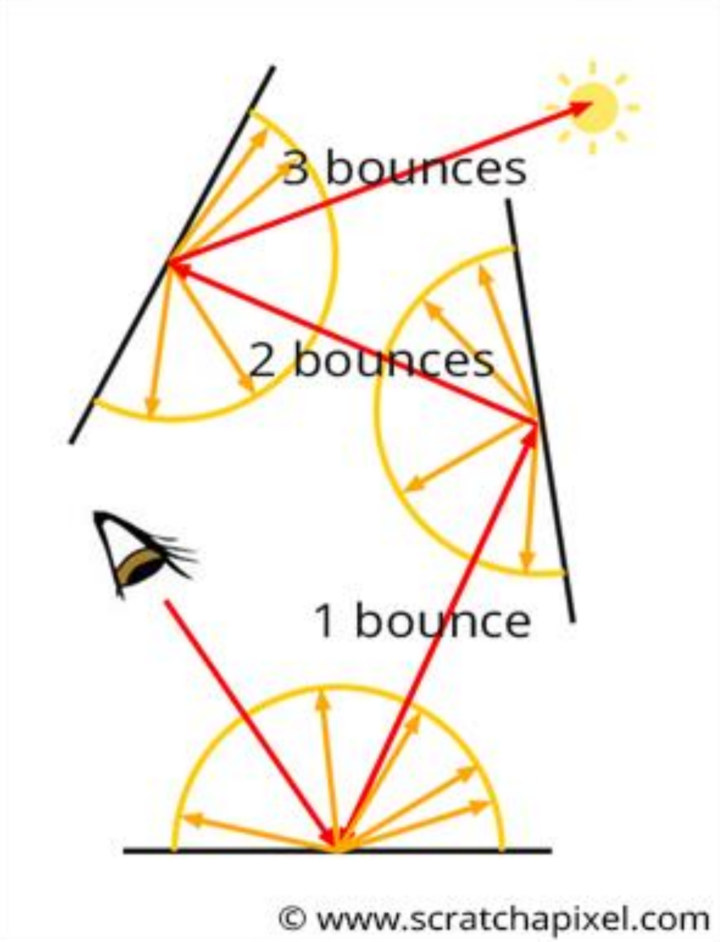
- How to solve an integral, when it's too hard to solve it analytically?

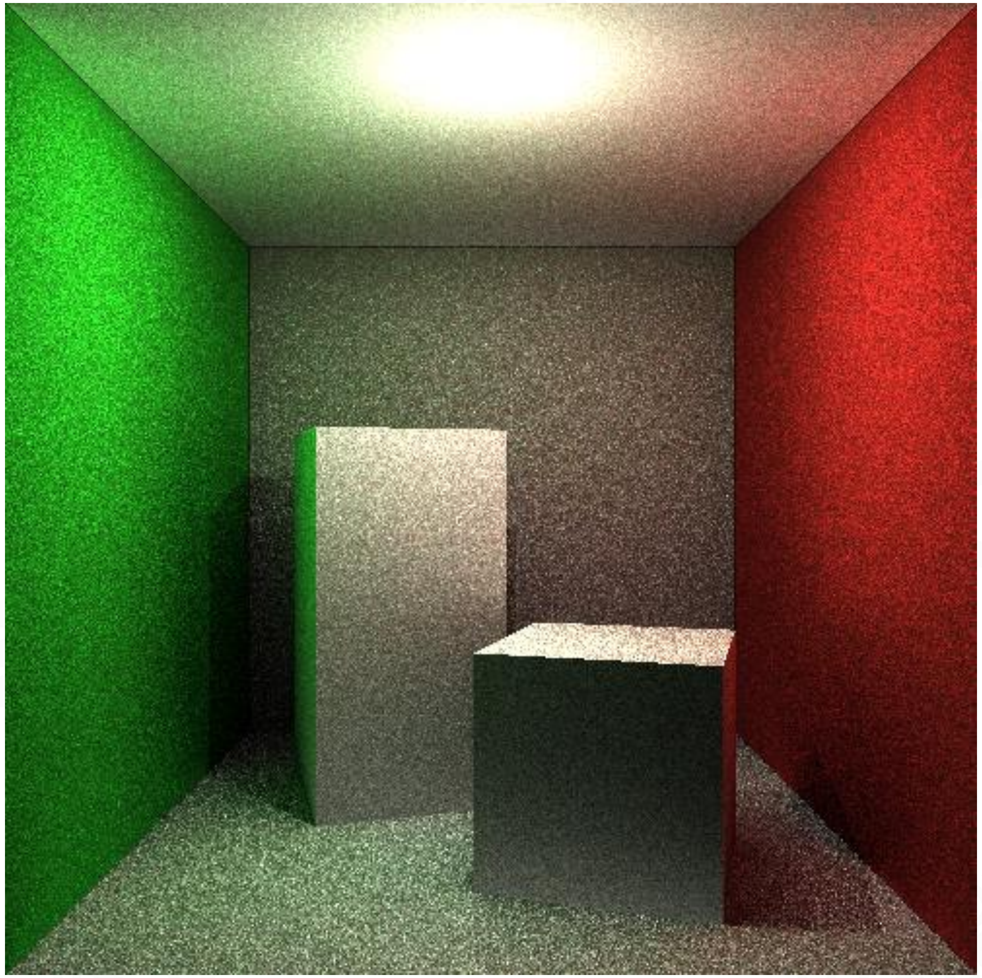
Monte Carlo Integration



- Approximate integral with the average of randomly sample values

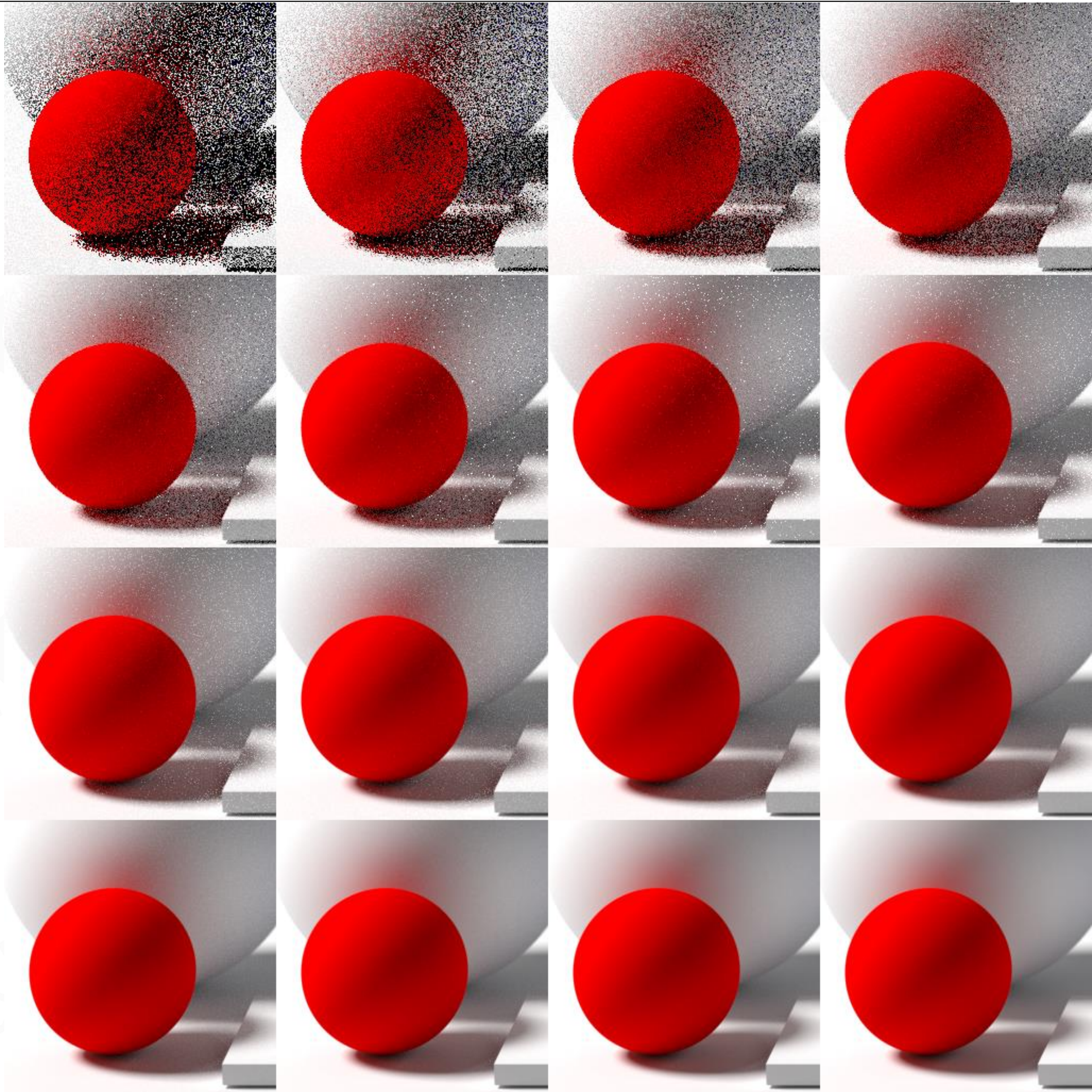
Monte Carlo Ray Tracing (Offline)



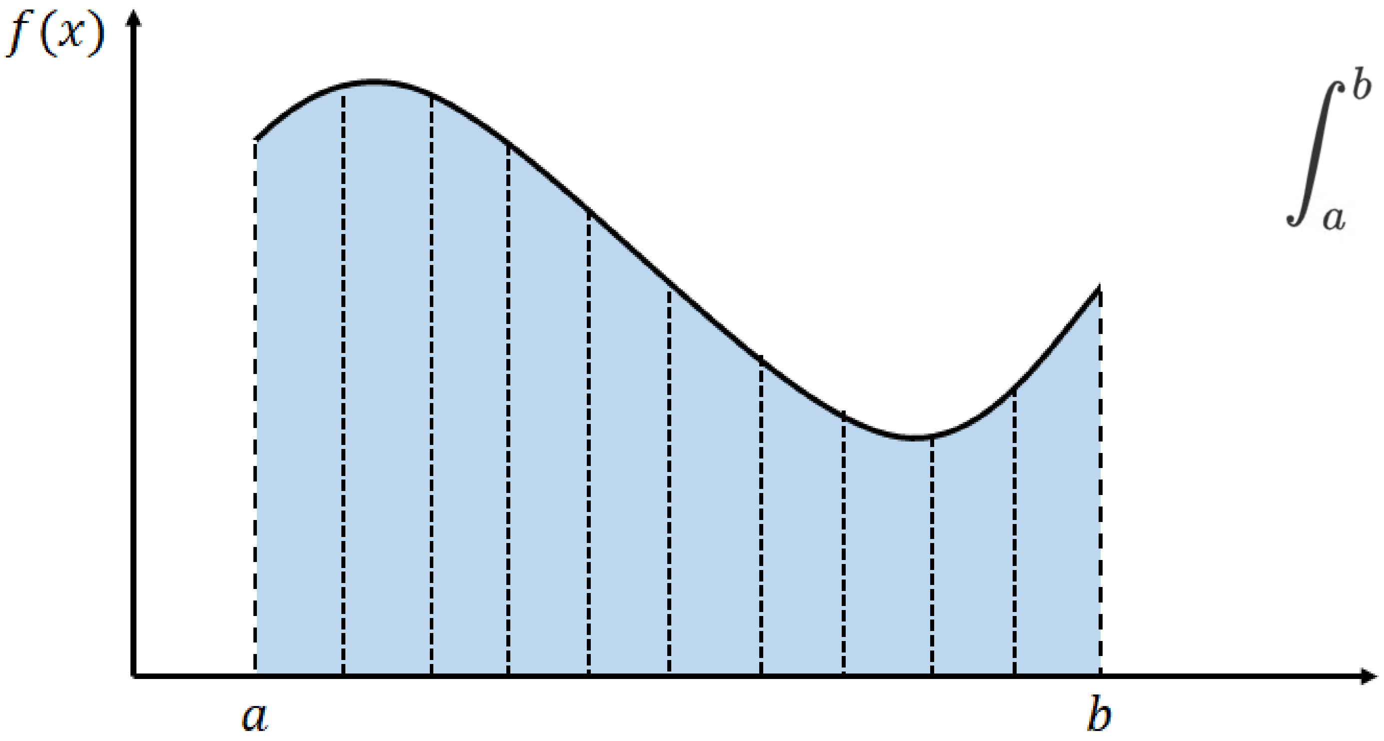


Sampling is the Key

- Noise decreases as the number of samples per pixel increases. The top left shows 1 sample per pixel, and doubles from left to right each square.



Sampling : Uniform Sampling



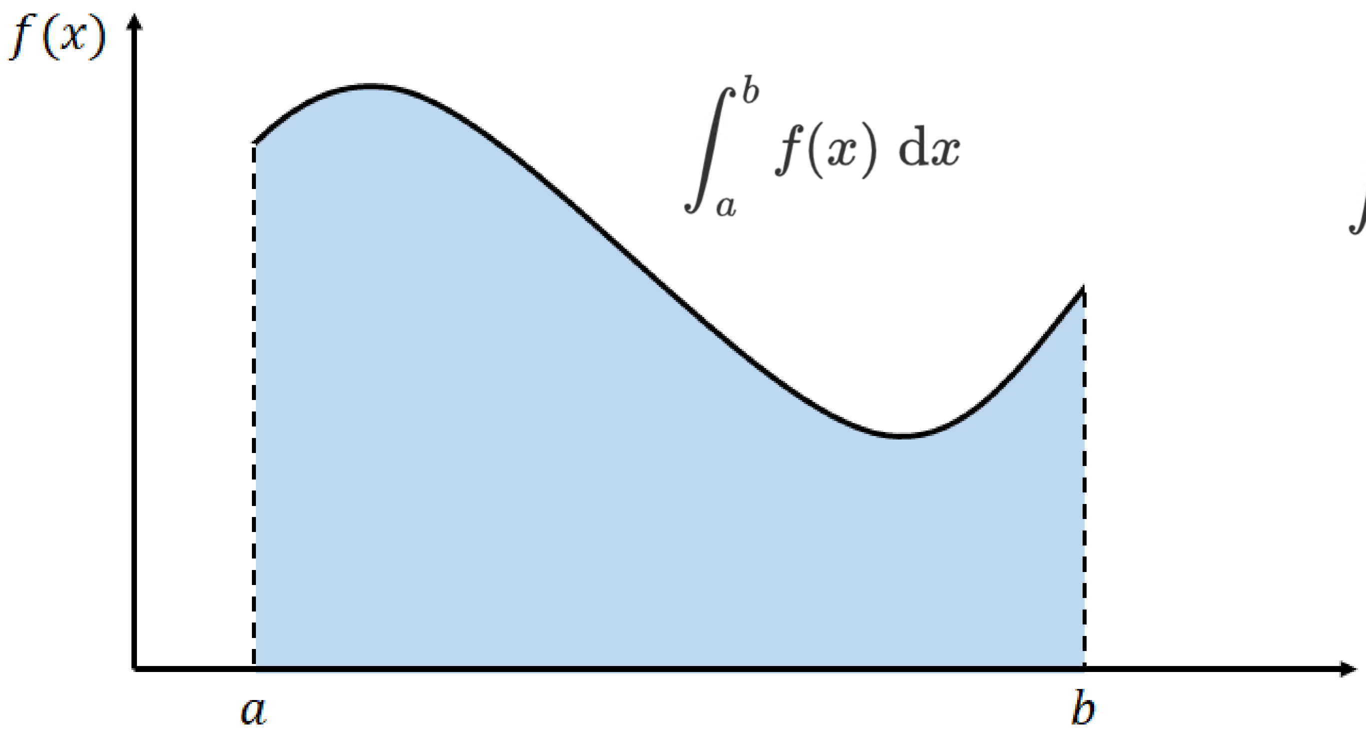
$$\int_a^b f(x)\,\mathrm{d}x = \lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} f(x_i)(b-a)$$

$$= \lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} \frac{f(x_i)}{\boxed{\frac{1}{b-a}}}$$

Probability Density Function

- We are doing uniform random sample, so we have $\frac{1}{b-a}$ factor here

Probability Distribution Function



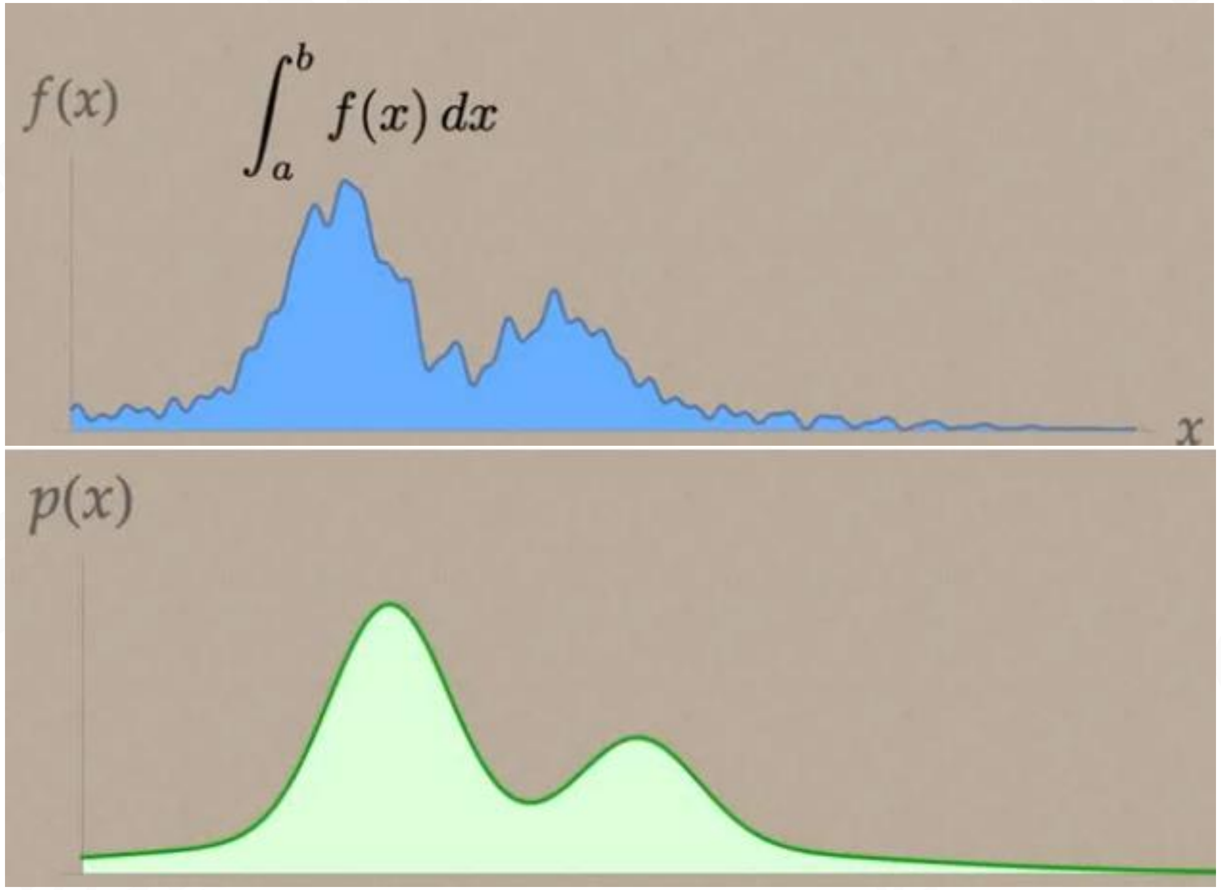
$$\int_a^b f(x)\mathrm{d}x \sim F_n(X) = \frac{1}{n}\sum_{k=1}^{n} \frac{f(X_k)}{PDF(X_k)}$$

Probability Distribution Function

- Describes the relative likehood for this random variable to take on a given value
- Higher means more possible to be chosen

Importance Sampling

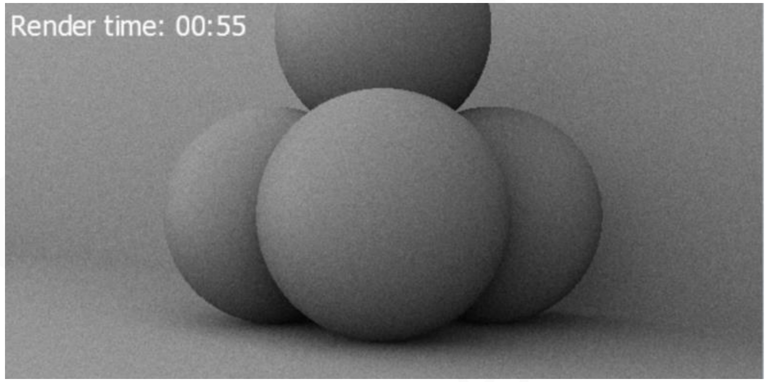
The PDF can be arbitrary, but which is the best?



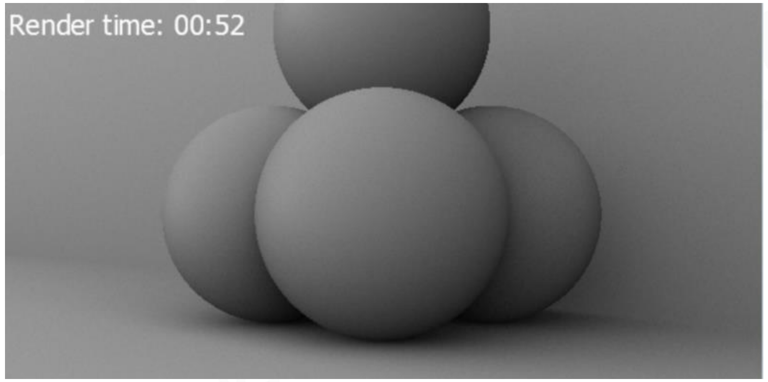
Importance Sampling : Best PDF for Rendering?

- Rendering equation: $$L_o(p, \omega_o) = \int_{\Omega^+} L_i(p, \omega_i) f_r(p, \omega_i, \omega_o)(n \cdot \omega_i) \, \mathrm{d}\omega_i$$
- Monte Carlo Integration: $$L_o(p, \omega_o) \approx \frac{1}{N} \sum_{i=1}^{N} \frac{L_i(p, \omega_i) f_r(p, \omega_i, \omega_o)(n \cdot \omega_i)}{p(\omega_i)}$$
- What's our f(x) ? $$L_i(p, \omega_i) f_r(p, \omega_i, \omega_o)(n \cdot \omega_i)$$
- What's our pdf ?
  - Uniform: $p(\omega_i) = \frac{1}{2\pi}$
  - Other pdf ? (cosine-weight, GGX)

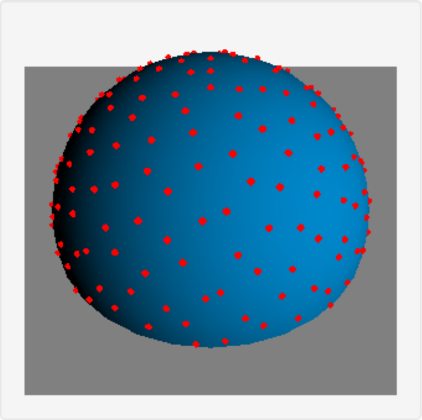
Importance Sampling : PDF is Matter

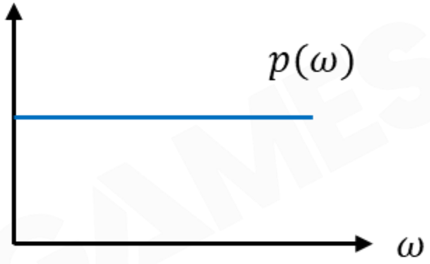


uniform sampling 256spp

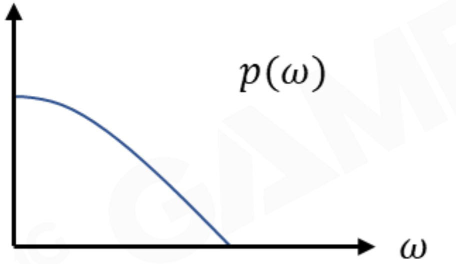


cosine weights importance sampling 256spp

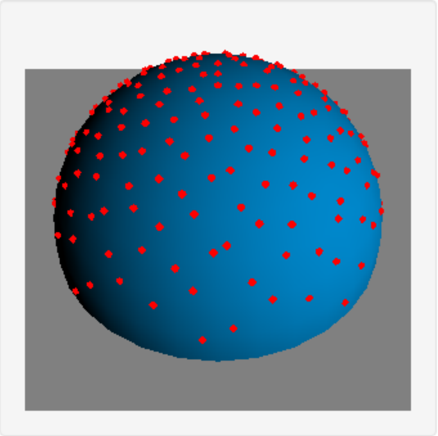




$$p(\omega) = \frac{1}{2\pi}$$

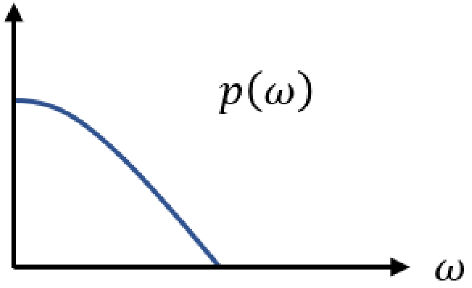


$$p(\omega) = \frac{\cos\theta}{\pi}$$

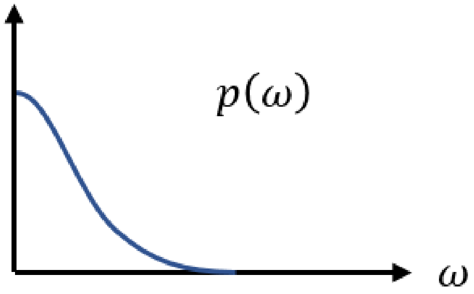


spp: samples per pixel

Importance Sampling： Cosine and GGX PDF



$$p(\omega) = \frac{\cos\theta}{\pi}$$

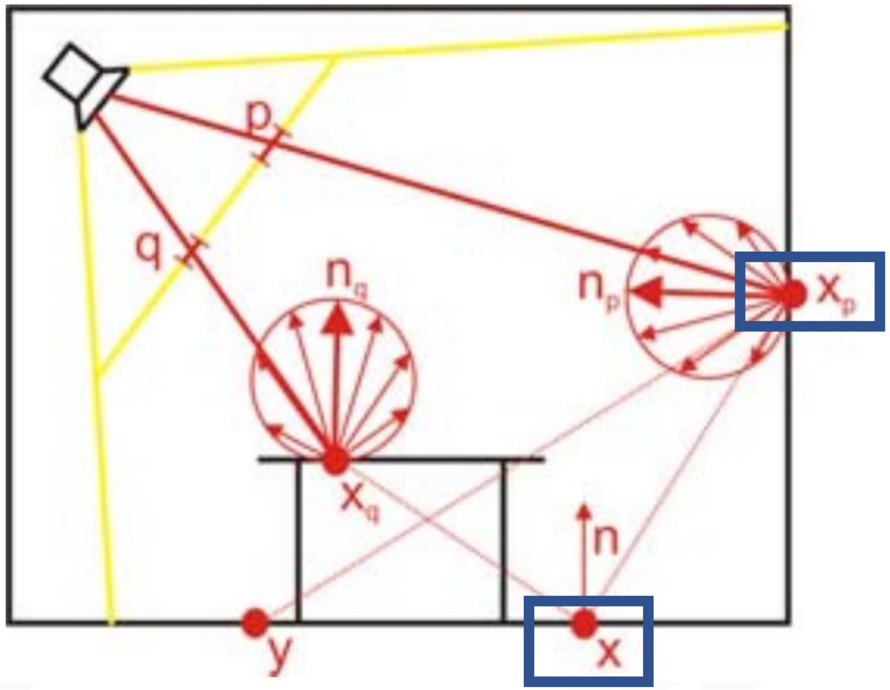


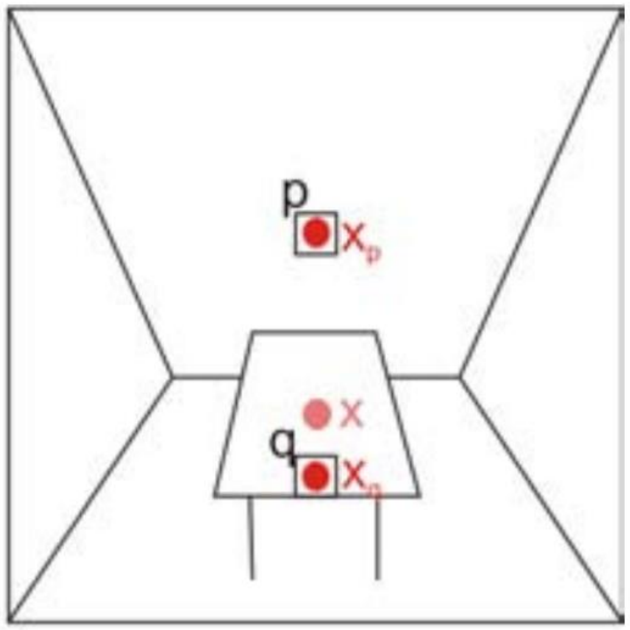
$$p(\omega) = \frac{\alpha^2 \cos\theta}{\pi((\alpha^2 - 1)\cos^2\theta + 1)^2}$$

Reflective Shadow Maps (RSM, 2005)

Let’s inject light in. (Photon Mapping?)

- Each pixel on the shadow map is a indirect light source





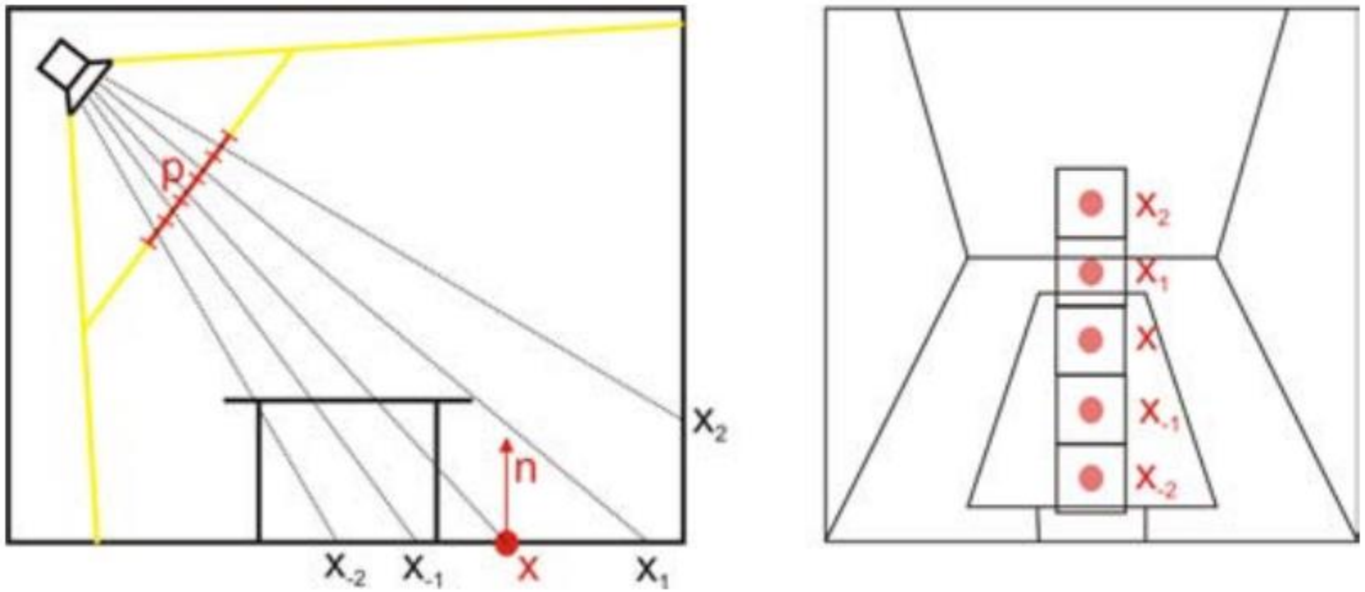
- How the RSM pixel X_p illuminates position x?

$$E_p(x,n) = \phi_p \frac{\max\{0,\langle n_p | x - x_p\rangle\}\max\{0,\langle n | x_p - x\rangle\}}{\| x - x_p \|^4}$$

- The indirect irradiance at a surface point x can be approximated by summing up the illumination due to all pixel lights.

- Do not consider occlusion.

$$E(x,n) = \sum_{\text{pixels}\,p} E_p(x,n)$$



Cone Tracing with RSM

- Gathering Indirect Illumination
  - random sampling RSM pixels
  - precompute such a sampling pattern and reuse it for all indirect light computations
    - 400 samples were sufficient
    - use Poisson sampling to obtain a more even sample distribution

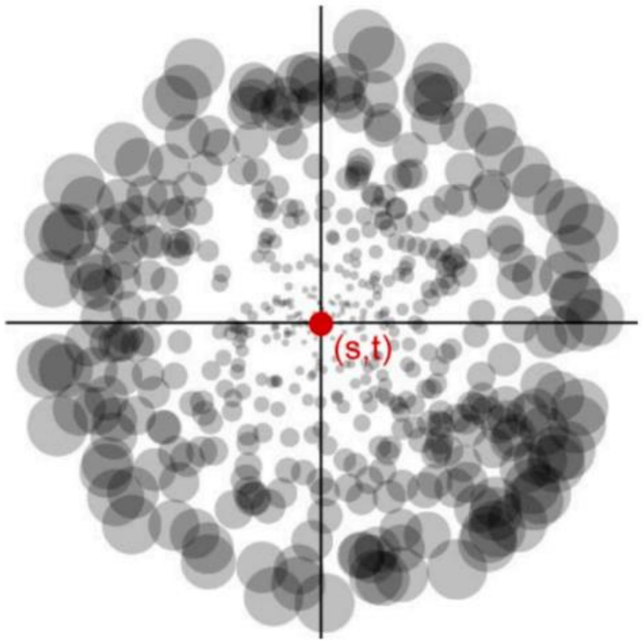
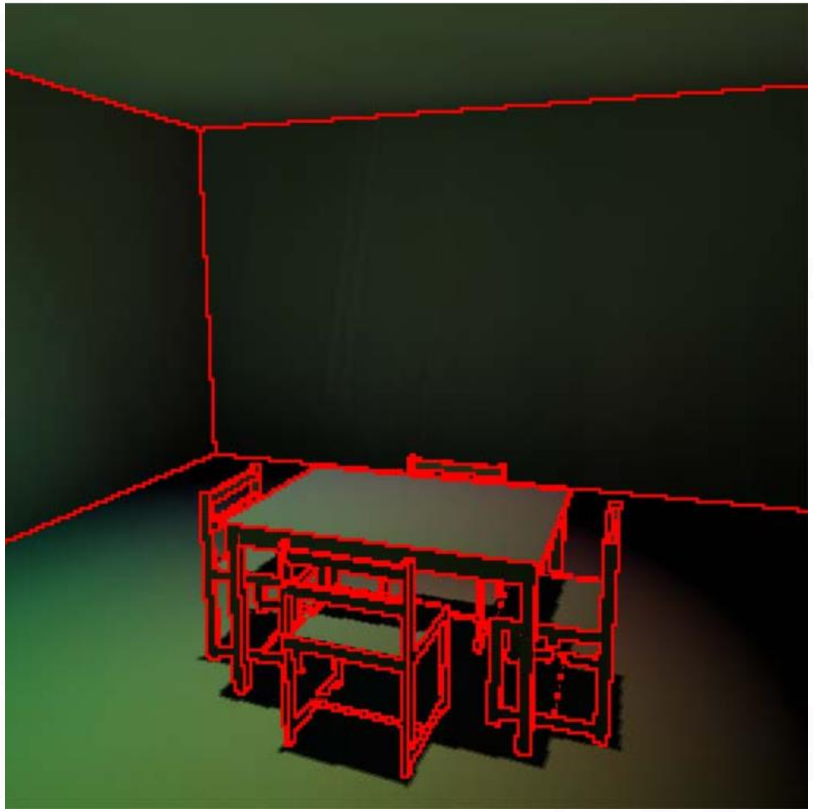


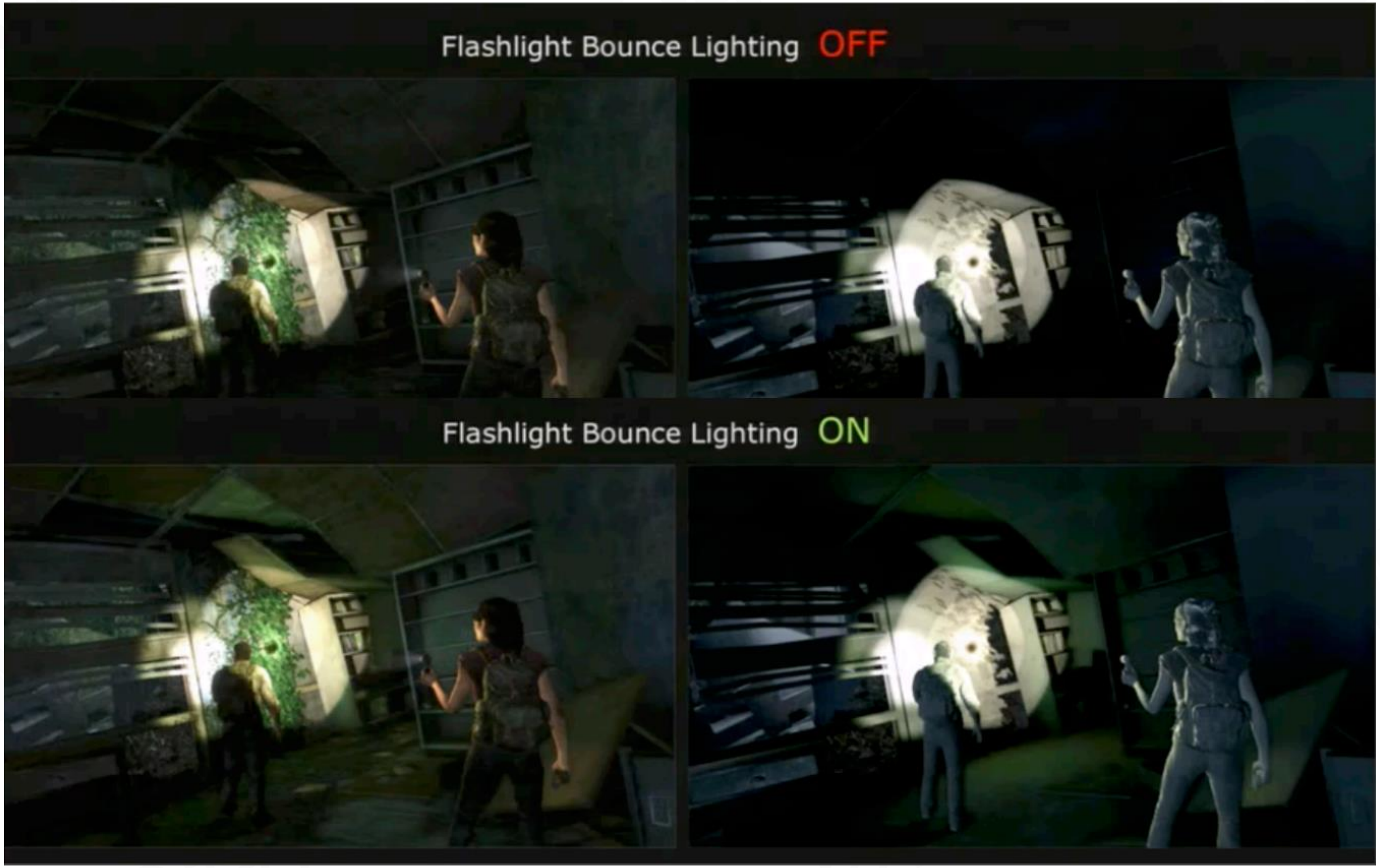
Figure 4: Sampling pattern example. The sample density decreases and the sample weights (visualized by the disk radius) increases with the distance to the center.

Acceleration with Low-Res Indirect Illumination

- Compute the indirect illumination for a low resolution image
- For each pixel on full resolution:
  - get its four surrounding low-res samples
  - validate by comparing normal and world space position
  - bi-linear interpolation
- Recompute the left (red pixels)



Gears of War 4, Uncharted 4, The Last of US, etc



Thanks, RSM

Cool Ideas

- Easy to be implemented
- Photon Injection with RSM
- Cone sampling in mipmap
- Low-res Indirect illumination with error check

Cons

- Single bounce
- No visibility check for indirect illumination

Reflective Shadow Maps

Carsten Dachsbacher*
University of Erlangen-Nuremberg

Marc Stamminger†
University of Erlangen-Nuremberg

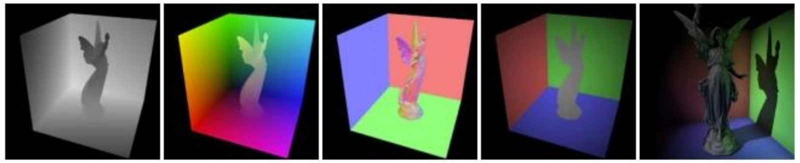


Figure 1: This figure shows the components of the reflective shadow map (depth, world space coordinates, normal, flux) and the resulting image rendered with indirect illumination from the RSM. Note that the angular decrease of flux is shown exaggerated for visualization.

Abstract

In this paper we present "reflective shadow maps", an algorithm for interactive rendering of plausible indirect illumination. A reflective shadow map is an extension to a standard shadow map, where every pixel is considered as an indirect light source. The illumination due to these indirect lights is evaluated on-the-fly using adaptive sampling in a fragment shader. By using screen-space interpolation of the indirect lighting, we achieve interactive rates, even for complex scenes. Since we mainly work in screen space, the additional effort is largely independent of scene complexity. The resulting indirect light is approximate, but leads to plausible results and is suited for dynamic scenes. We describe an implementation on current graphics hardware and show results achieved with our approach.

CR Categories: I.3.3 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and textureI.3.3 [Computer Graphics]: Hardware Architecture—Graphics processors

Keywords: indirect illumination, hardware-assisted rendering

1 Introduction

Interactive computer graphics has developed enormously over the last years, mainly driven by the advance of graphics acceleration hardware. Scenes of millions of polygons can be rendered in real-time on consumer-level PC cards nowadays. Programmability allows the inclusion of sophisticated lighting effects. However, these effects are only simple subcases of global illumination, e.g. reflections of distant objects or shadows of point lights. Real global illumination, however, generates subtle, but also important effects that are mandatory to achieve realism.

Unfortunately, due to their global nature, full global illumination and interactivity are usually incompatible. Ray Tracing and Radiosity—just to mention the two main classes of global illumination algorithms—require minutes or hours to generate a single image with full global illumination. Recently, there has been remarkable effort to make ray tracing interactive (e.g. [Wald et al. 2003]). Compute clusters are necessary to achieve interactivity at good image resolution and dynamic scenes are difficult to handle, because they require to update the ray casting acceleration structures for every frame. Radiosity computation times are even further from interactive. Anyhow, a once computed radiosity solution can be rendered from arbitrary view points quickly, but, as soons as objects move, the update of the solution becomes very expensive again.

It has been observed that for many purposes, global illumination solutions do not need to be precise, but only plausible. In this paper, we describe a method to compute a rough approximation for the one-bounce indirect light in a scene. Our method is based on the idea of the shadow map. In a first pass, we render the scene from the view of the light source (for now, we assume that we have only one spot or parallel light source in our scene). The resulting depth buffer is called *shadow map*, and can be used to generate shadows. In a *reflective shadow map*, with every pixel, we additionally store the light reflected off the hit surface. We interpret each of the pixels as a small area light source that illuminates the scene. In this paper, we describe how the illumination due to this large set of light sources can be computed efficiently and coherently, resulting in approximate, yet plausible and coherent indirect light.

2 Previous Work

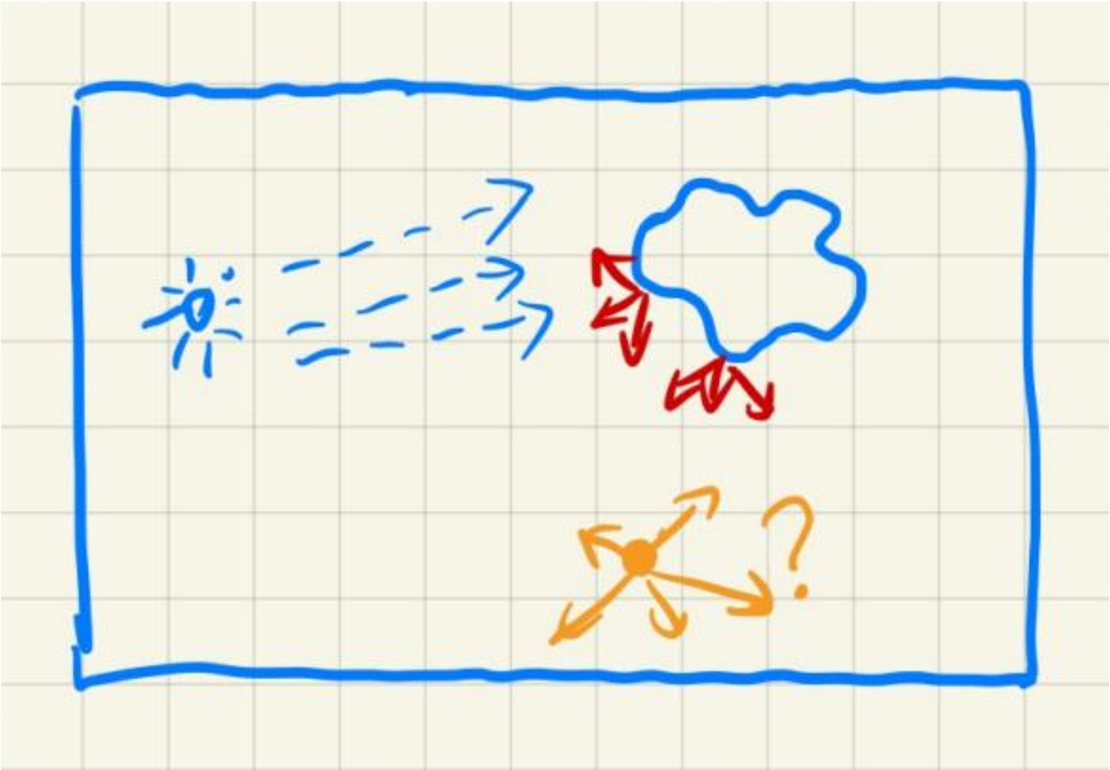
Shadow maps [Williams 1978; Reeves et al. 1987] and shadow volumes [Crow 1977] are the standard shadowing algorithms for interactive applications. Recently, there have been extensions of both approaches to area lights [Assarsson and Akenine-Möller 2003; Chan and Durand 2003; Wyman and Hansen 2003]. Sometimes, such soft shadows are already referred to as 'global illumination'. In this paper, we concentrate on indirect illumination from point lights, but our approach can easily be combined with any of these soft shadow techniques.

*e-mail: dachsbacher@cs.fau.de
†e-mail: stamminger@cs.fau.de

Light Propagation Volumes (LPV)

First introduced in CryEngine 3（SIGGRAPH 2009）

- Key Idea
  - Use a 3D grid to propagate radiance from directly illuminated surfaces to anywhere else



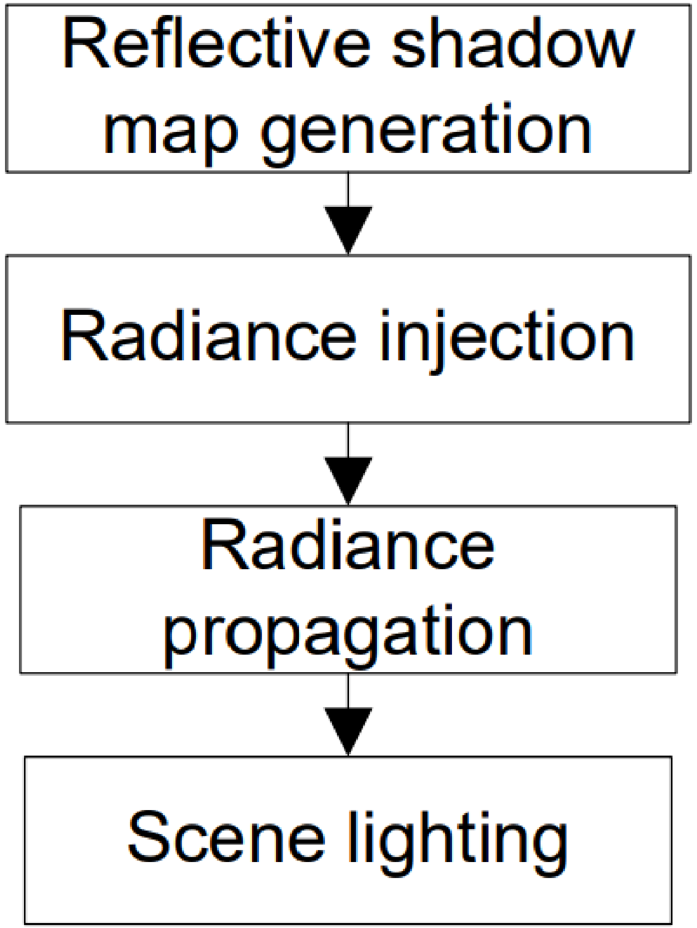
Steps

1. Generation of radiance point set scene representation

2. Injection of point cloud of virtual light sources into radiance volume

3. Volumetric radiance propagation

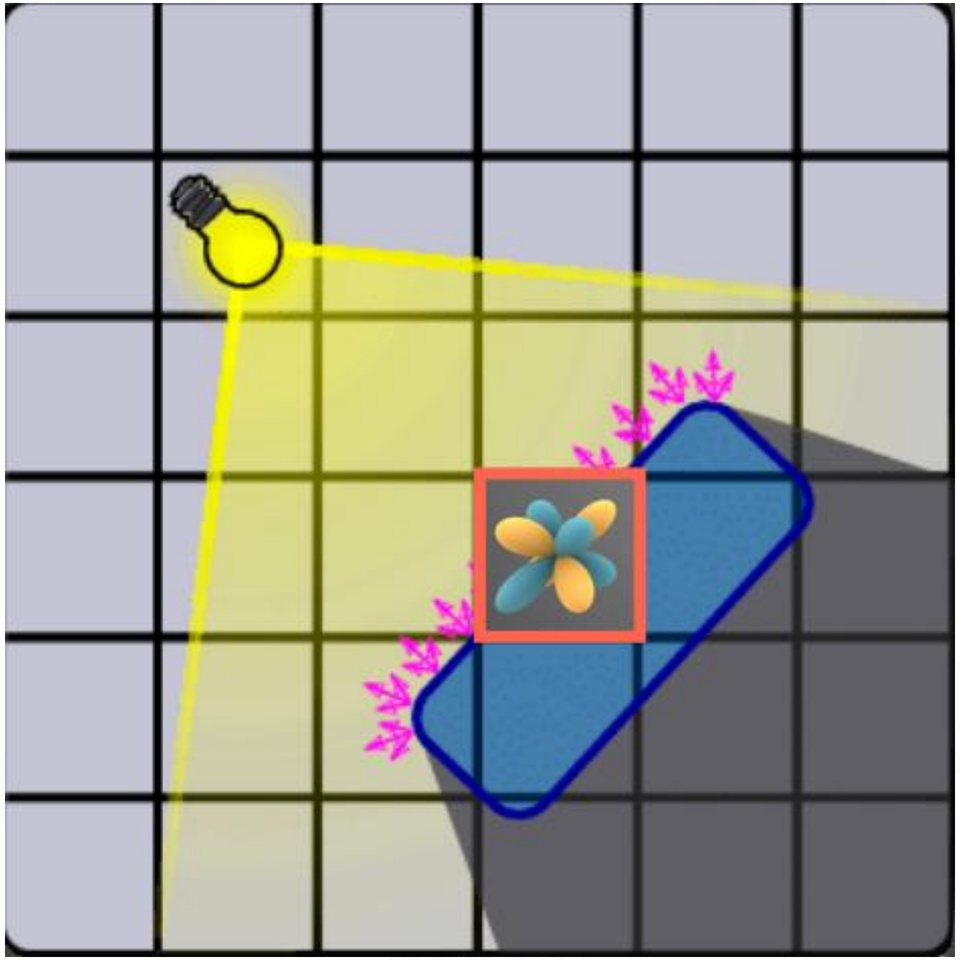
4. Scene lighting with final light propagation volume



"Freeze" the Radiance in Voxel

Light Injection

- Pre-subdivide the scene into a 3D grid
- For each grid cell, find enclosed virtual light sources
- Sum up their directional radiance distribution
- Project to first 2 orders of SHs (4 in total)



Radiance Propagation

- For each grid cell, collect the radiance received from each of its 6 faces
- Sum up, and again use SH to represent
- Repeat this propagation several times till the volume becomes stable

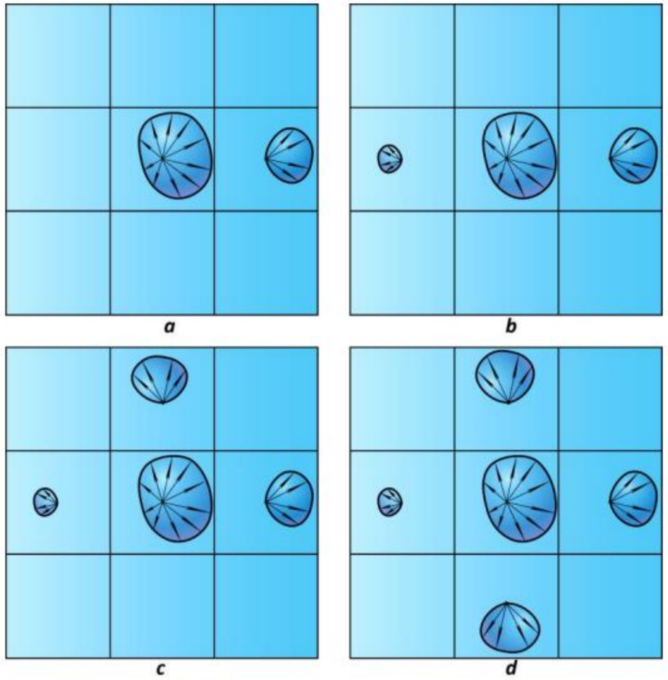
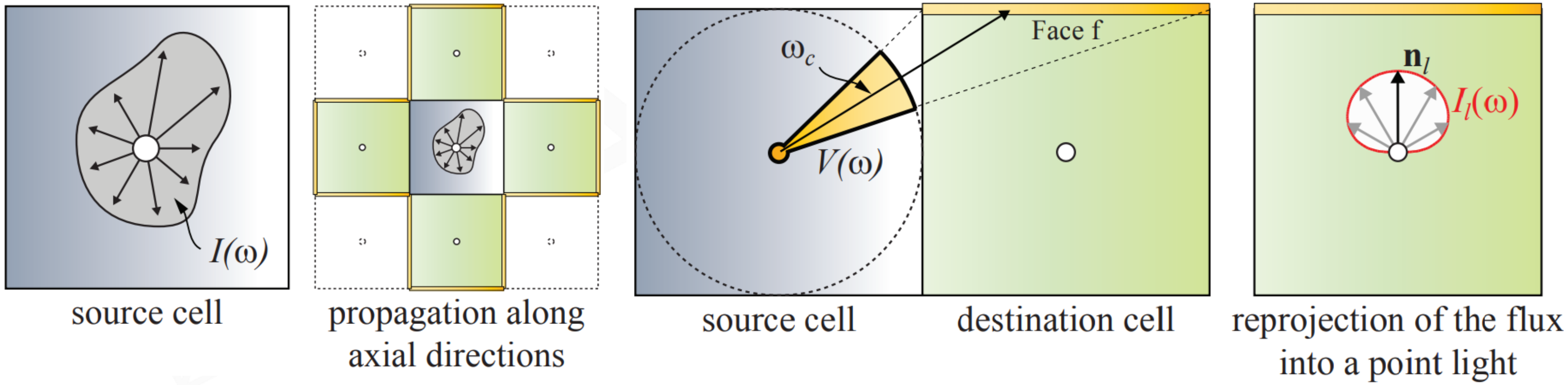
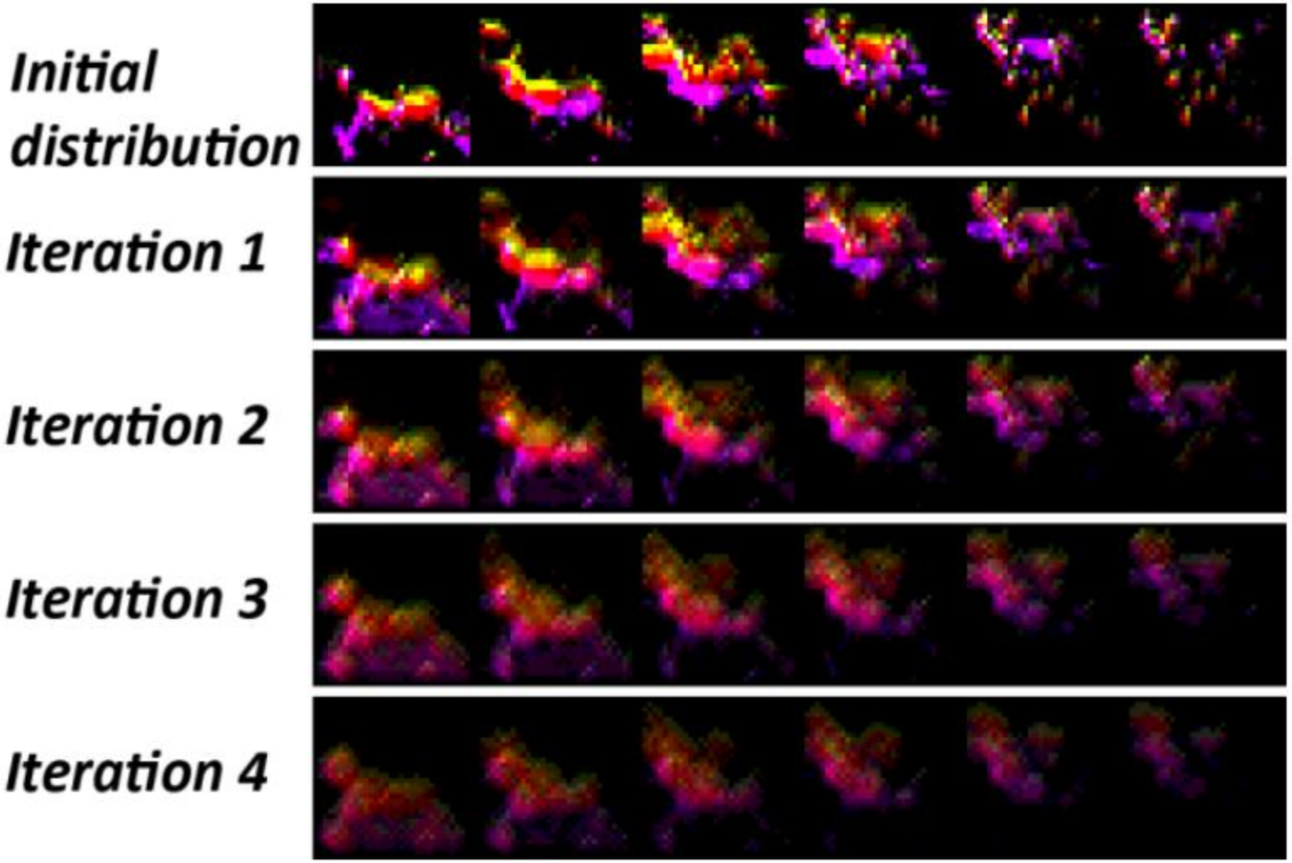


Figure 4. Radiance propagation iteration

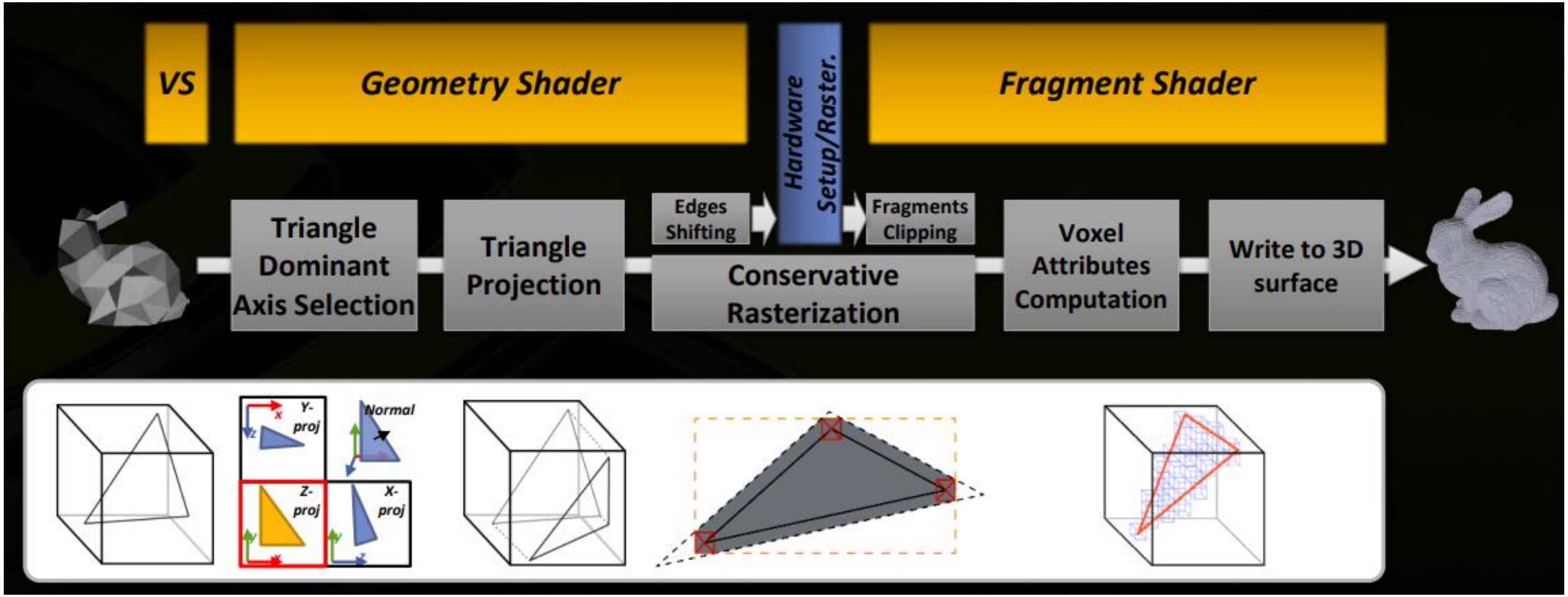


Light with “Limit Speed”?

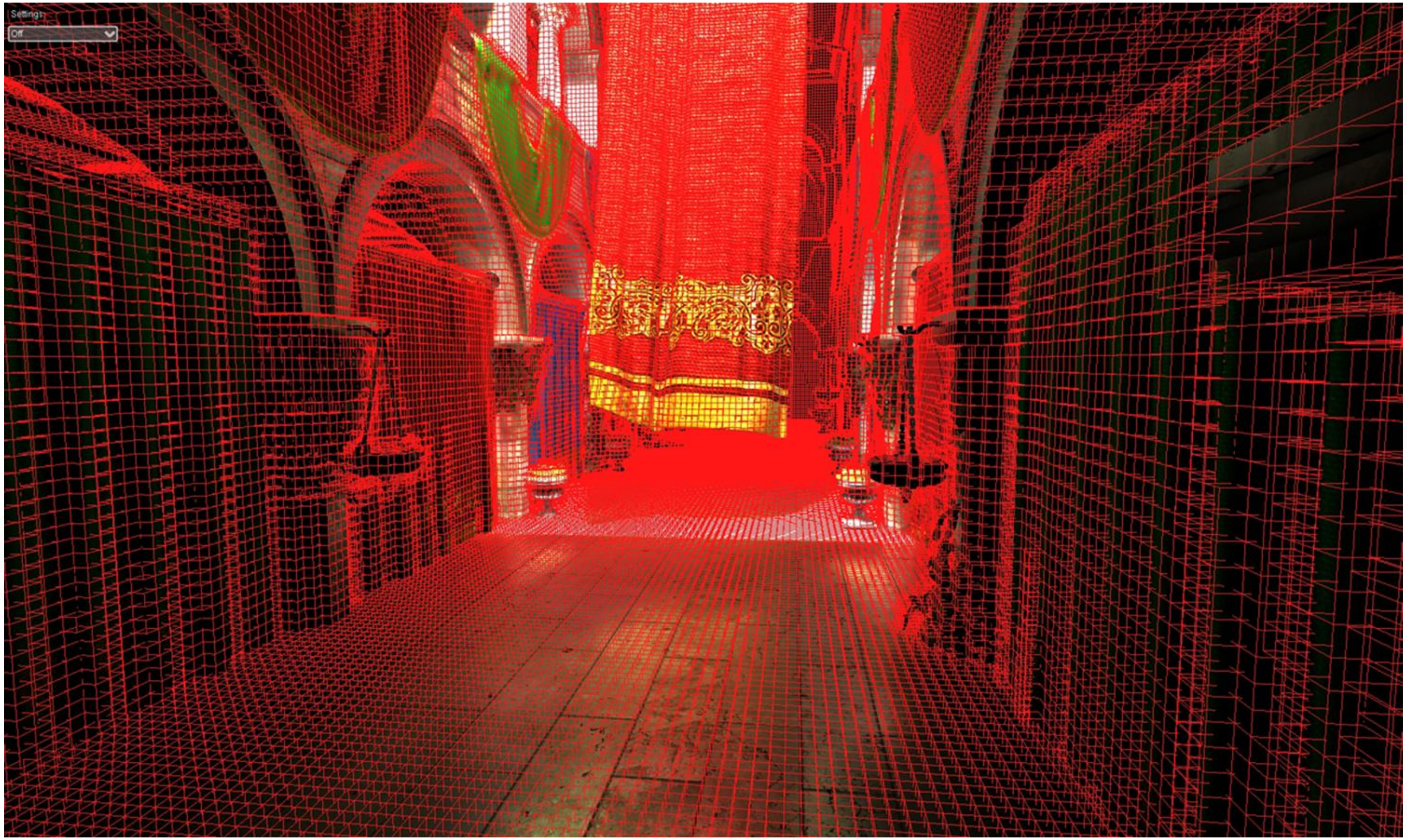


Sparse Voxel Octree for Real-time Global Illumination (SVOGI)

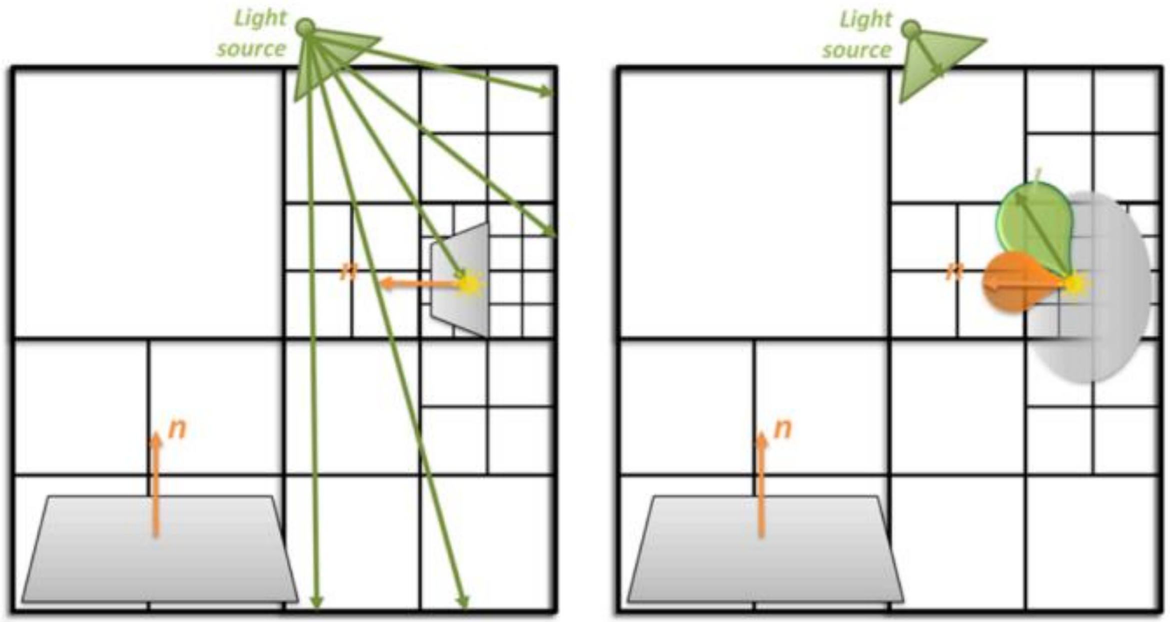
Voxelization Pass



Collect Surface Voxels

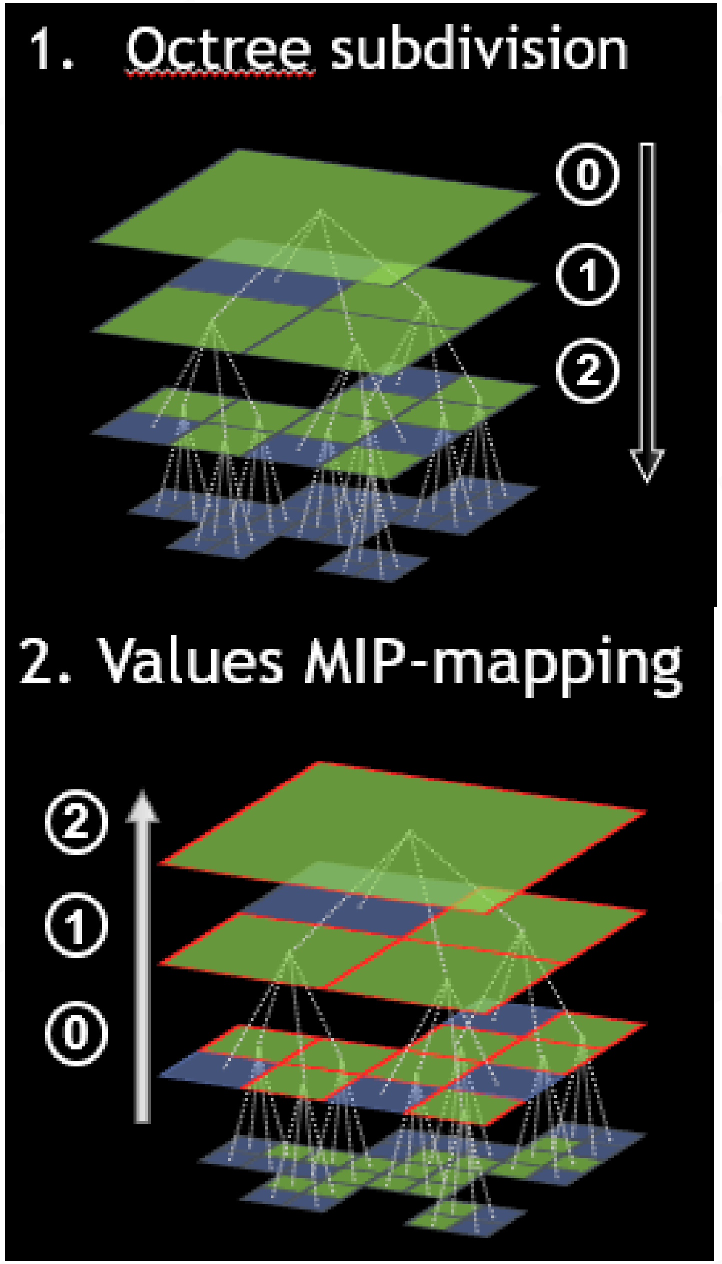


- Inject Irradiance into voxels from light
- Filter irradiance inside the octree



Step 1: Render from light sources. Bake incoming radiance and light direction into the octree

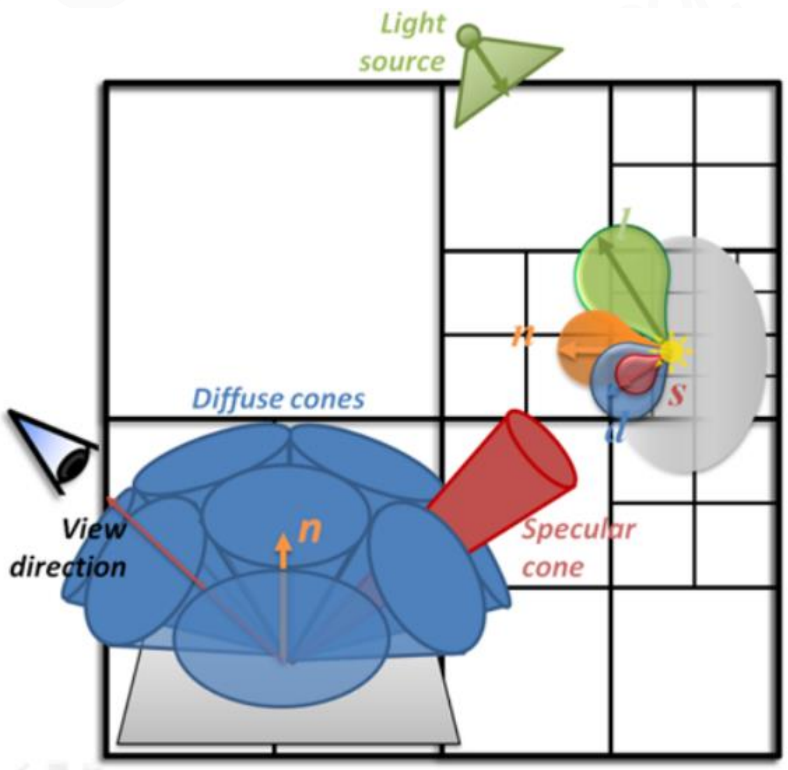
Step 2: Filter irradiance values and light directions inside the octree

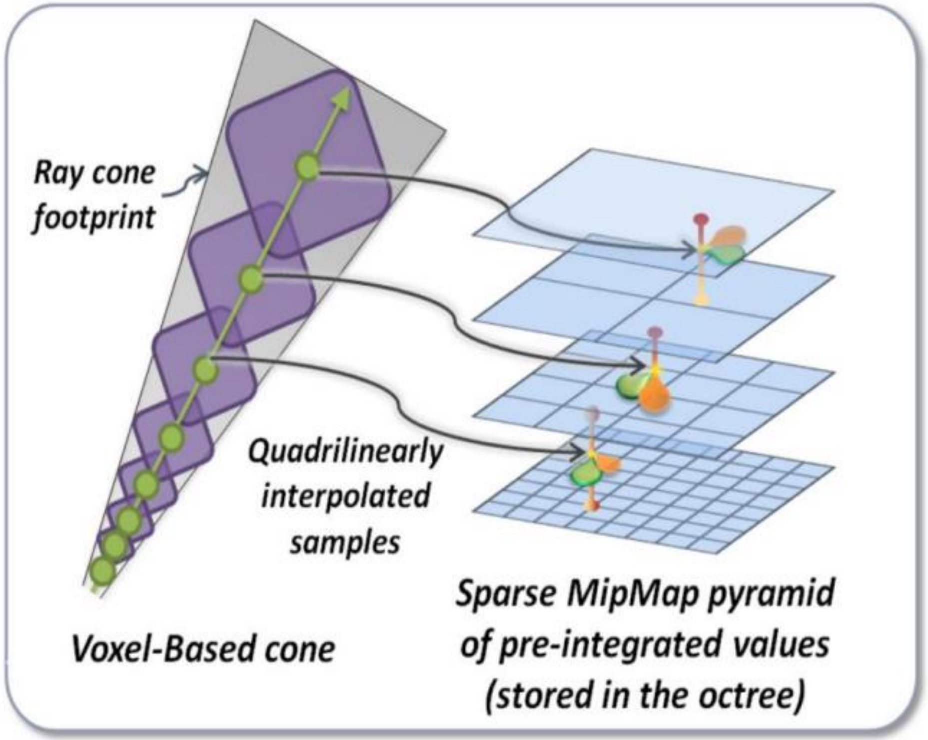


Shading with Cone Tracing in Voxel Tree

Pass 2 from the camera

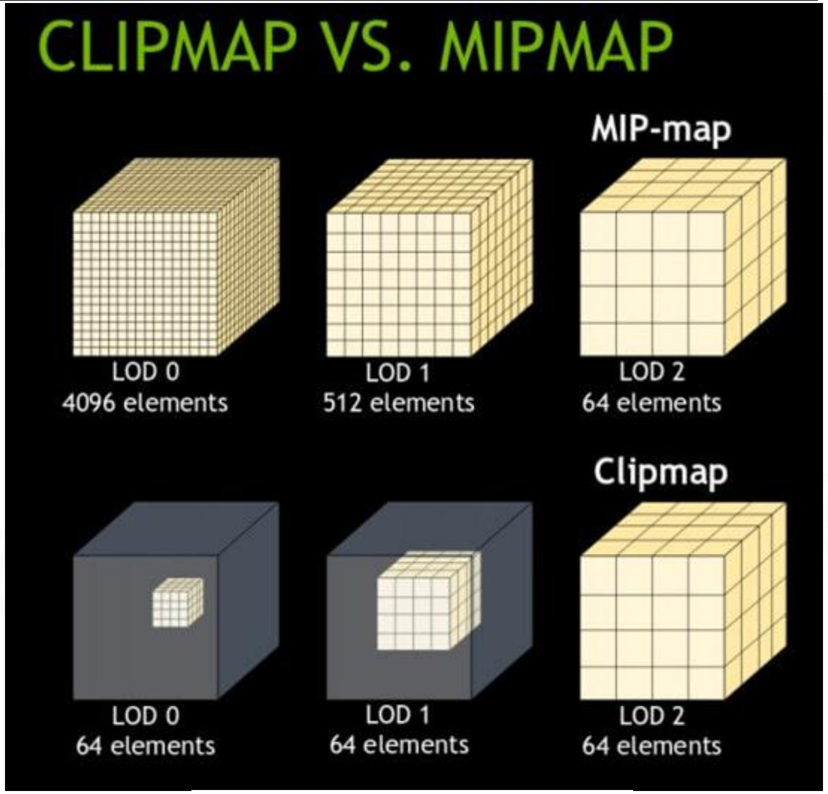
- Emit some cones based on diffuse+specular BRDF
- Query in octree based on the (growing) size of the cone

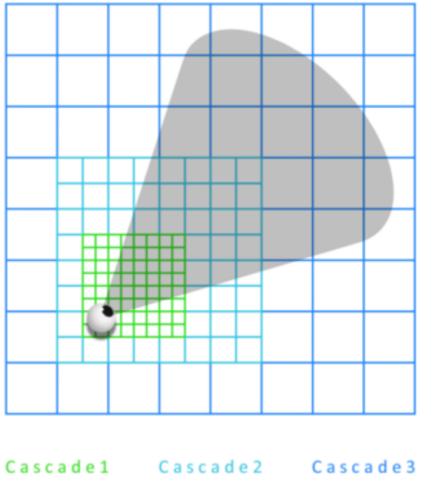




Voxelization Based Global Illumination (VXGI)

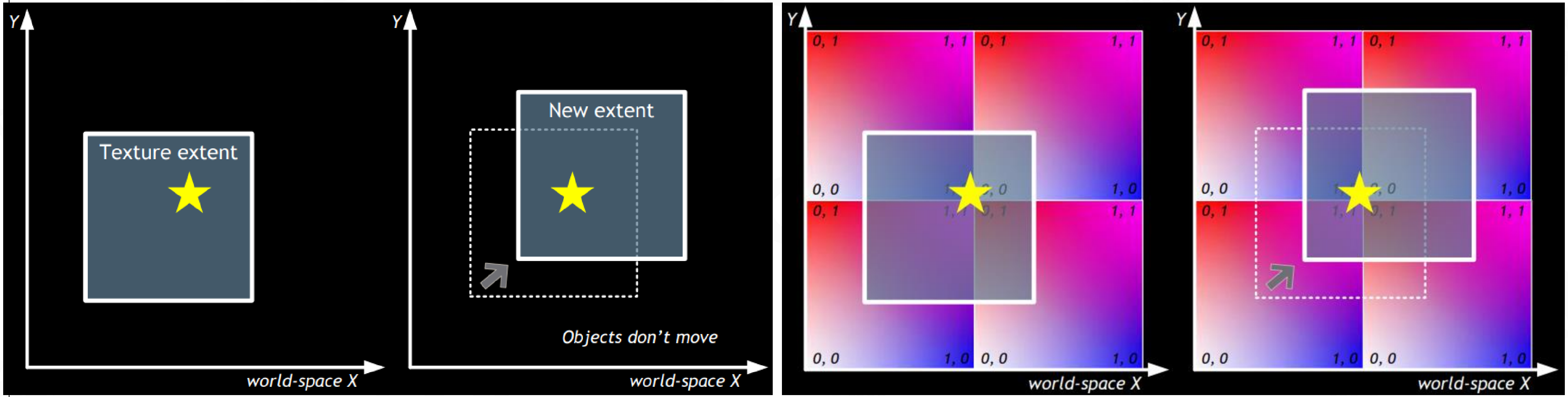
- Store the voxel data in clipmaps
  - Multi-resolution texture
  - Regions near the center have higher spatial resolution
  - Seems to map naturally to cone tracing needs
- A clipmap is easier to build than SVO
  - No nodes, pointers etc., handled by hardware
- A clipmap is easier to read from
- Clipmap size is (64…256)^3 with 3…5 levels of detail
  - 16…32 bytes per voxel => 12 MB … 2.5 GB of video memory required

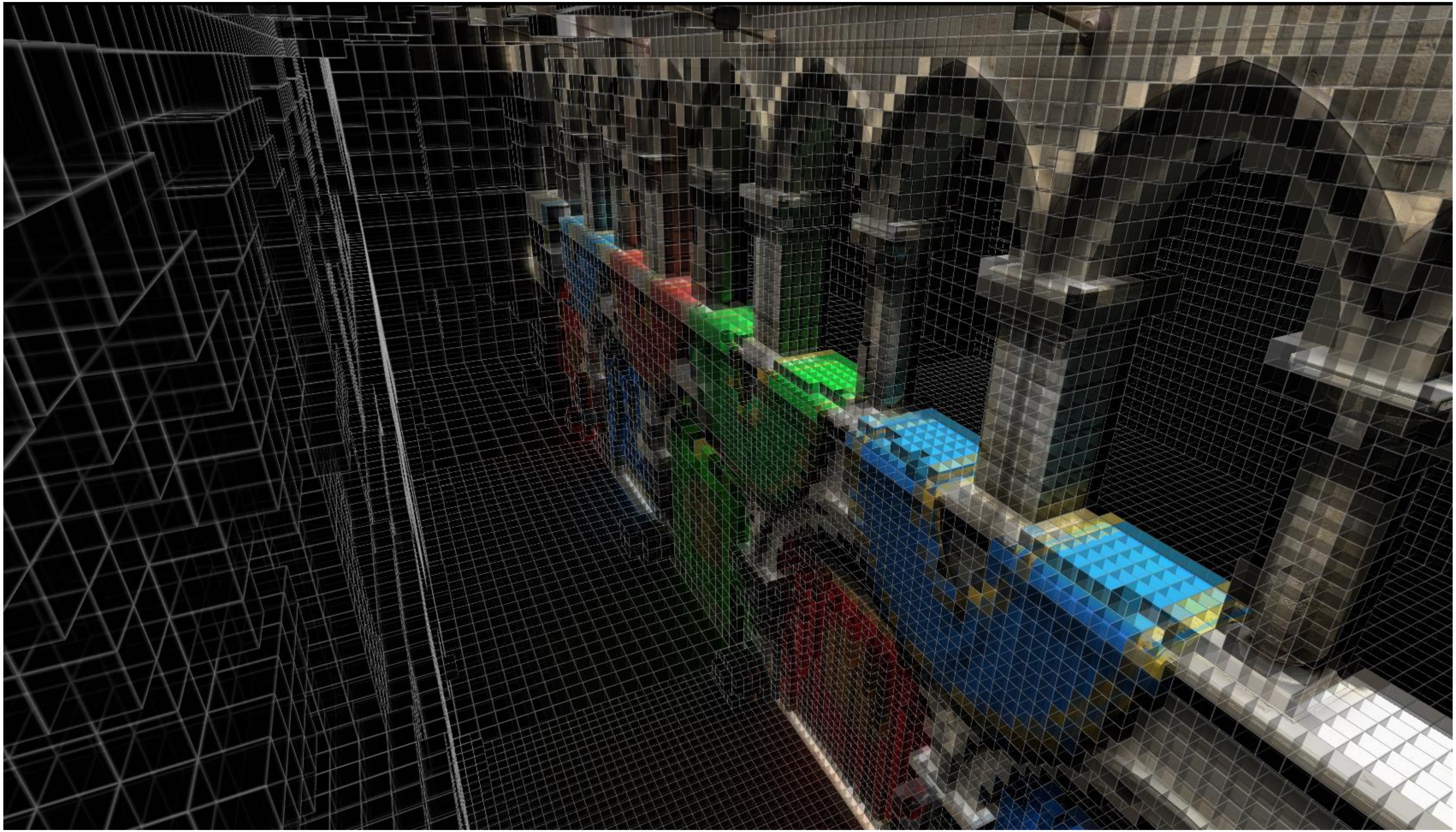




Voxel Update and Toroidal Addressing

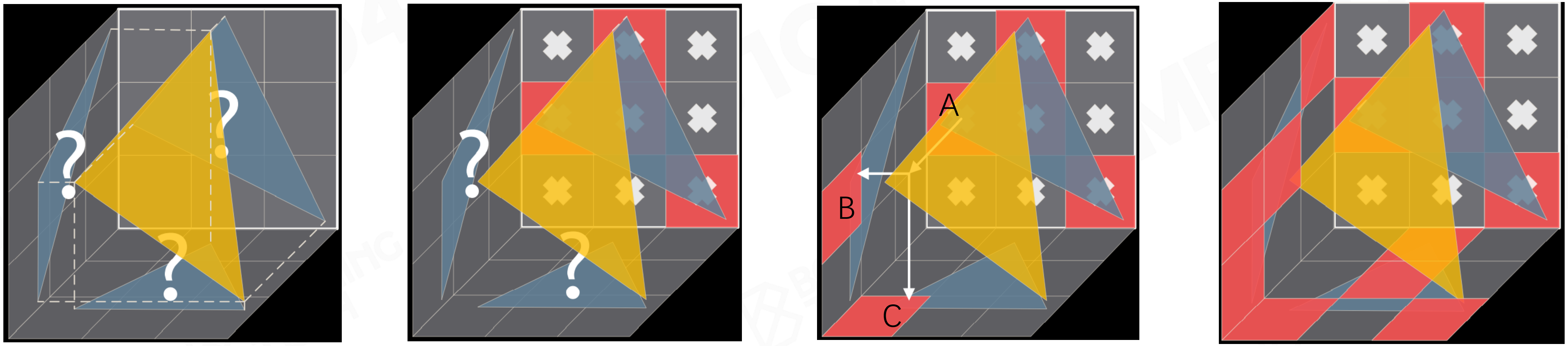
- A fixed point in space always maps to the same address in the clipmap
- The background shows texture addresses: frac(worldPos.xy / clipmapSize.xy)





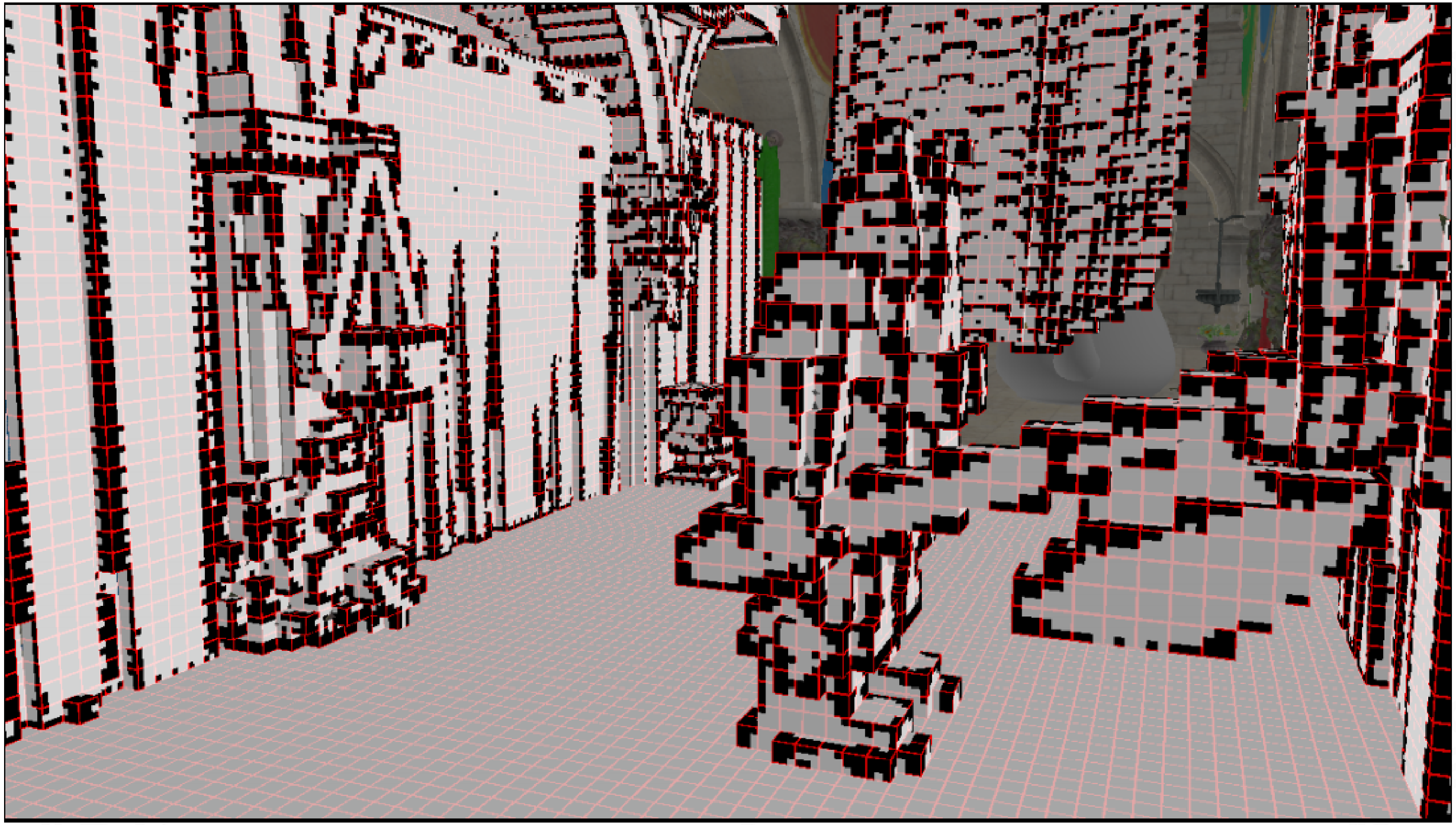
Voxelization for Opacity

- We have a triangle and a voxel
- Select the projection plane that yields the biggest projection area
- Rasterize the triangle using MSAA to compute one coverage mask per pixel
- Take the MSAA samples and reproject them onto other planes
- Repeat that process for all covered samples
- Thicken the result by blurring all the reprojected samples



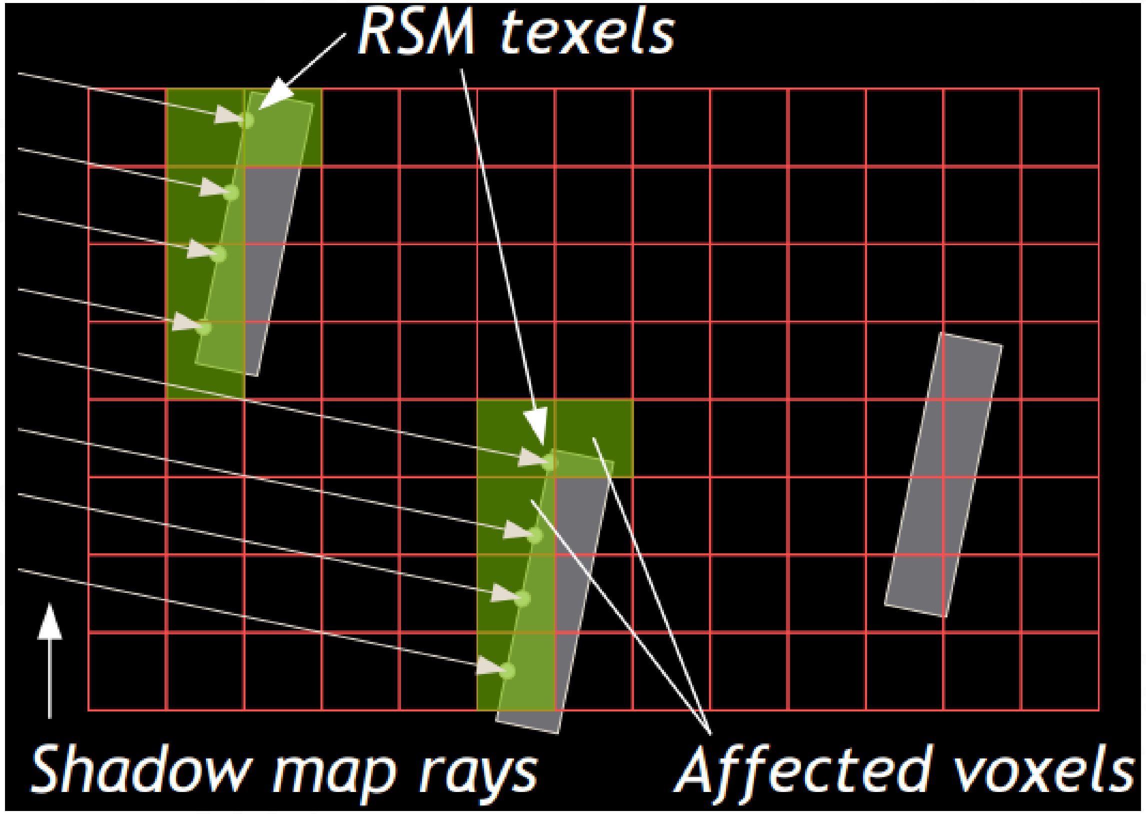
Opacity = (number of the covered MSAA samples) / MSAA_Resolution^2

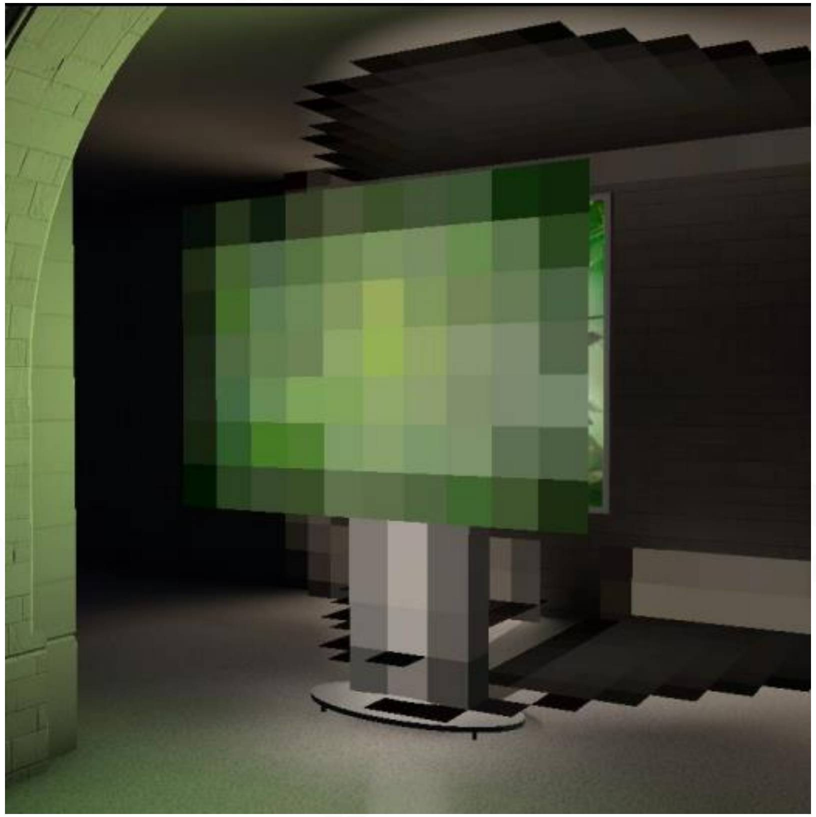
Voxelization: Directional Coverage

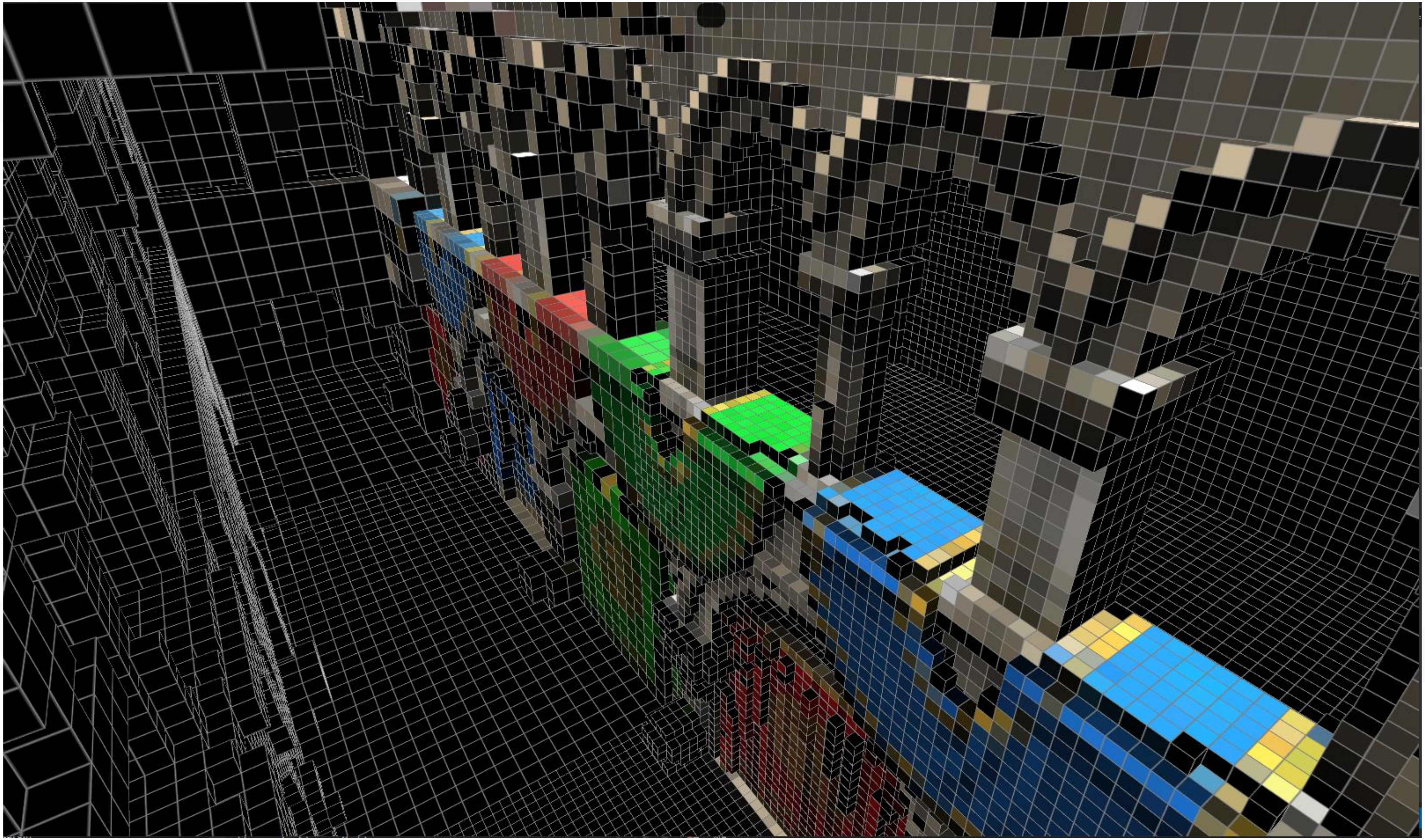


Light Injection

- Calculate emittance of voxels that contain surfaces lit by direct lights
- Take information from reflective shadow maps (RSM)



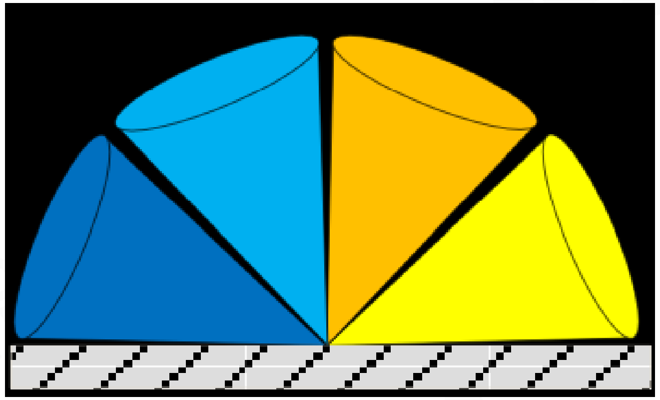


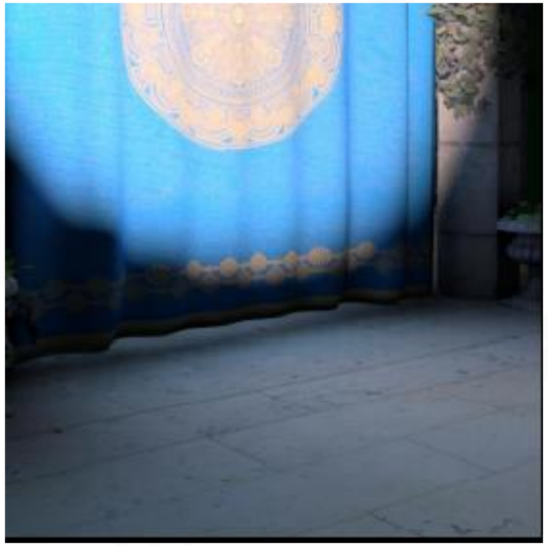


Shading with Cone Tracing

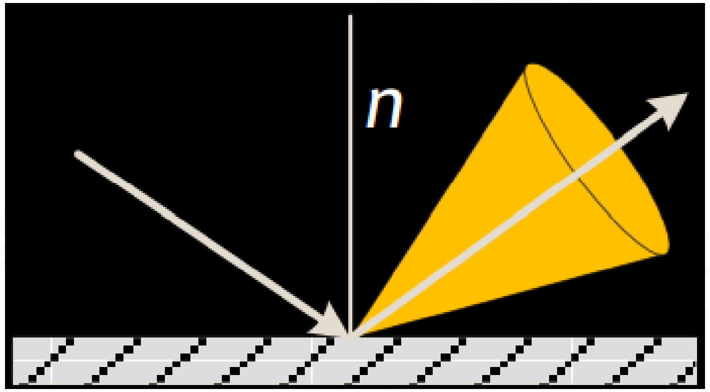
- generate several cones based on BRDF

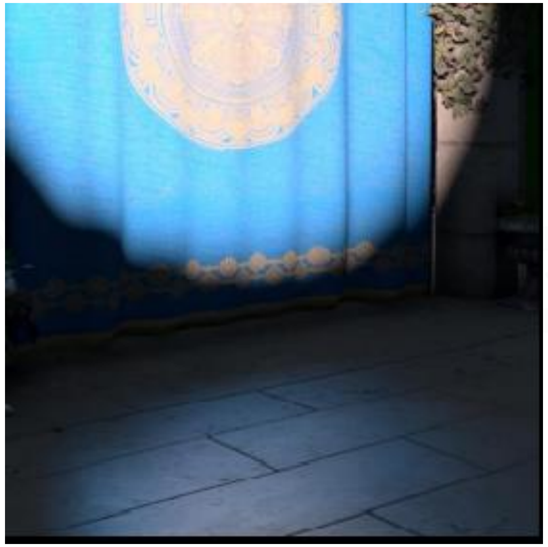
Diffuse



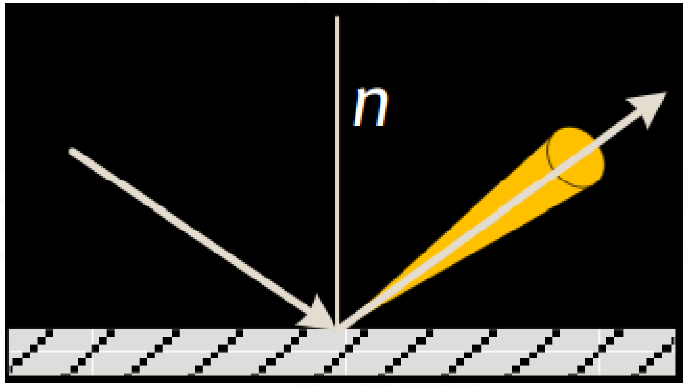


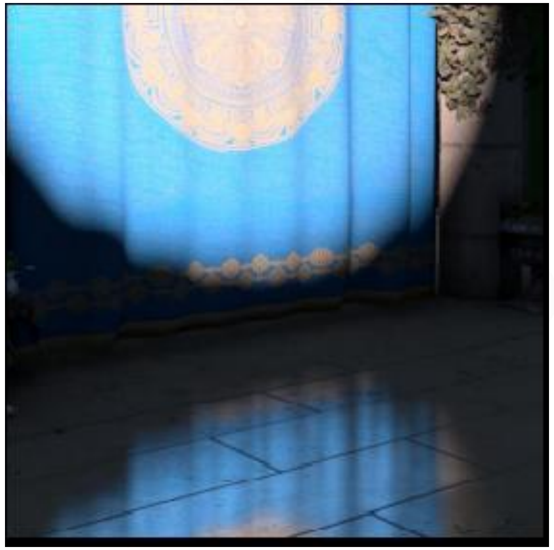
Rough Specular





Fine Specular

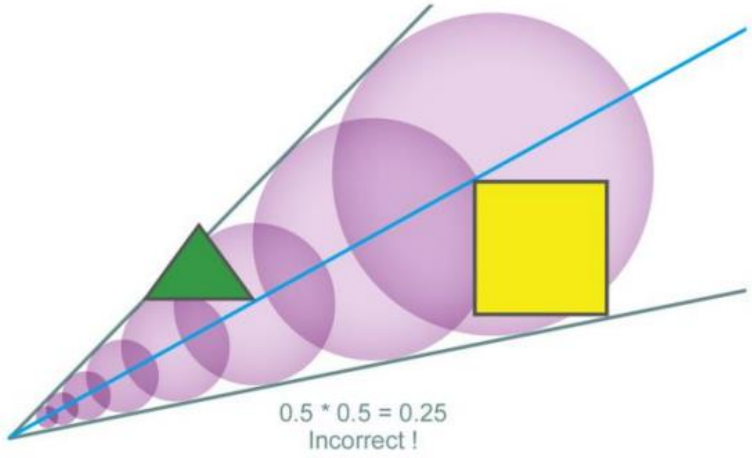




Accumulate Voxel Radiance and Opacity along the Path

$$C_{\text{dst}} \leftarrow C_{\text{dst}} + (1 - \alpha_{\text{dst}})C_{\text{src}}$$
$$\alpha_{\text{dst}} \leftarrow \alpha_{\text{dst}} + (1 - \alpha_{\text{dst}})\alpha_{\text{src}}$$

Problems in VXGI

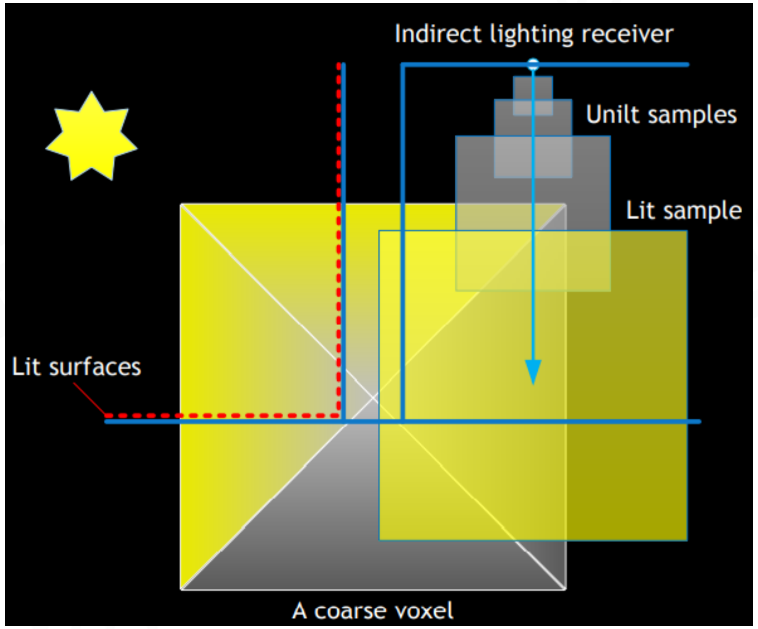


Incorrect Occlusion(opacity)

- naively combine the opacity with alpha blending.

Light Leaking

- when occlusion wall is much smaller than voxel size



Screen Space Global Illumination (SSGI)

SIGGRAPH2015
Xroads of Discovery

FROSTBITE™

empowers game creators to shape the future of gaming

SIGGRAPH 2015: Advances in Real-Time Rendering course

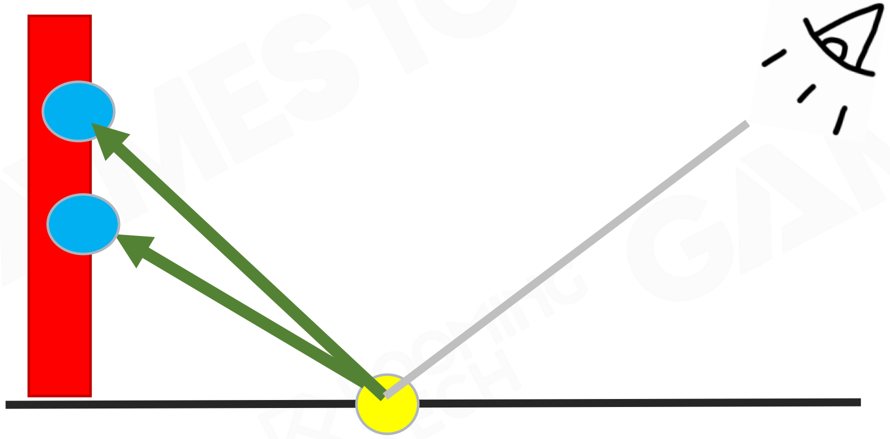
General Idea

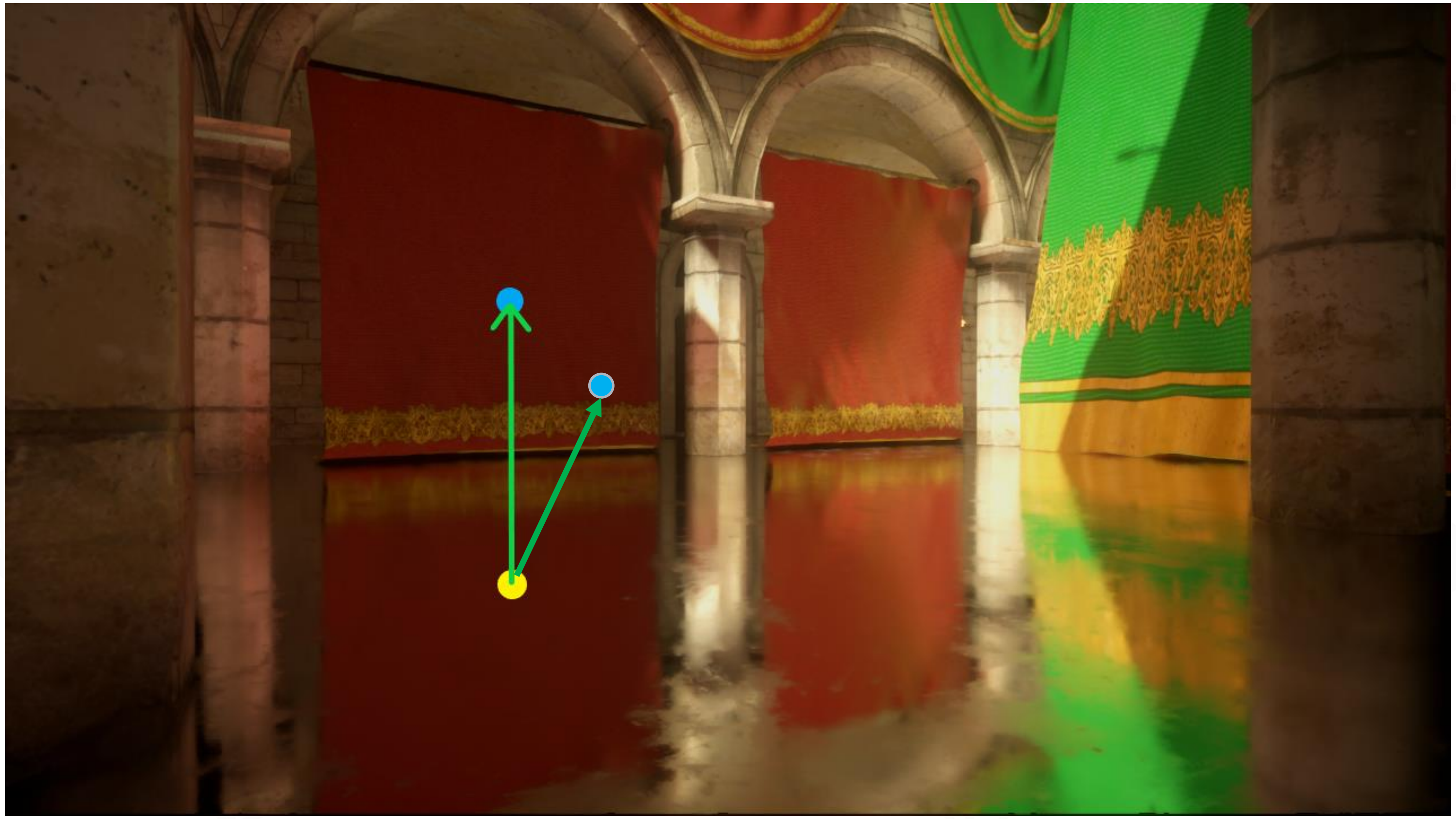
- Reuse screen-space data

Radiance Sampling in Screen Space

For each fragment:

- **Step 1:** compute many reflection rays
- **Step 2:** march along ray direction **(in depth gbuffer)**
- **Step3:** use color of hit point as indirect lighting

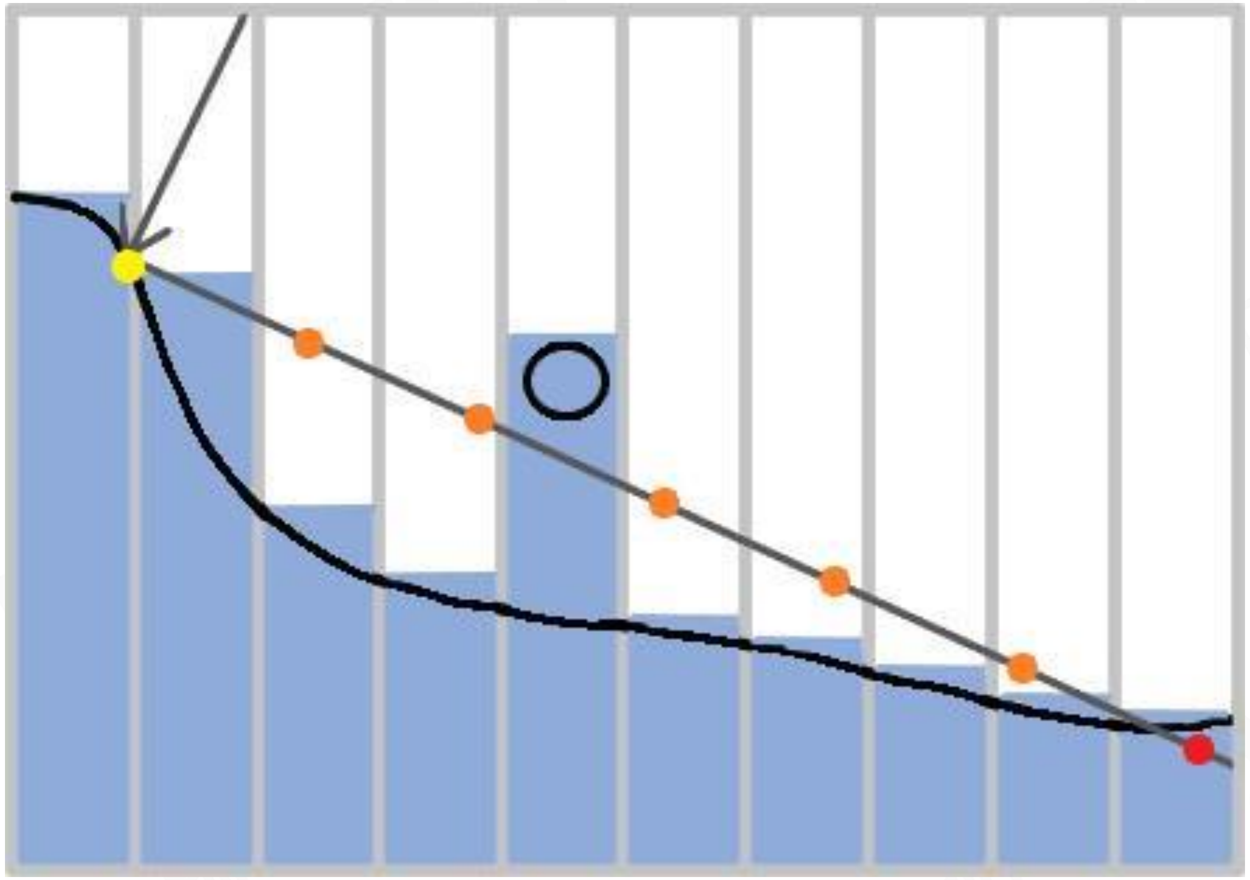




Linear Raymarching

- **General Steps**
  - Step forward at a fixed step size
  - At each step, check depth value

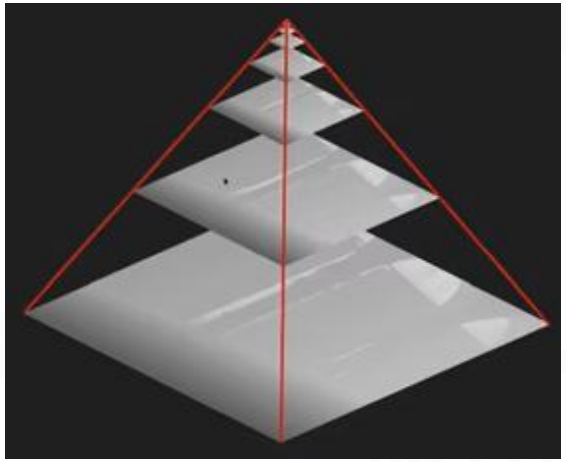
- **Features**
  - Fast
  - May skip thin objects



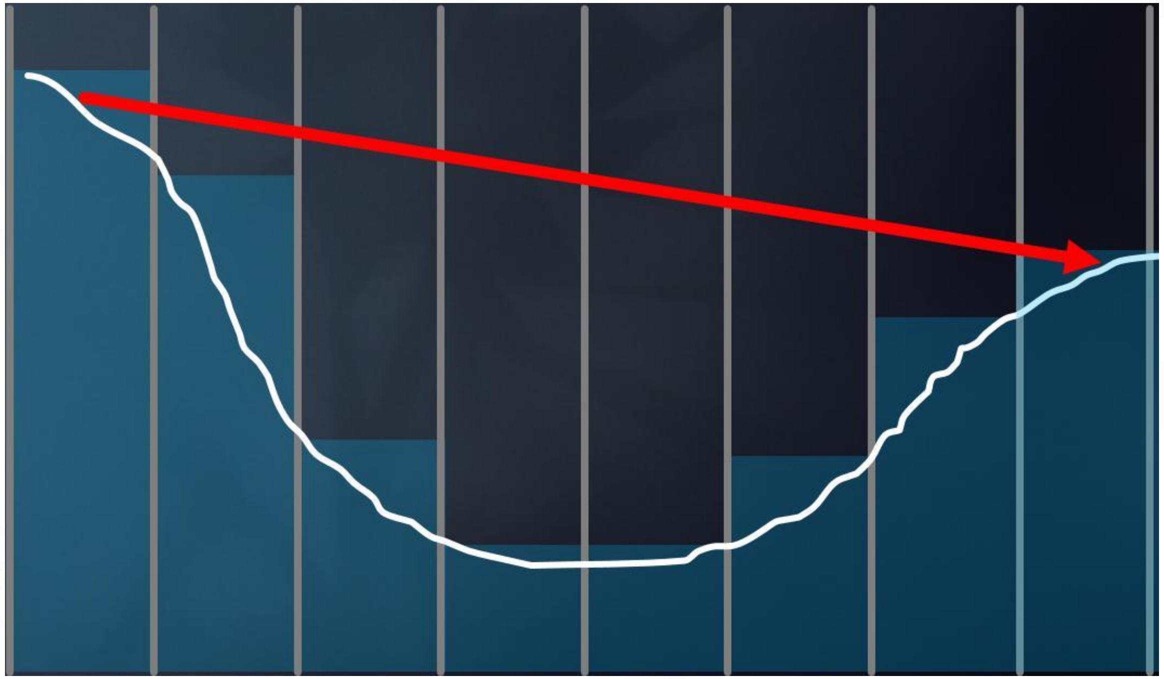
Hierachical Tracing

- Generate min-depth mipmap (pyramid)

- Stackless ray walk of min-depth mipmap

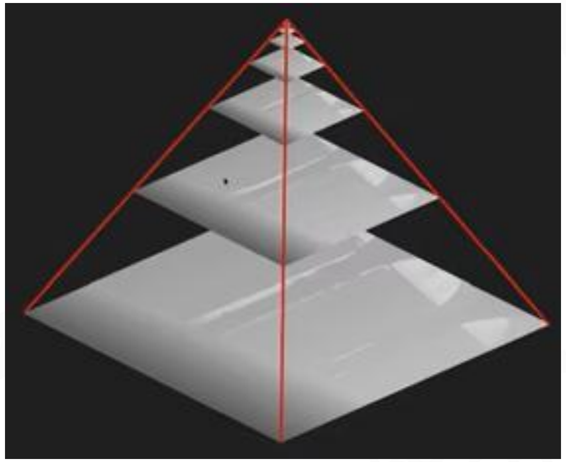


```
level = 0;
while (level > -1)
{
    stepCurrentCell();
    if (above Z plane) level++;
    if (below Z plane) level--;
}
```

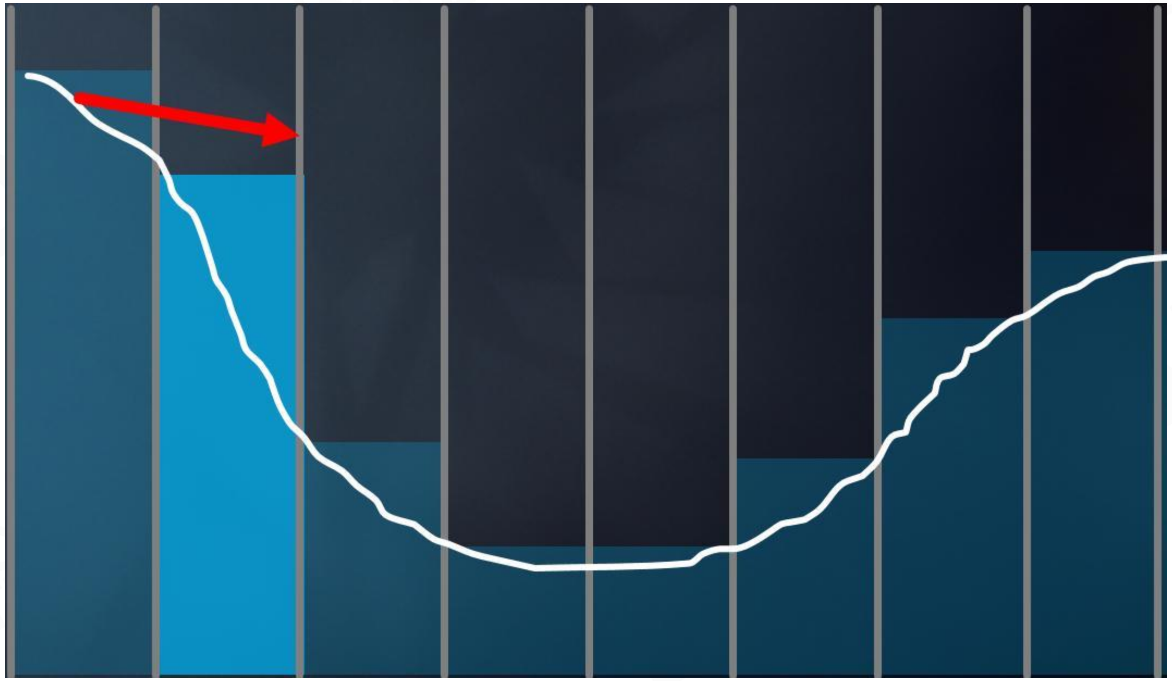


Hierachical Tracing

- Generate min-depth mipmap (pyramid)
- Stackless ray walk of min-depth mipmap



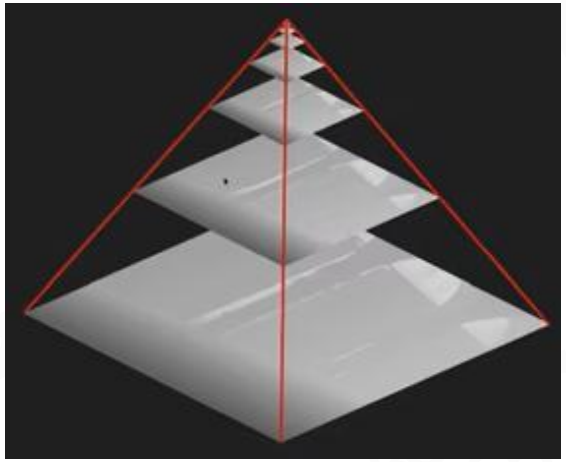
```
level = 0;
while (level > -1)
{
    stepCurrentCell();
    if (above Z plane) level++;
    if (below Z plane) level--;
}
```



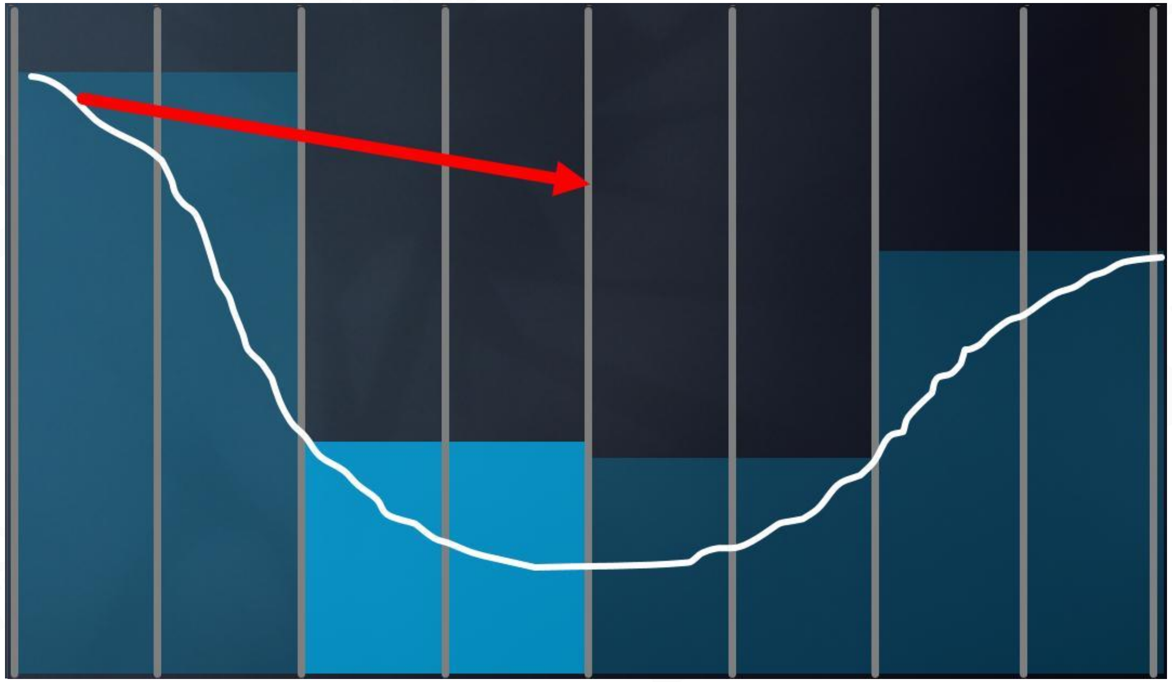
Hierachical Tracing

- Generate min-depth mipmap (pyramid)

- Stackless ray walk of min-depth mipmap



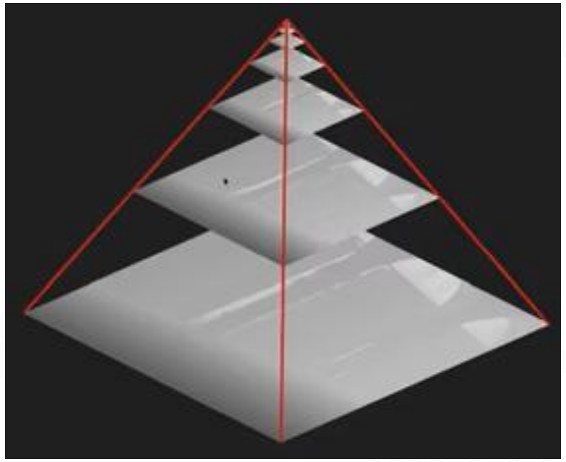
```
level = 0;
while (level > -1)
{
    stepCurrentCell();
    if (above Z plane) level++;
    if (below Z plane) level--;
}
```



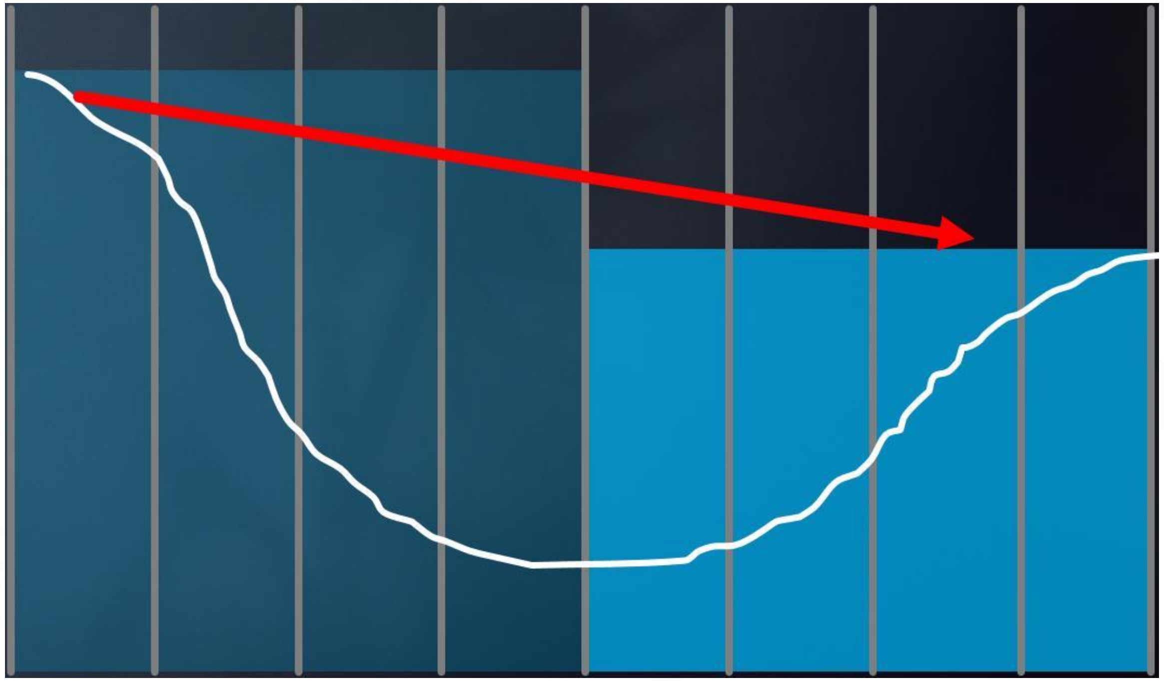
Hierachical Tracing

- Generate min-depth mipmap (pyramid)

- Stackless ray walk of min-depth mipmap

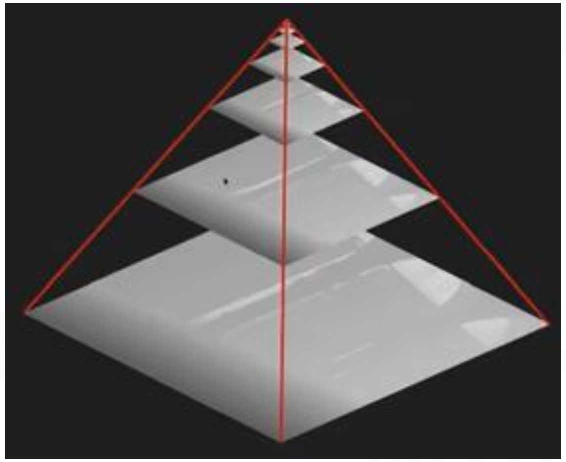


```
level = 0;
while (level > -1)
{
    stepCurrentCell();
    if (above Z plane) level++;
    if (below Z plane) level--;
}
```

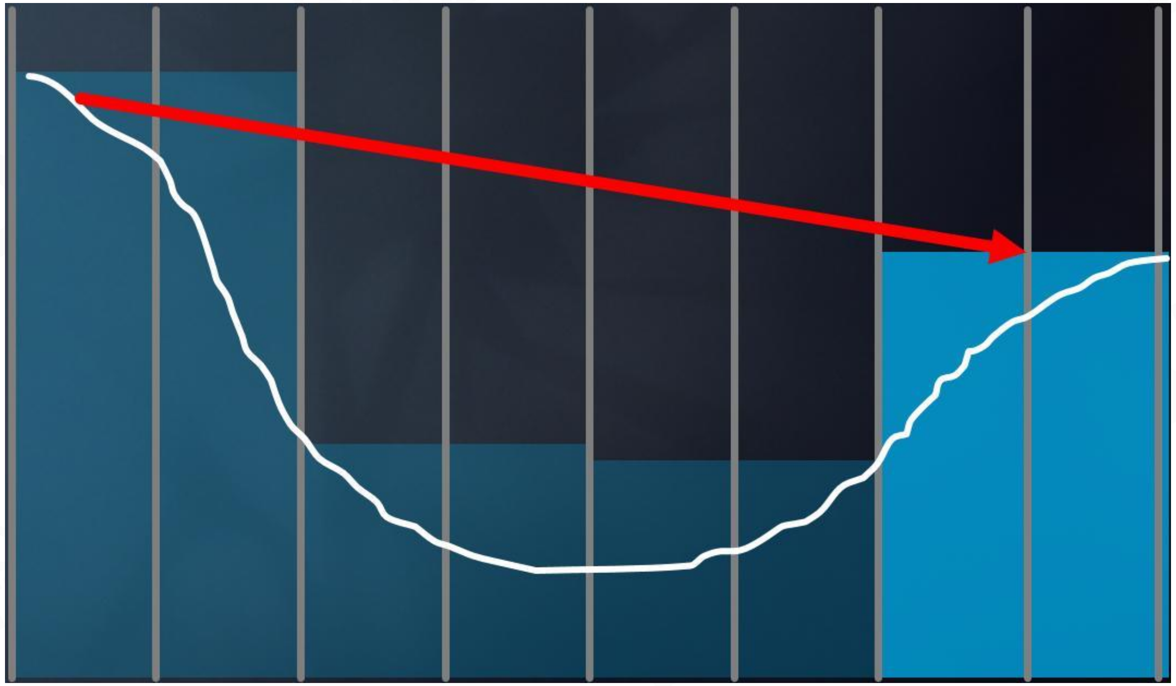


Hierachical Tracing

- Generate min-depth mipmap (pyramid)
- Stackless ray walk of min-depth mipmap



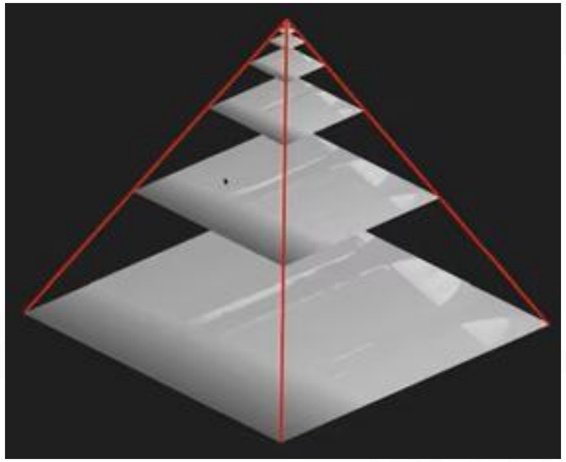
```
level = 0;
while (level > -1)
{
    stepCurrentCell();
    if (above Z plane) level++;
    if (below Z plane) level--;
}
```



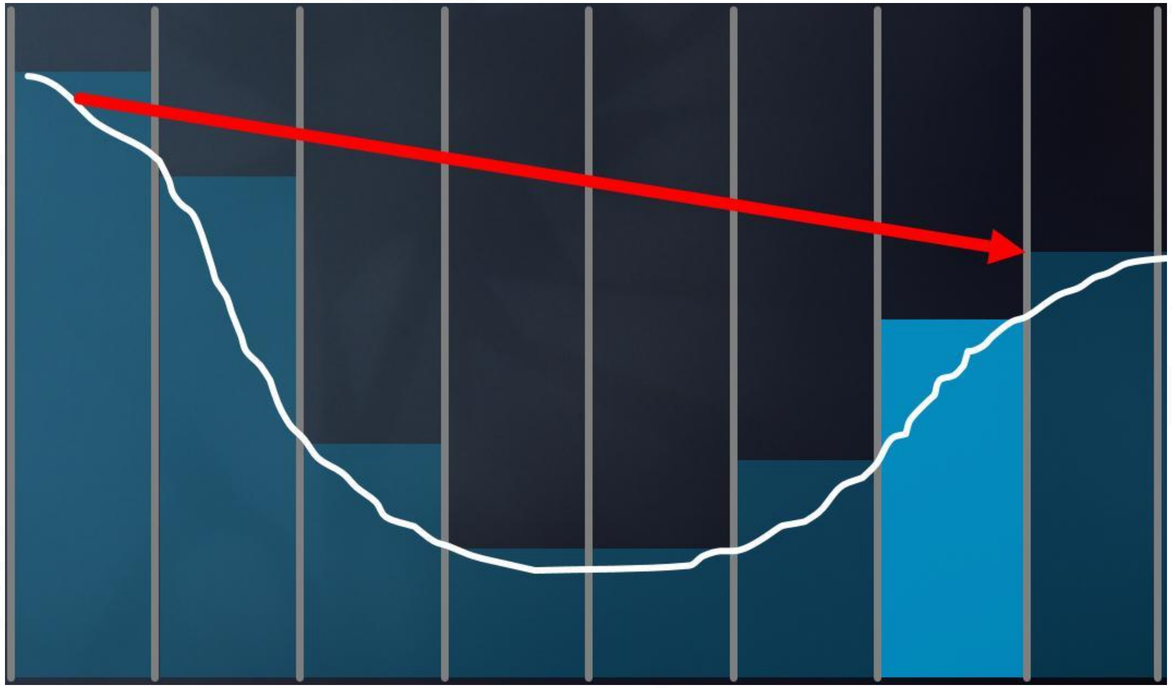
Hierachical Tracing

- Generate min-depth mipmap (pyramid)

- Stackless ray walk of min-depth mipmap



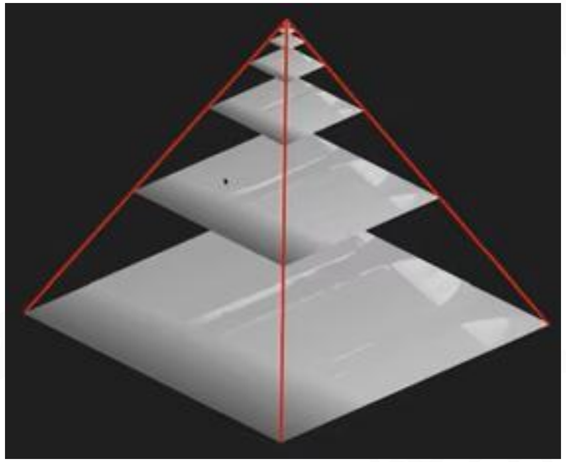
```
level = 0;
while (level > -1)
{
    stepCurrentCell();
    if (above Z plane) level++;
    if (below Z plane) level--;
}
```



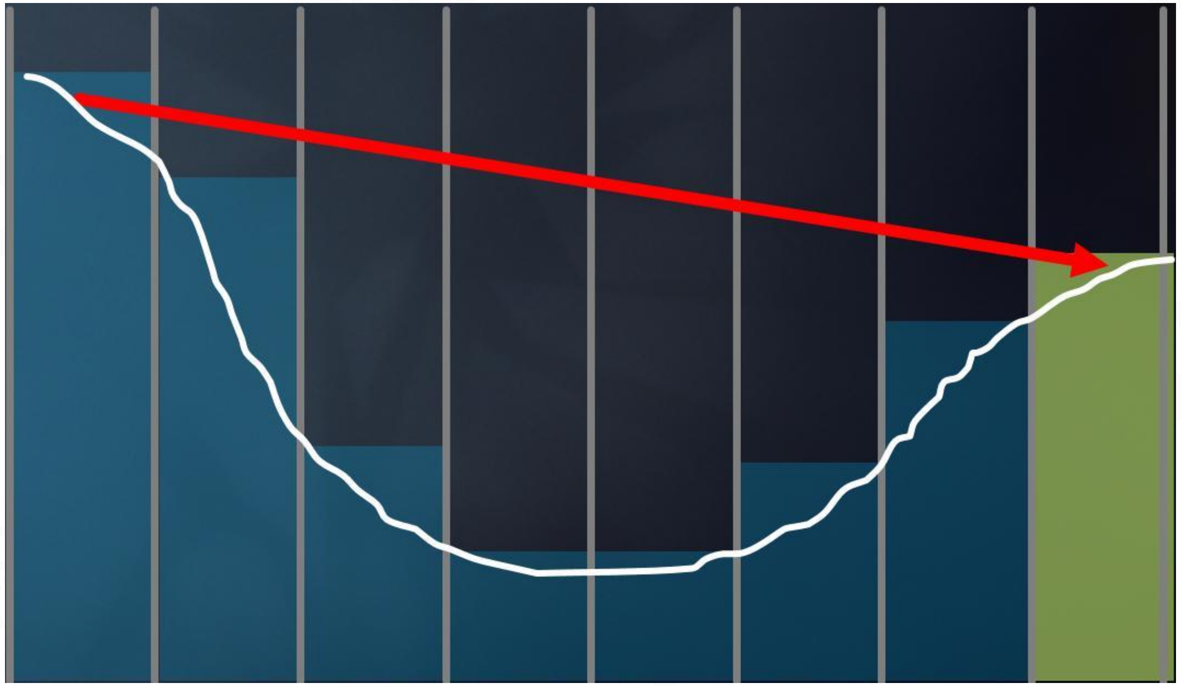
Hierachical Tracing

- Generate min-depth mipmap (pyramid)

- Stackless ray walk of min-depth mipmap

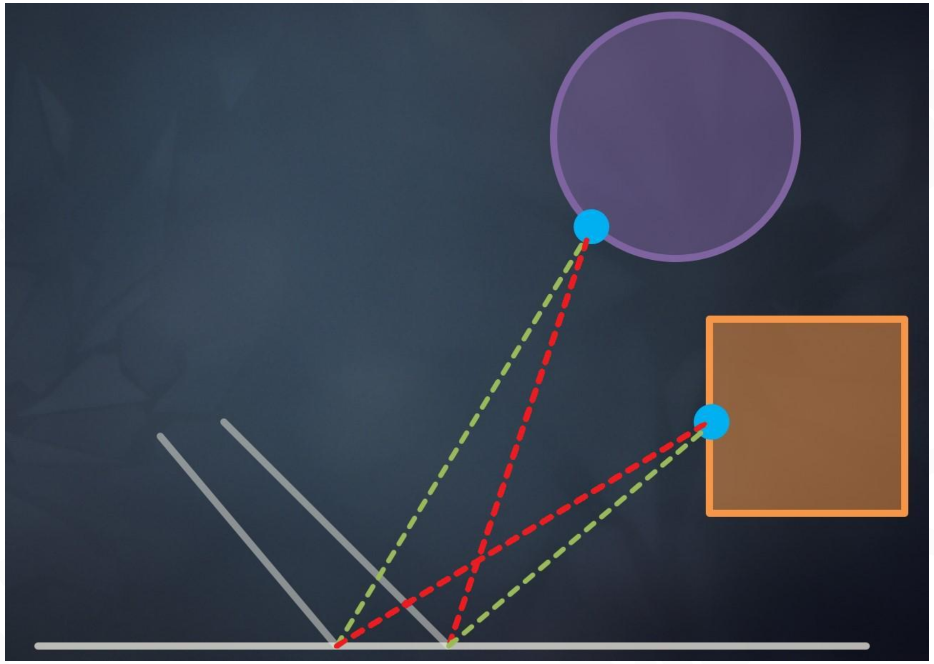


```
level = 0;
while (level > -1)
{
    stepCurrentCell();
    if (above Z plane) level++;
    if (below Z plane) level--;
}
```



Ray Reuse among Neighbor Pixels

- Store hitpoint data
- Assume visibility is the same between neighbors
- Regard ray to neighbor's hitpoint as valid



Cone Tracing with Mipmap Filtering

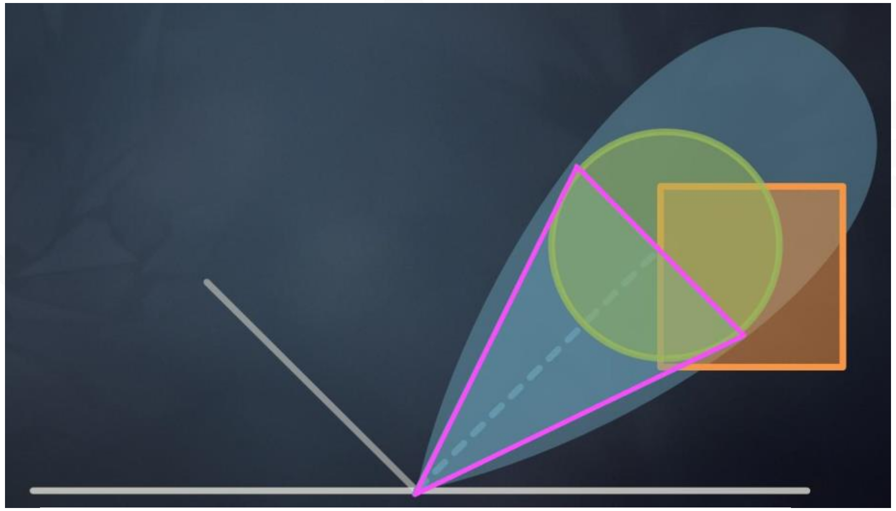
Estimate footprint of a cone at hit point

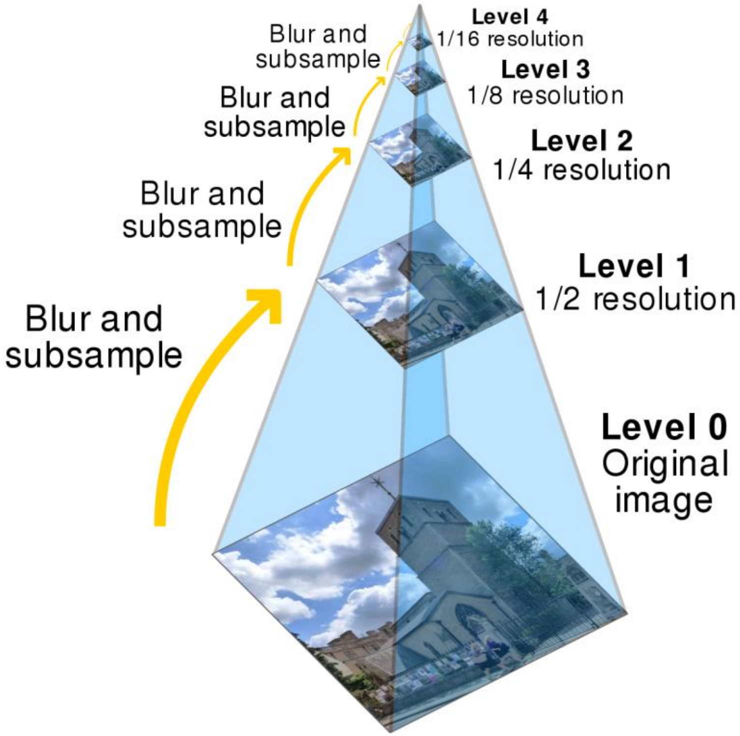
- roughness
- distance to hit

Sample the color mipmap

- mip level is determined by footprint

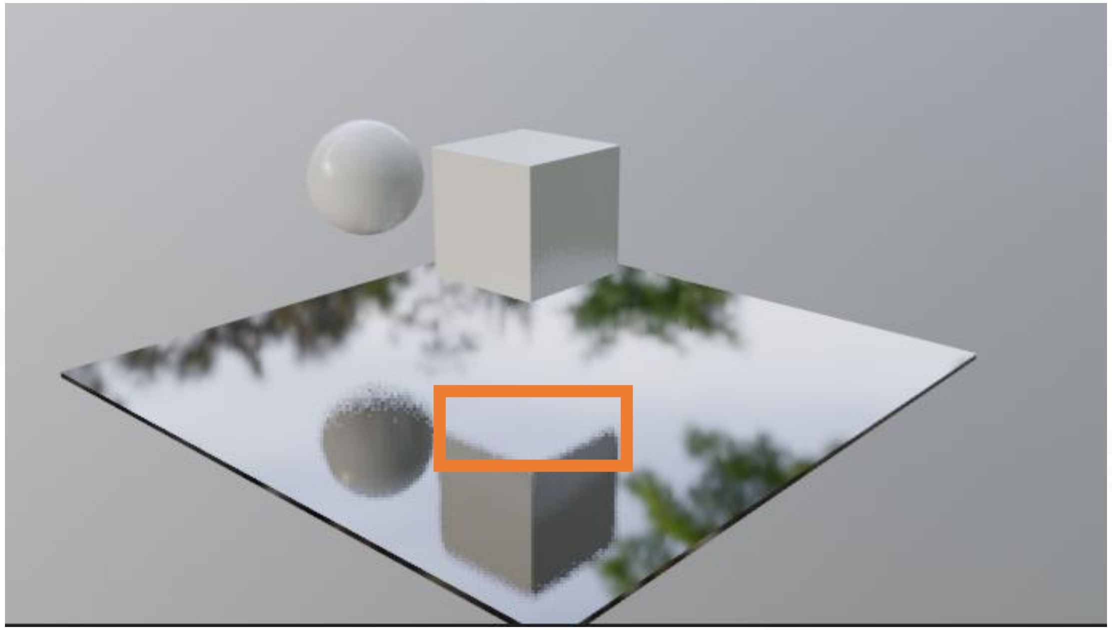
Pre-filter color mipmap (pyramid)

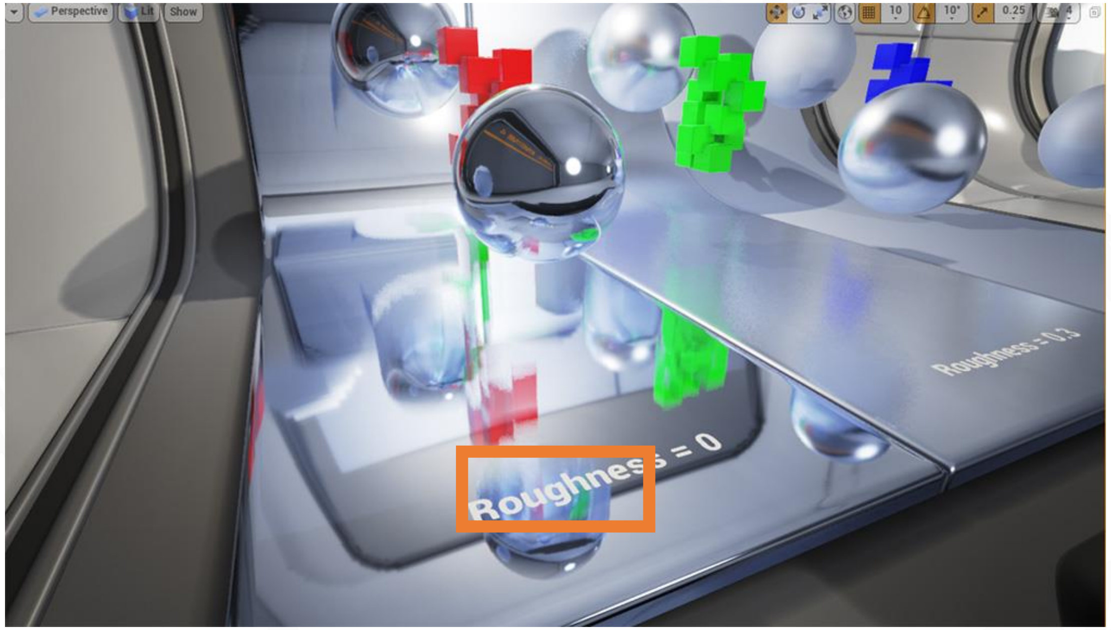




SSGI Summary

- Pros:
  - Fast for glossy and specular reflections
  - Good quality
  - No occlusion issues
- Cons:
  - Missing information outside screen
  - Affects of incorrect visibility of neighbor ray reuse





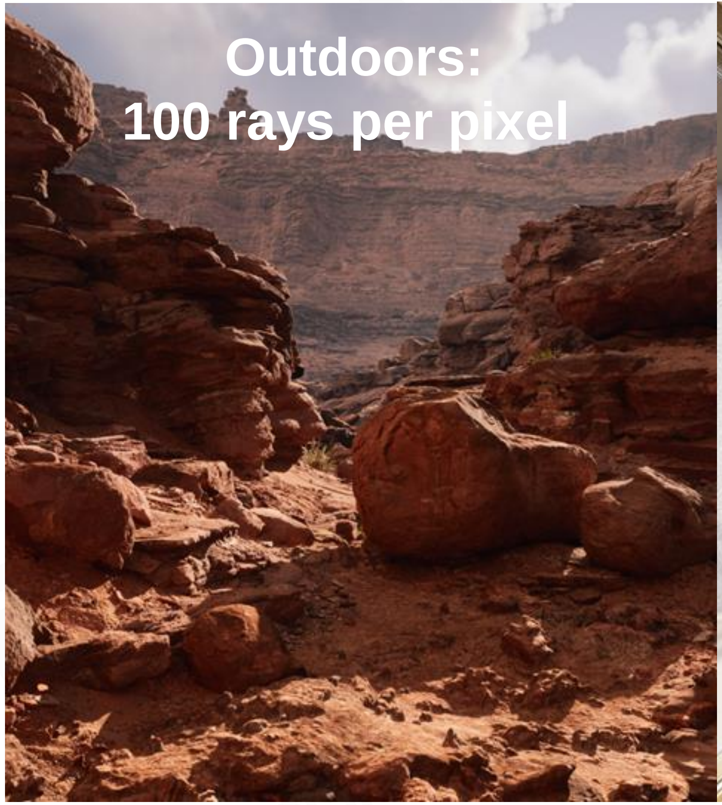
Unique Advantages of SSGI

- Easy to handle close contact shadow
- Precise hit point calculation
- Decouple from scene complexity
- Handle dynamic objects

Lumen

Ray Traces are slow

- Can only afford 1/2 ray per pixel
- But quality GI needs hundreds



Sampling is hard

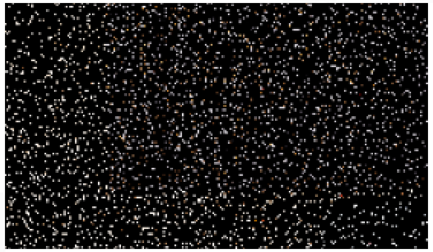
Previous real-time work: Irradiance Fields

- Problems:
  - Leaking and over-occlusion
  - Probe placement
  - Slow lighting update
  - Distinctive flat look

Previous real-time work: Screen Space Denoiser

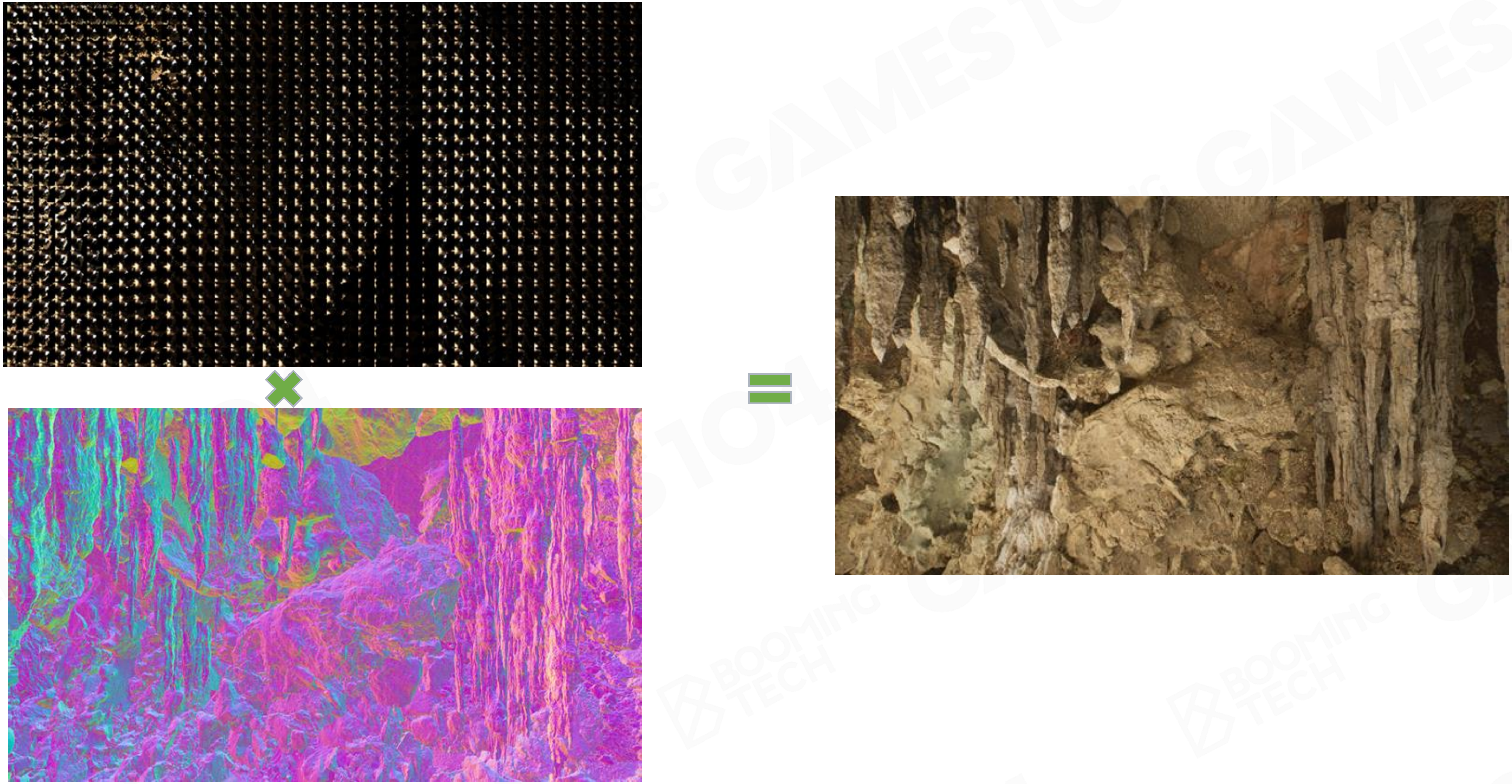
- Problems:
  - Too noisy in many difficult indoor cases
  - Noise is not constant.

Near bright window



Far bright window

Low-res filtered scene space probes lit full pixels

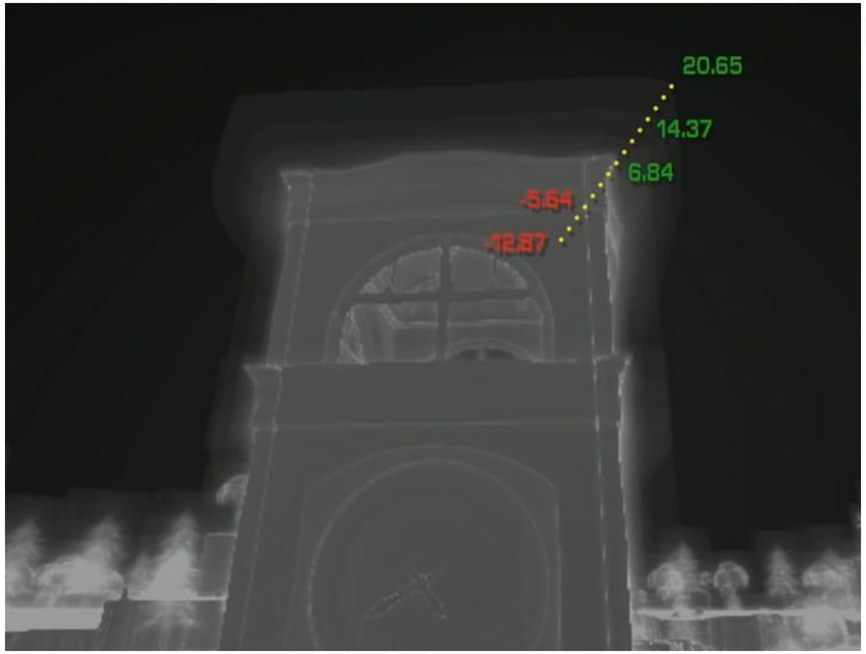


Phase 1 : Fast Ray Trace in Any Hardware

Signed Distance Field (SDF)

What is SDF

- The distance to the nearest surface at every point
- Inside regions store negative distance (signed)
- Distance = 0 is the surface



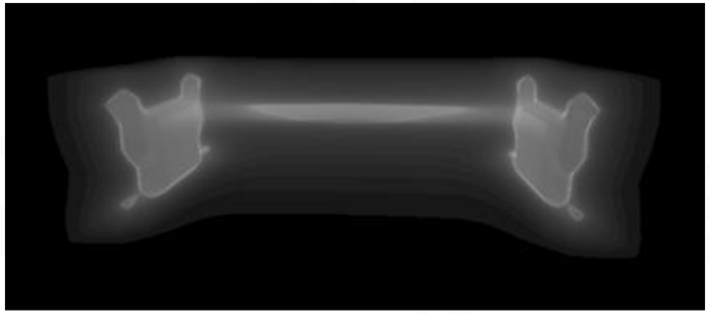
Per-Mesh SDF

Store SDF of the whole scene is expensive

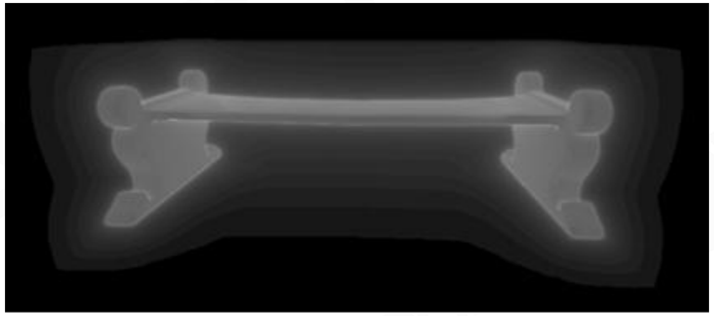
Generated for each mesh

- Resolution based on mesh size
- Embree point query
- Trace rays and count triangle back faces for sign (more than 25% hit back is negative)

Original Mesh



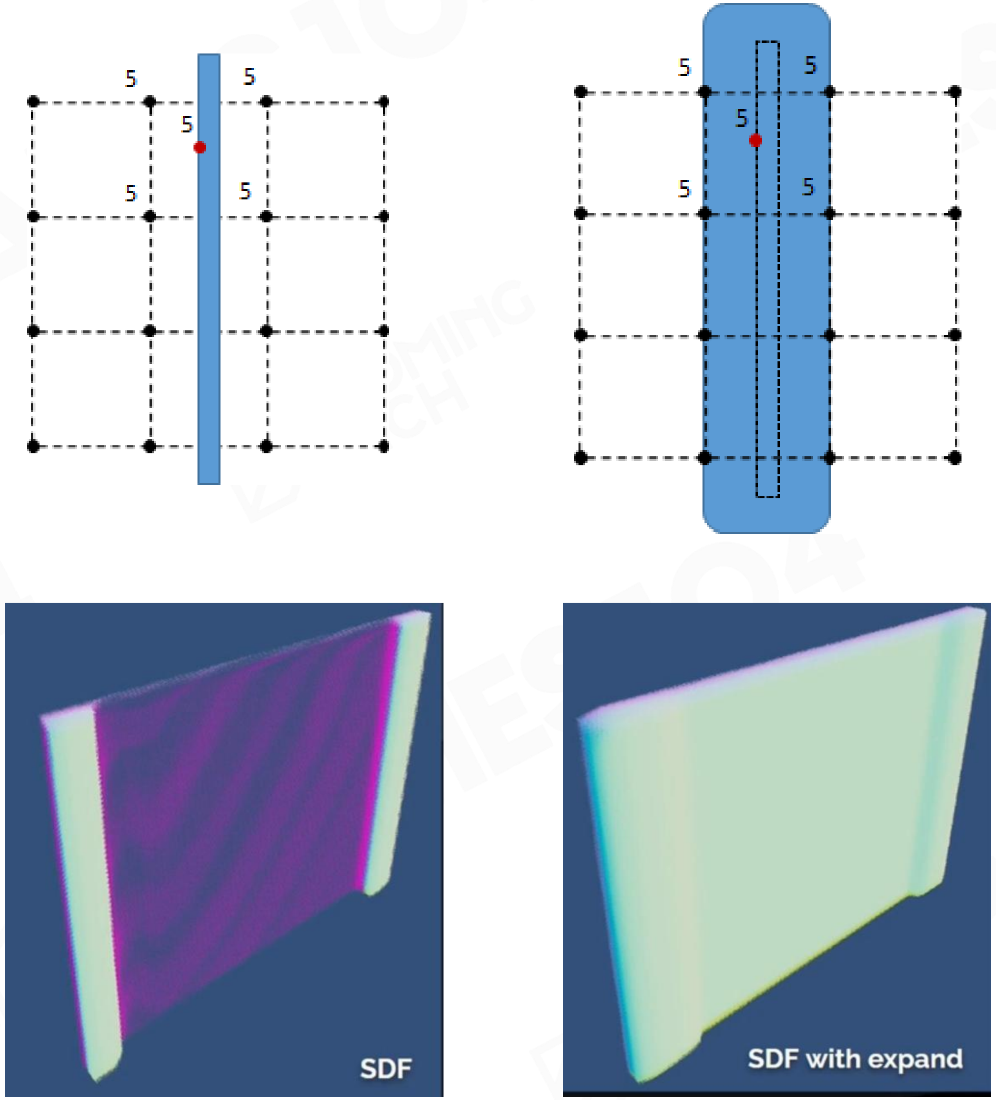
Resolution is too low, important features are lost



Resolution has been increased, important features represented

SDF for Thin meshes

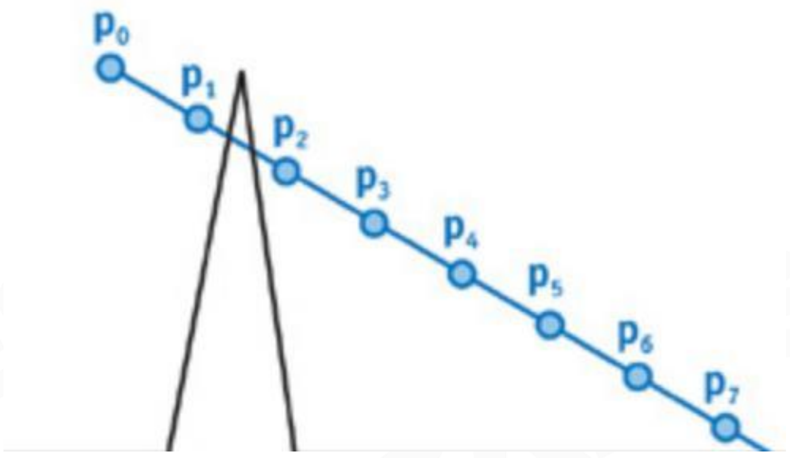
- Half voxel expand to fix leaking
- Lost contact shadows due to surface bias
  - Over occlusion better than leaking



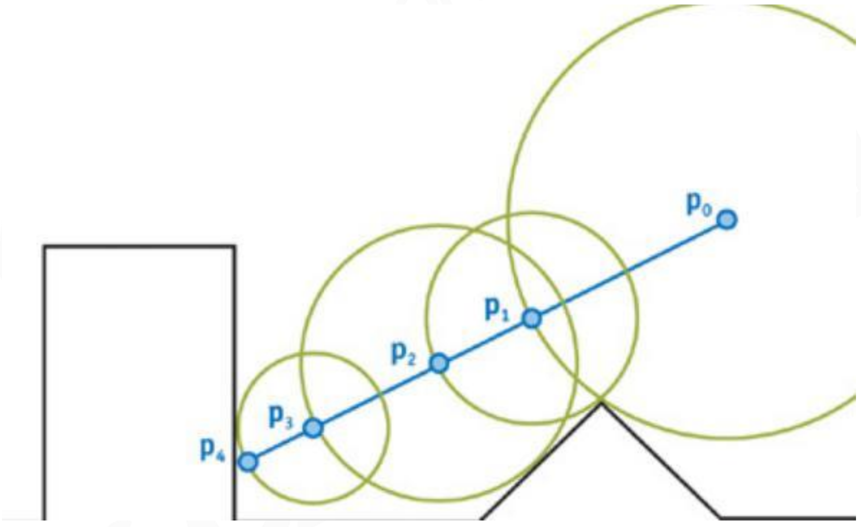
Ray Tracing with SDF

Ray intersection skips through empty space based on distance to surface

- Safe and fast
- Each time at p, just travel SDF(p) distance

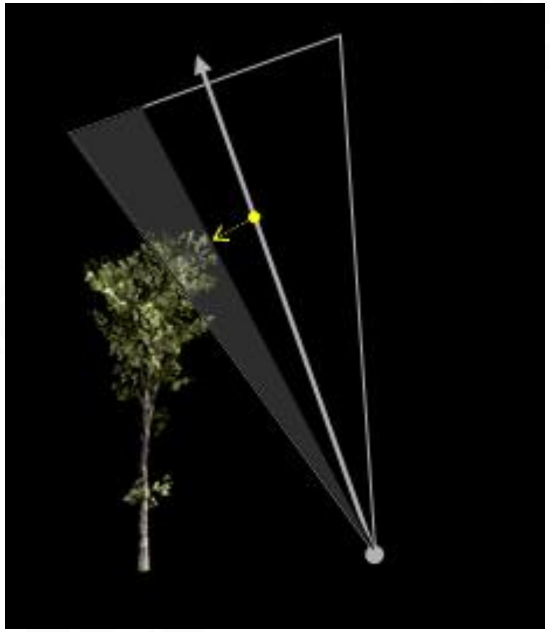


Fixed steps tracing

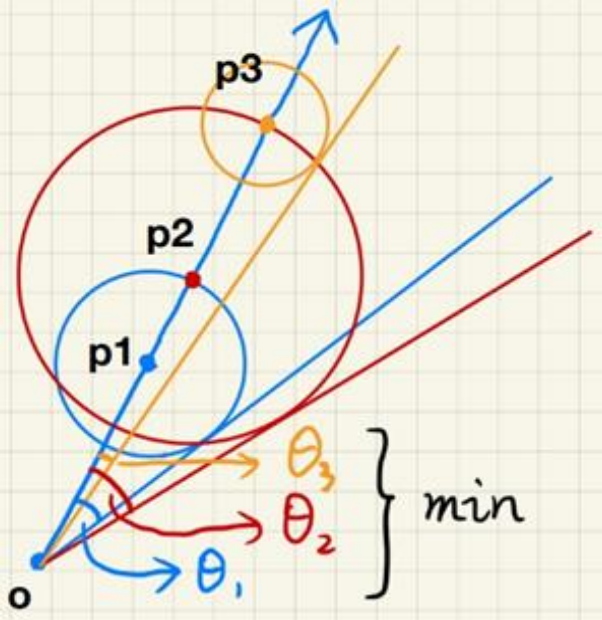


Sphere tracing

Cone Tracing with SDF(ie. Soft Shadow)

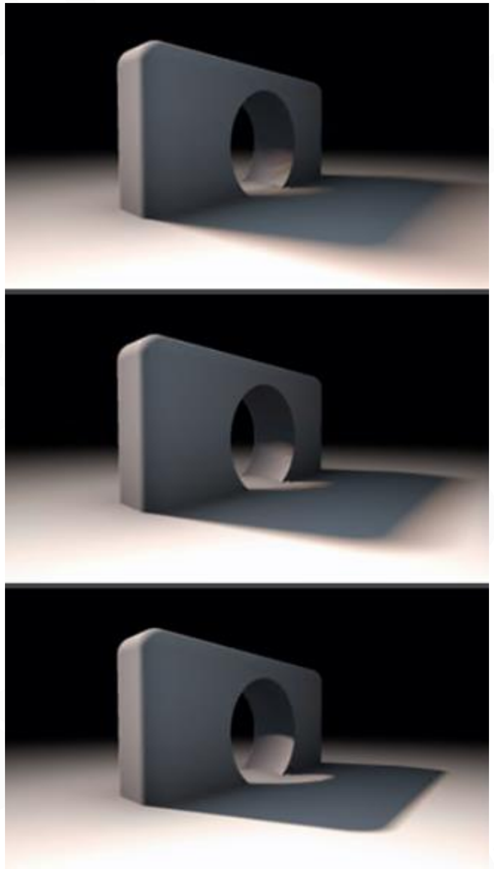


Cone intersection



$$\theta = arcsin\frac{\mathrm{SDF}(p)}{\|p-o\|}$$

$$\min\theta \approx \min\left\{\frac{k\cdot\mathrm{SDF}(p)}{\|p-o\|}, 1.0\right\}$$



$k = 2$

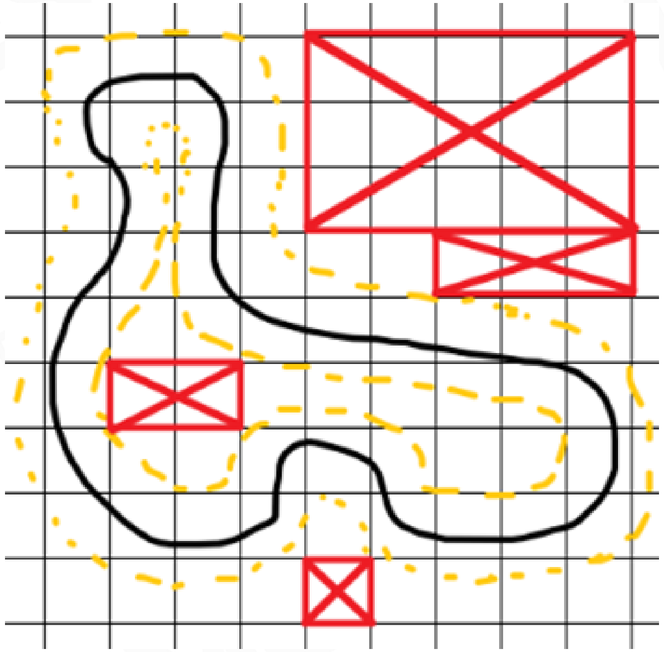
$k = 8$

$k = 32$

Sparse Mesh SDF

Divides the Mesh SDF into bricks

- Define a max_encode_distance
  - Invalid if $\forall$ sdf(brick) > max_encode_distance
- IndirectionTable store the index of each brick



Sparse Mesh SDF

Divides the Mesh SDF into bricks

- Define a max_encode_distance
  - Invalid if ∀ sdf(brick) > max_encode_distance
- IndirectionTable store the index of each brick

Example:

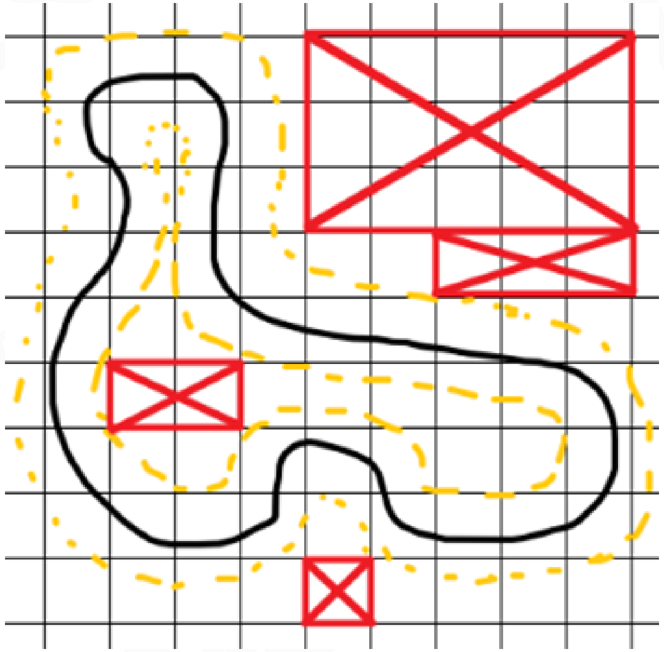
0	1	2
3	4	5
6	7	8

IndirectionTable:

0	1	X	2	X	3	X	4	5

BrickData:

Brick 0	Brick 1	Brick 3	Brick 5	Brick 7	Brick 8

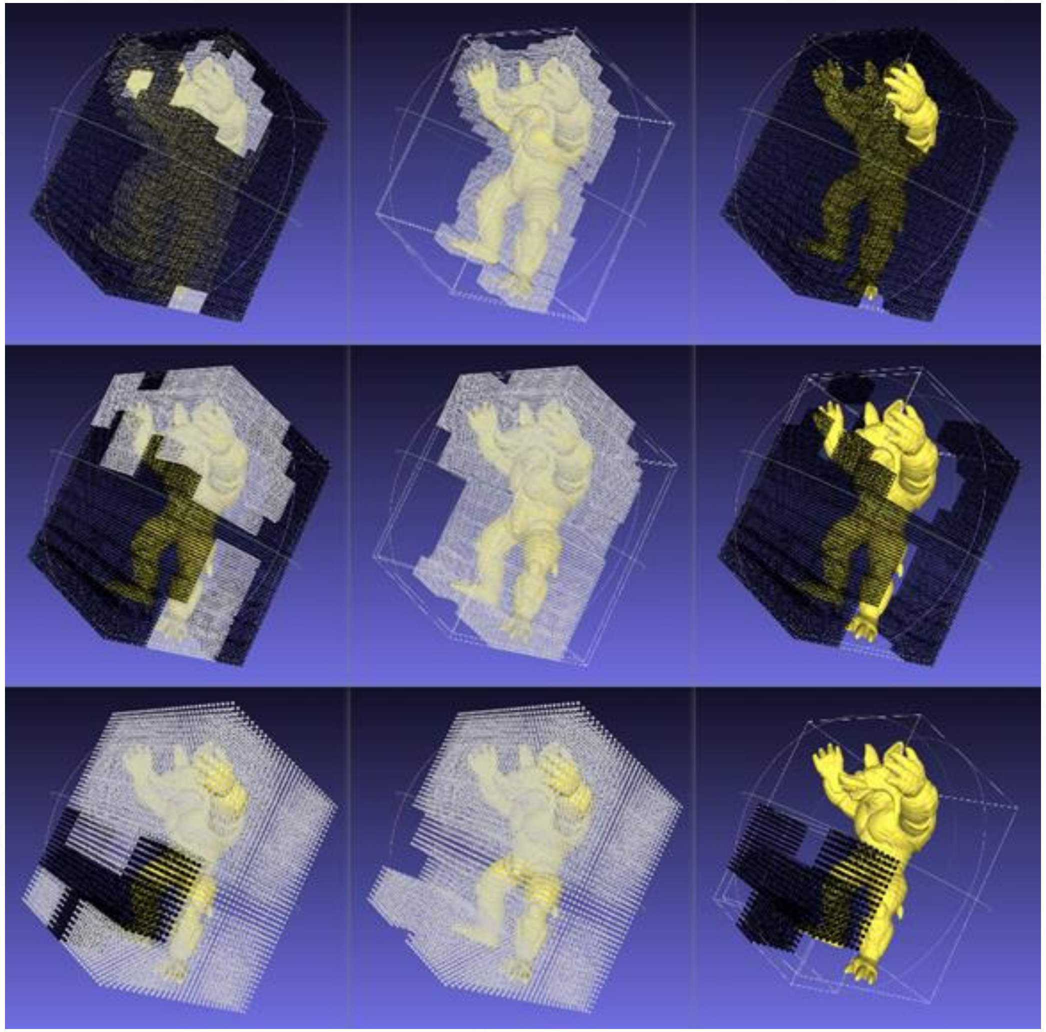


Mesh SDF LoD

- Every frame GPU gathers requests
- CPU download requests and streams pages in/out
- 3 mips are generated
  - Lowest resolution always loaded and the other 2 streamed

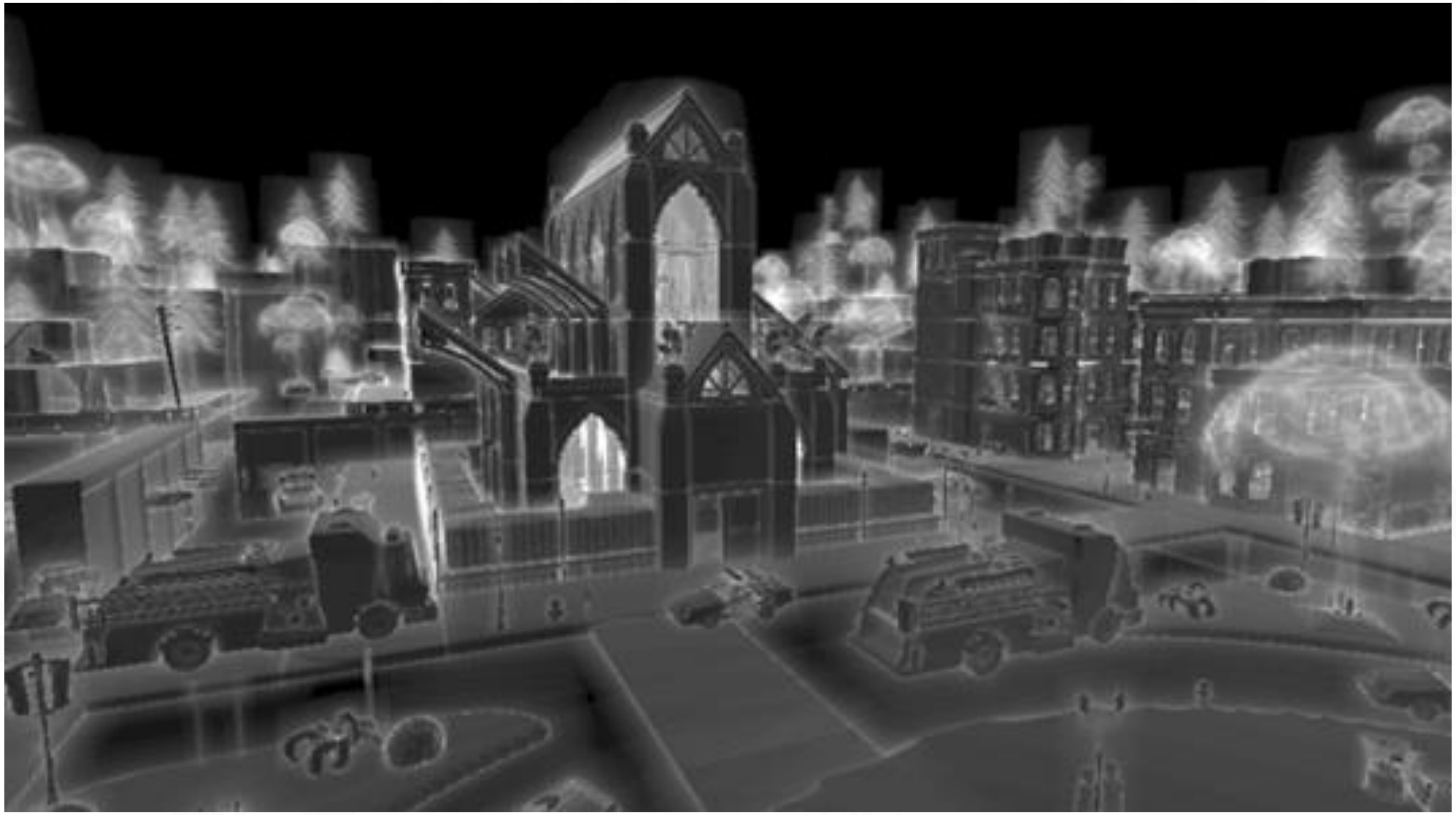
Sparse Mesh SDF

model size = 6.35m x 7.57m x 5.77m



	brick
mip0 126x154x112 5cm	**brick:2310/6336** 1.15MB/3.09MB
mip1 63x77x56 10cm	**brick:532/792** 0.26MB/0.39MB
mip2 35x42x28 20cm	**brick:112/120** 56.5KB/60.0KB

Ray Tracing Cost in Real Scene



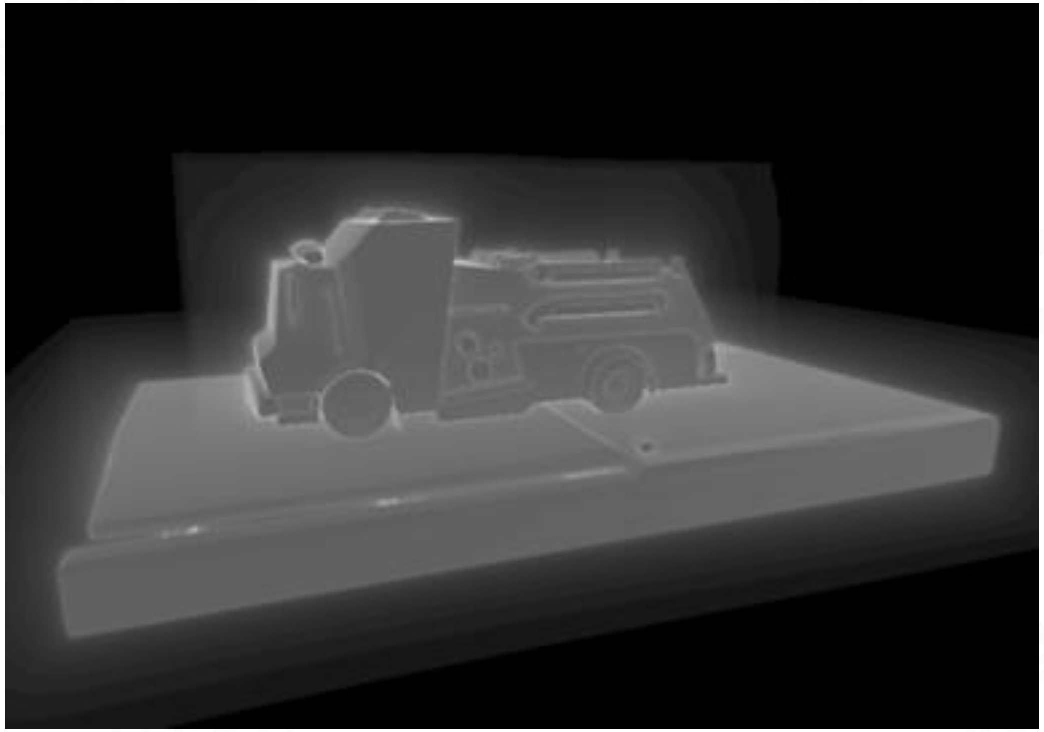
Trace camera rays and visualize the number of steps

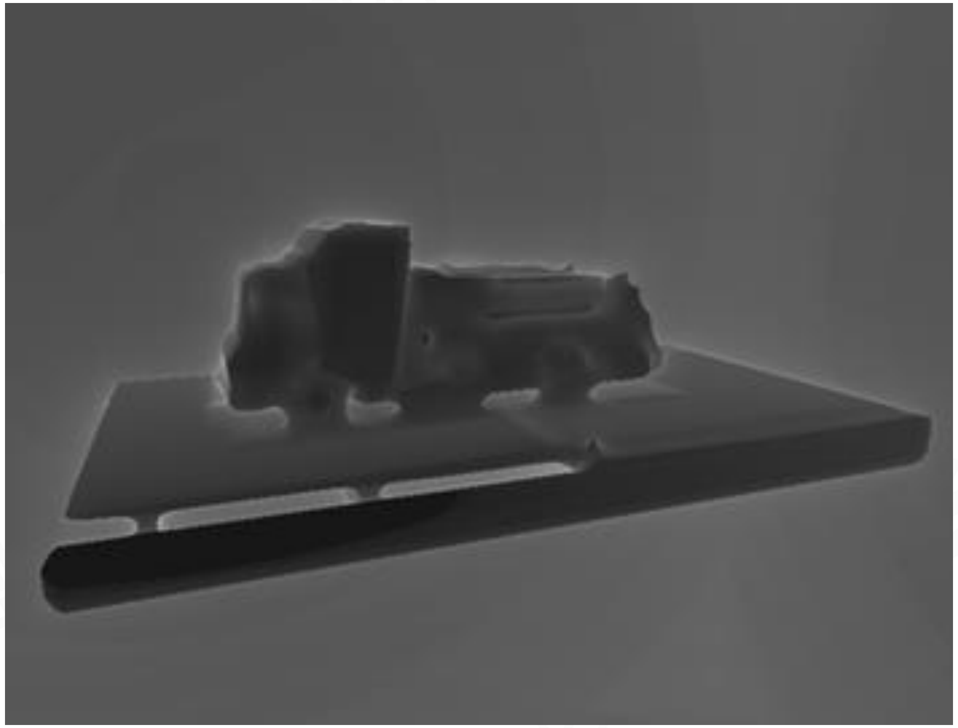
Many Objects along Each Ray

Number of hit objects along each ray

Global SDF

- Global SDF is inaccurate near surface
- Sample object SDFs near start of cone, global SDF for the rest



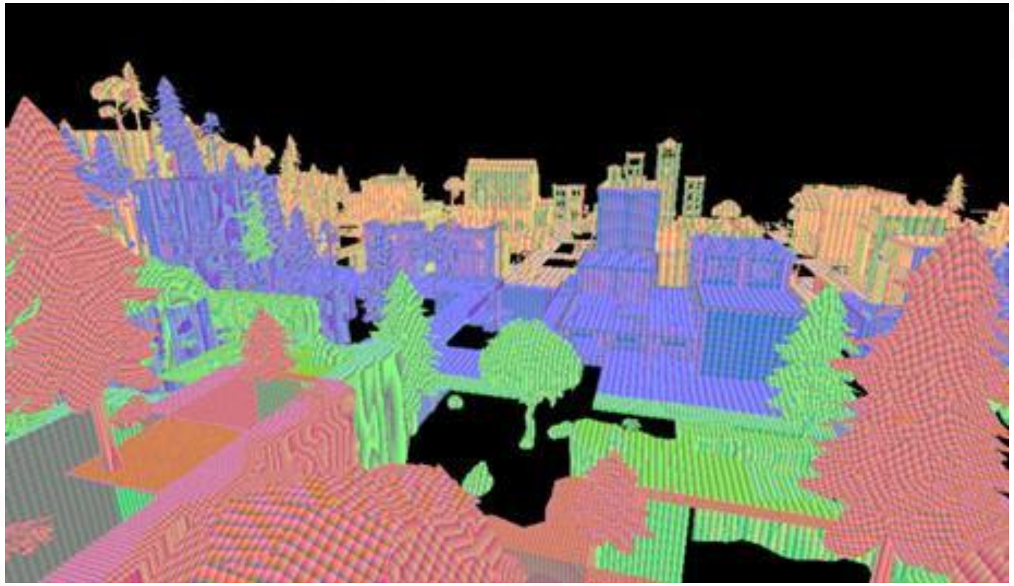


Ray Tracing with Global SDF

Massively reduces tracing cost on overlapping objects

Cache Global SDF around Camera

- 4 clipmaps centered around camera
- Clipmaps are scrolled with movement
- Distant clipmaps updated less frequently
- Also sparsely stored (~16x memory saving)

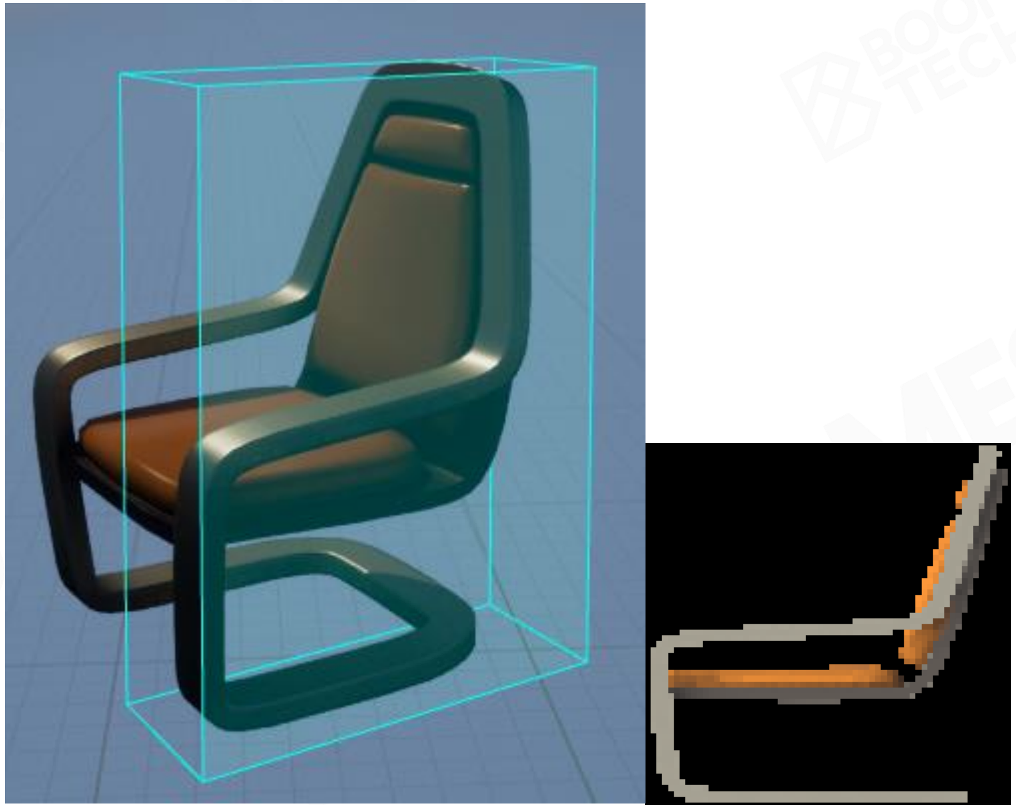


Phase 2 : Radiance Injection and Caching

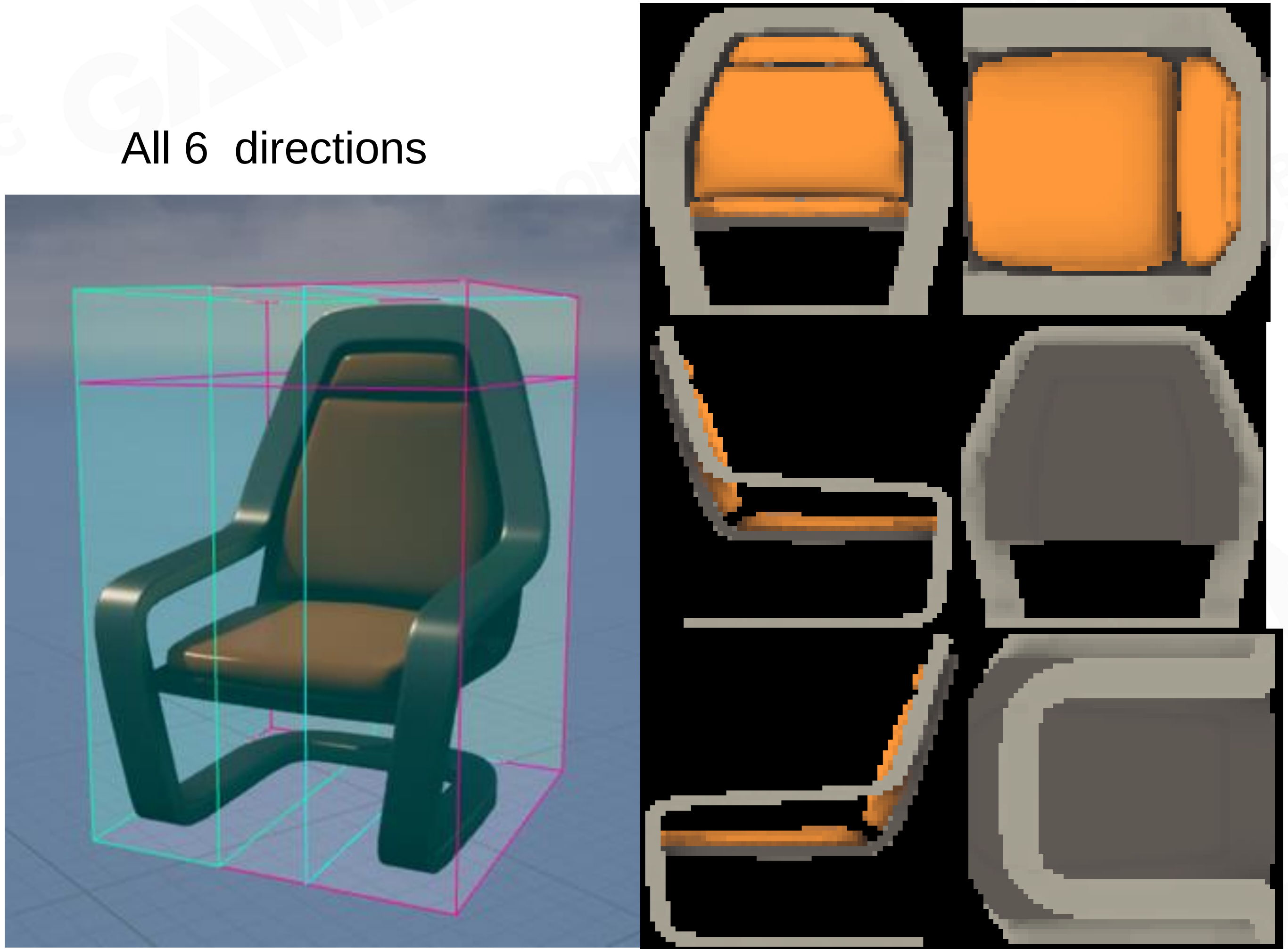
Mesh card – orthogonal camera on 6-Axis Aligned directions

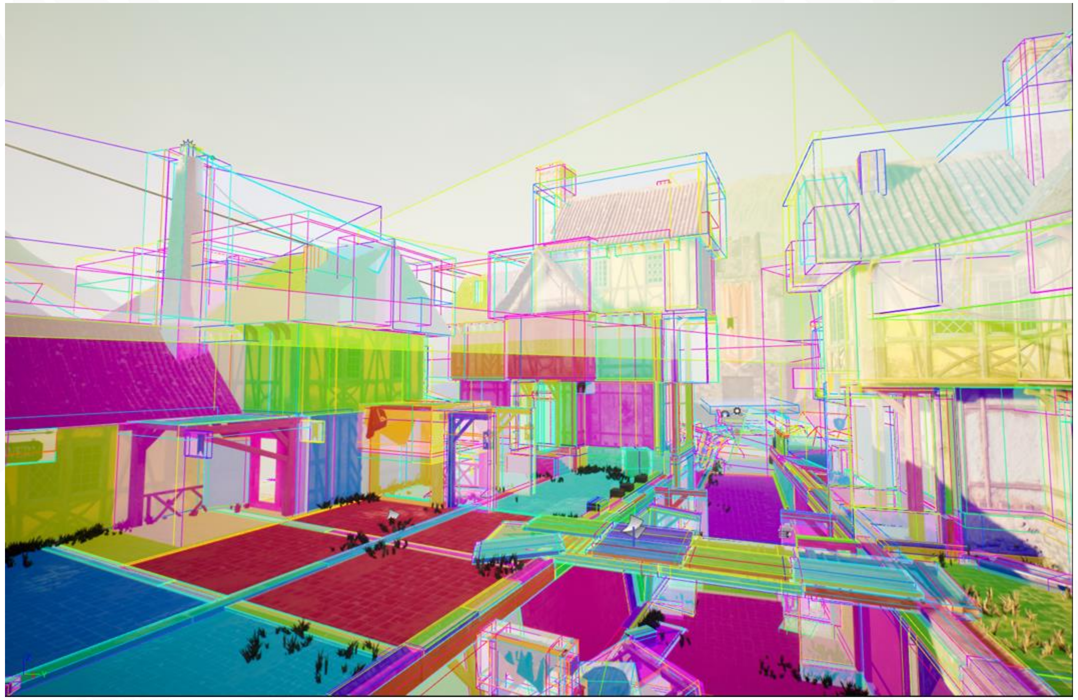
```
class FLumenCard
{
    FLumenCardOBB LocalOBB;
    FLumenCardOBB WorldOBB;
    uint8 AxisAlignedDirectionIndex;
};
```

-Y direction



All 6 directions





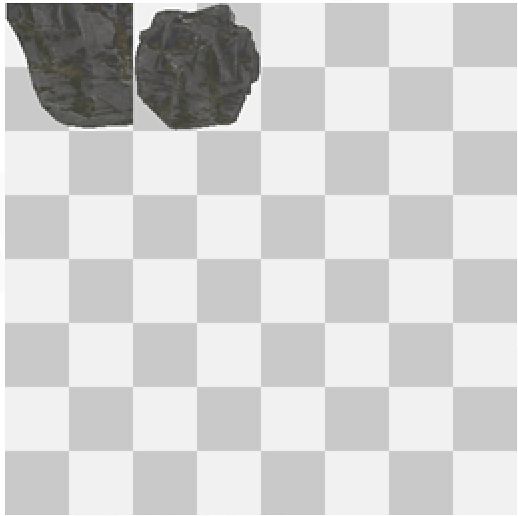
Generate Surface Cache

Two Passes

Pass 1： Card capture

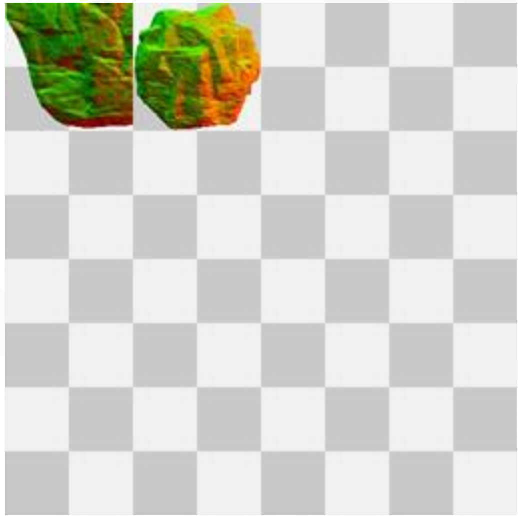
- Fix texel budget per frame (512x512)
- Sort by distance to camera and GPU feedback
- Capture resultion depends on card projection on screen

Albedo



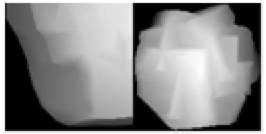
512x512 RGBA8

Normal



512x512 R8G8

Depth



512x512 D32S8

Emissive

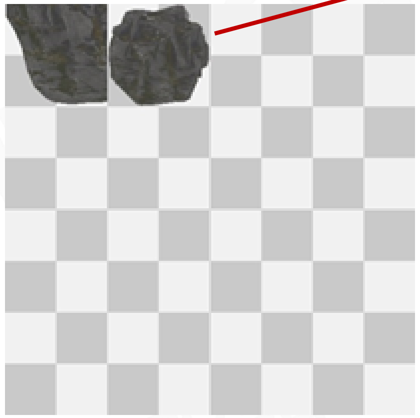
512x512 R11G11B10

Generate Surface Cache

Two Passes

Pass 1： Card capture
Pass 2： Copy cards to surface cache and compress

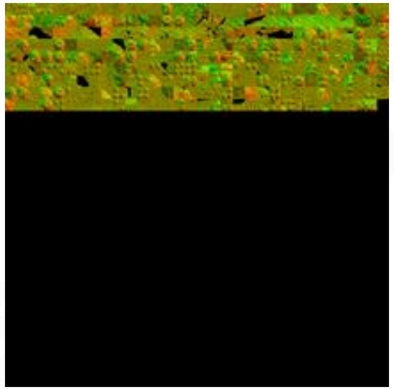
512x512

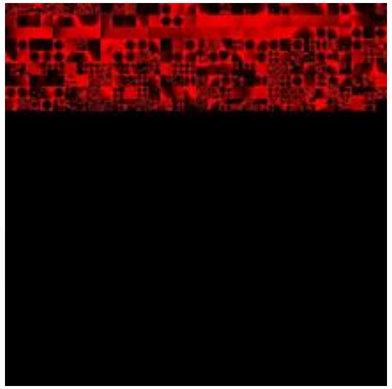


Lumen.CardCaptureAlbedoAltas

4096x4096

Lumens.SceneAlbedo





Generate Surface Cache

Two Passes

Pass 1: Card capture
Pass 2: Copy cards to surface cache and compress

4096x4096 Surface Cache Atlas			
Albedo	RGB8	BC7	16mb
Opacity	R8	BC4	8mb
Depth	R16	-	32mb
Normal	Hemisphere RG8	BC4	16mb
Emissive	RGB Float16	BC6H	16mb

compress from 320mb to 88mb

```
static FLumenSurfaceLayerConfig Configs[(uint32)ELumenSurfaceCacheLayer::MAX] =
{
    { TEXT("Depth"),      PF_G16,            PF_Unknown, PF_Unknown,            FVector(1.0f, 0.0f, 0.0f) },
    { TEXT("Albedo"),     PF_R8G8B8A8,       PF_BC7,     PF_R32G32B32A32_UINT,  FVector(0.0f, 0.0f, 0.0f) },
    { TEXT("Opacity"),    PF_G8,             PF_BC4,     PF_R32G32_UINT,        FVector(1.0f, 0.0f, 0.0f) },
    { TEXT("Normal"),     PF_R8G8,           PF_BC5,     PF_R32G32B32A32_UINT,  FVector(0.0f, 0.0f, 0.0f) },
    { TEXT("Emissive"),   PF_FloatR11G11B10, PF_BC6H,    PF_R32G32B32A32_UINT,  FVector(0.0f, 0.0f, 0.0f) }
};
```

View Dependent Per-Object Card Resolution

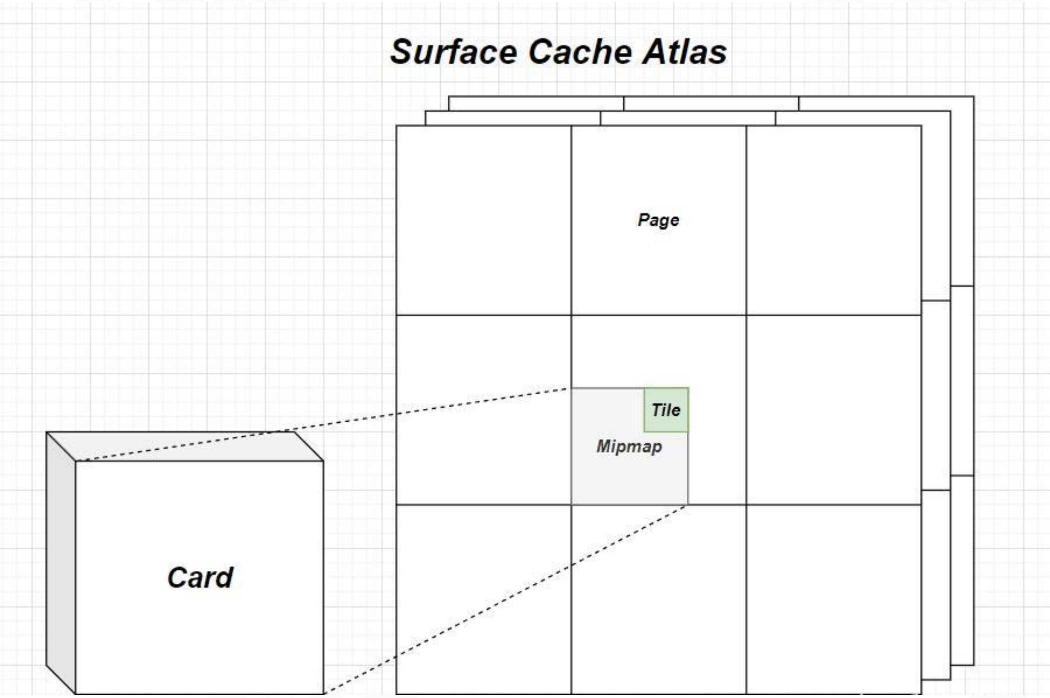
128x128 physical pages in a 4096x4096 atlas

Card capture res >= 128x128

- Split into multiple 128x128 physical pages

Card capture res < 128x128

- Sub-allocate from a 128x128 physical page

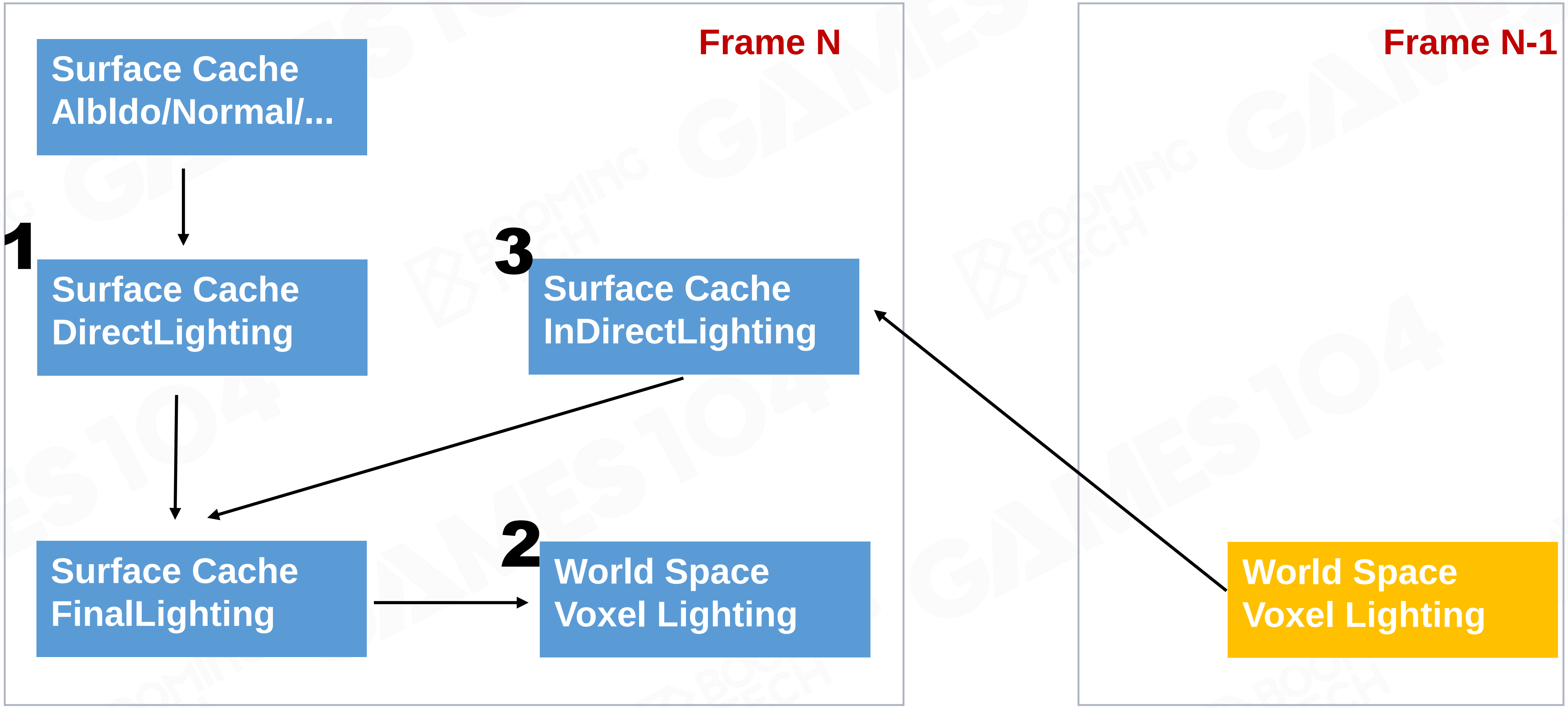


How can we “freeze” lighting on Surface Cache

How to compute lighting on hit?

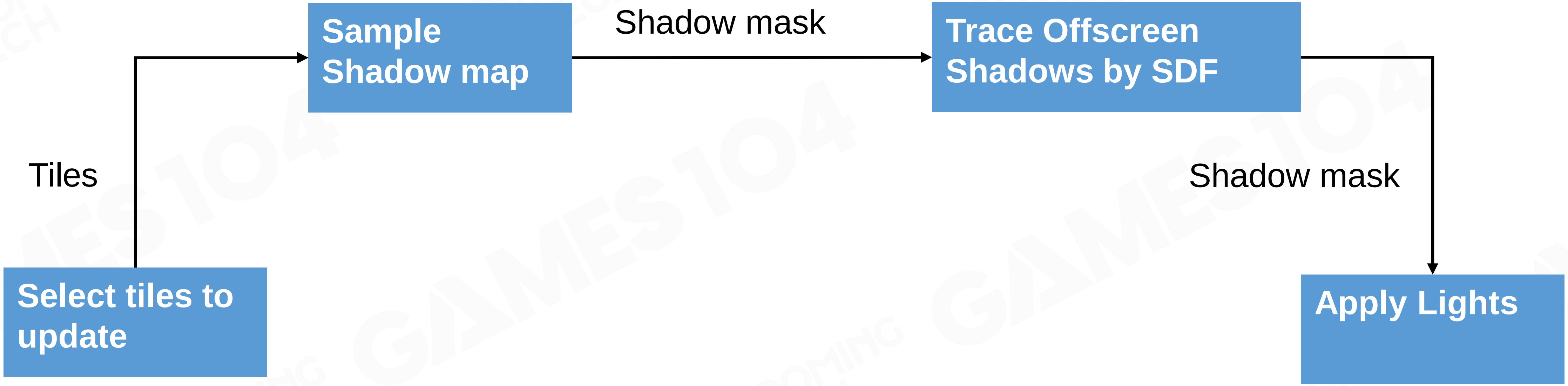
- Is the pixel under the shadow
- How can we handle multi-bounce

Lighting Cache Pipeline

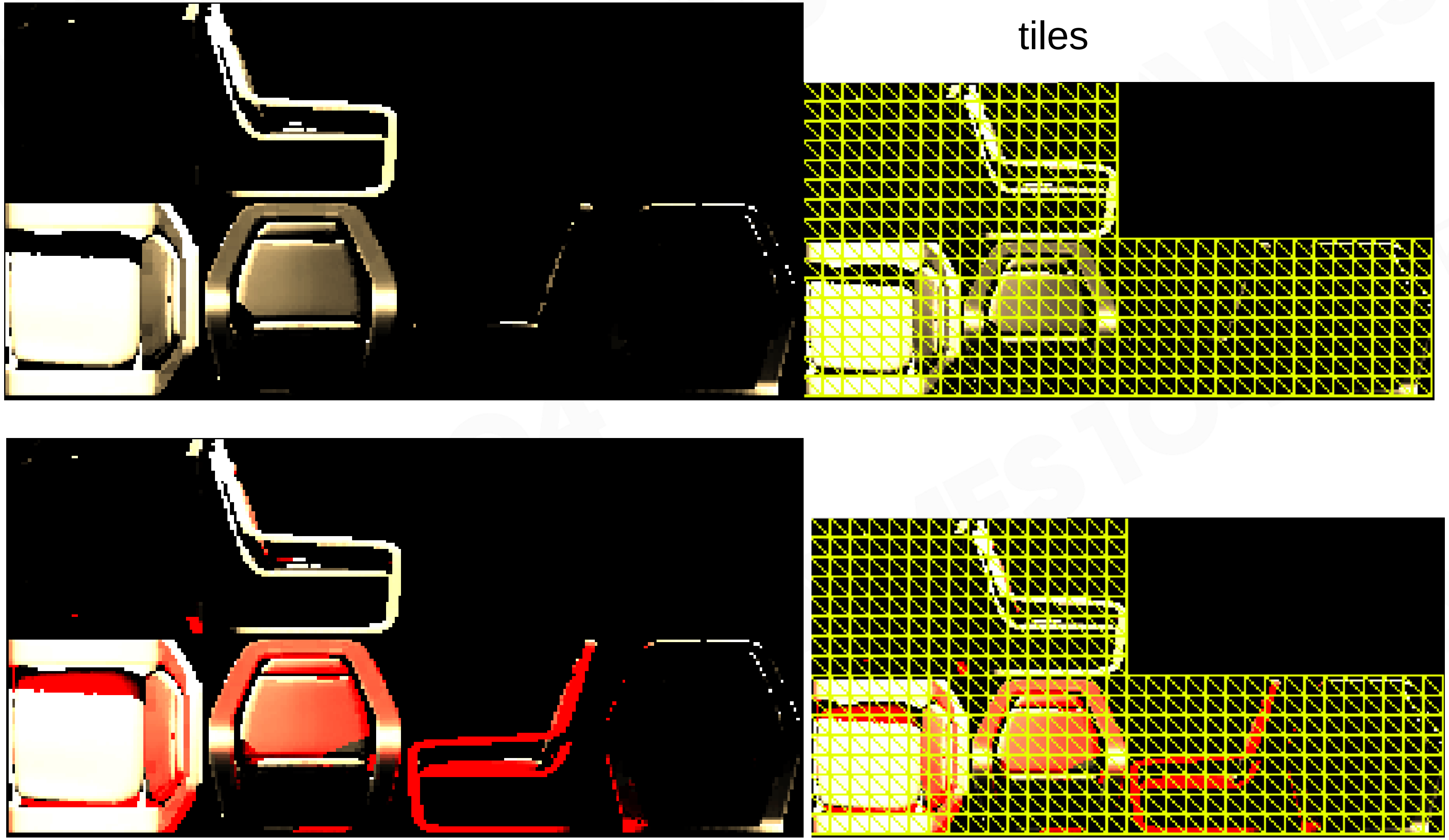


Direct Lighting

- Divide 128x128 page into 8x8 tiles
- Cull lights with 8x8 tile
- Select first 8 lights per tile
- 1 bit shadow mask



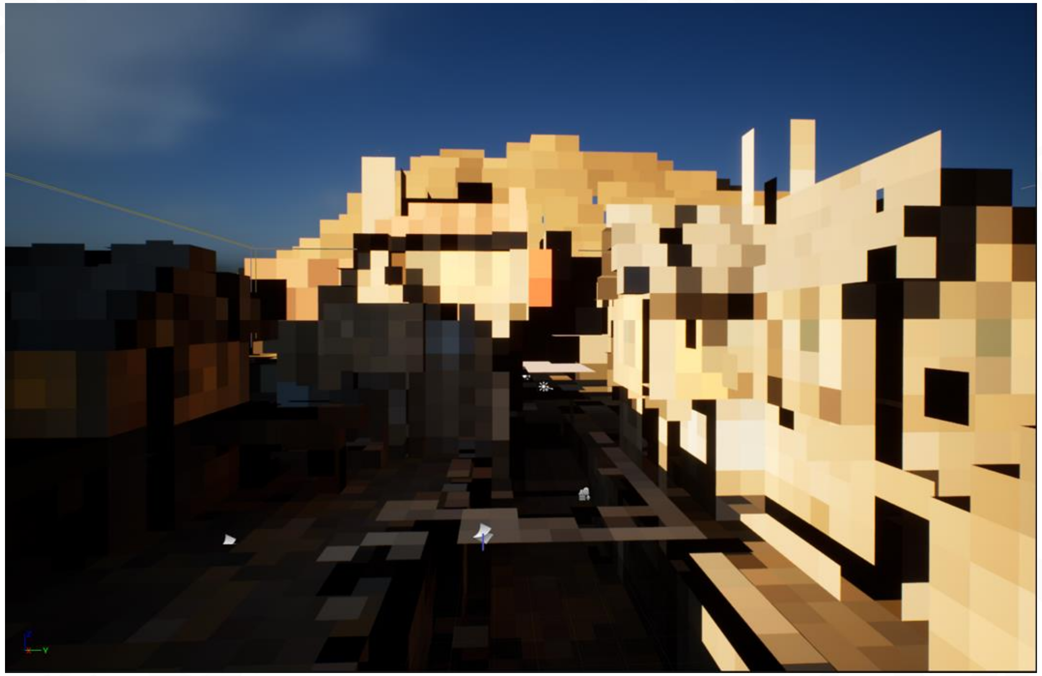
One tile can be lited by multi lights, the result will be accumlated



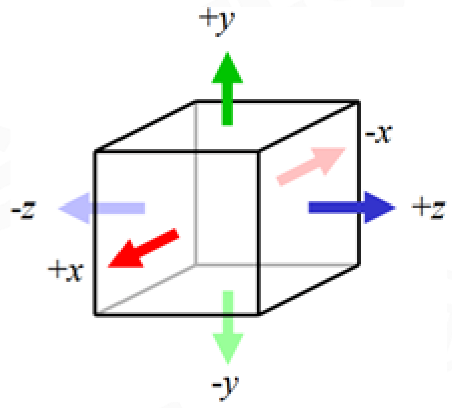
Global SDF can't sample surface cache

- no per mesh information, only hit position and normal

Use voxel lighting to sample



Voxel Clipmap for Radiance Caching of the Whole Scene

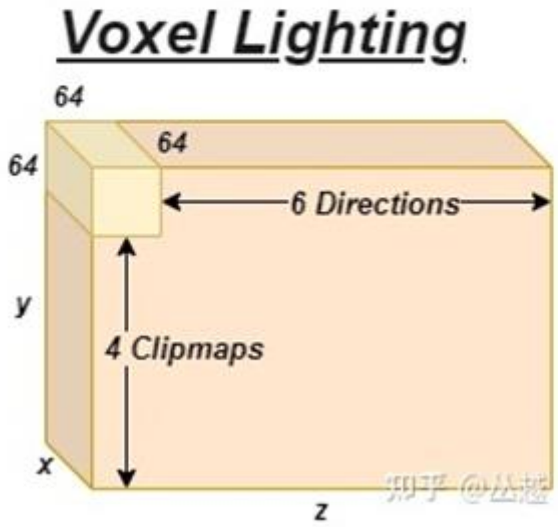


4 level clipmaps of 64x64x64 voxels

- Radiance per 6 directions per voxel
- Sample and interpolate 3 directions by normal
- Clipmap0 cover 50m^3, voxel size is 0.78m
- Store in 3D texture

Clipmap update frequency rules

	Clipmap 0	Clipmap 1	Clipmap 2	Clipmap3
Start_Frame	0	1	3	7
Update_interval	2	4	8	8

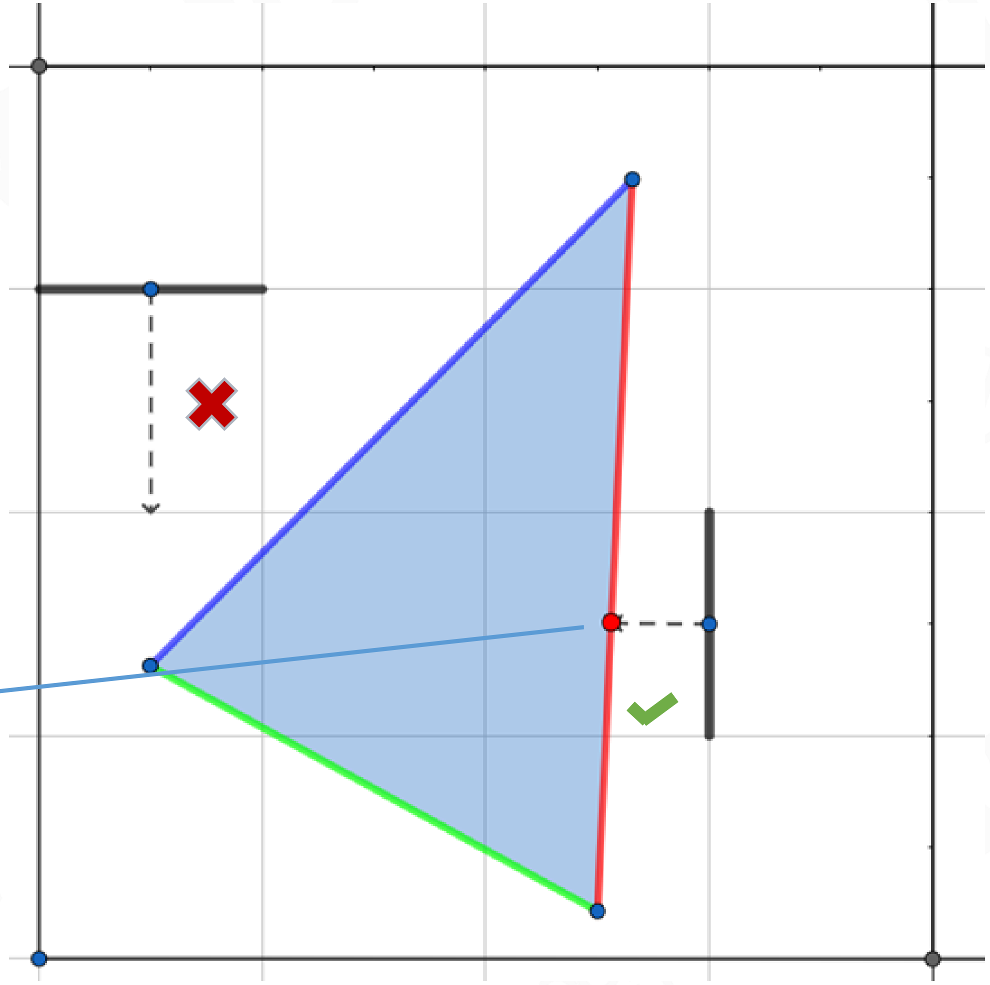


Build Voxel Faces by Short Ray cast

- Trace mesh DF on 6 directions per voxel
- Hit mesh id and hit distance
- RayStart=VoxelCenter – AxisDir *VoxelRaidus
- RayEnd=VoxelCenter + AxisDir *VoxelRaidus

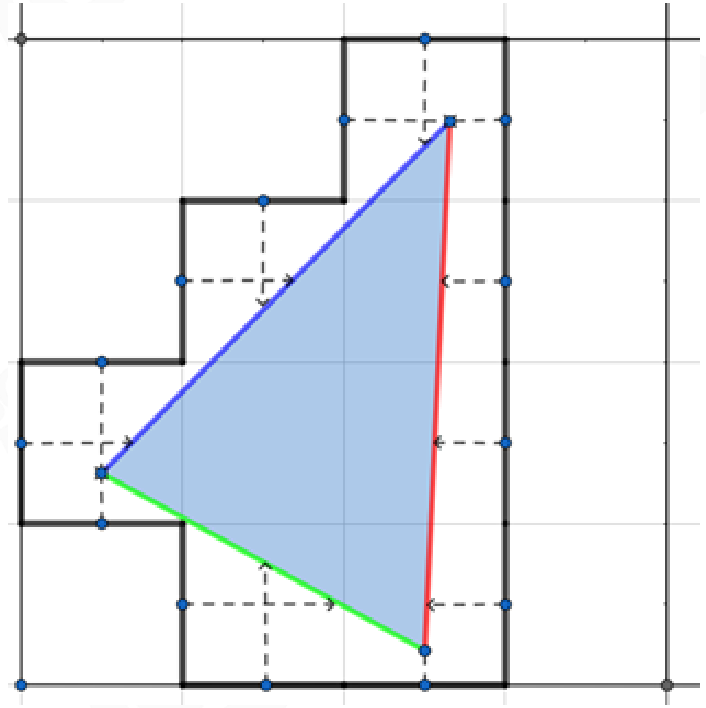
store hit infro into visibilty buffer

uint32 [Hit distance| Hit object id]

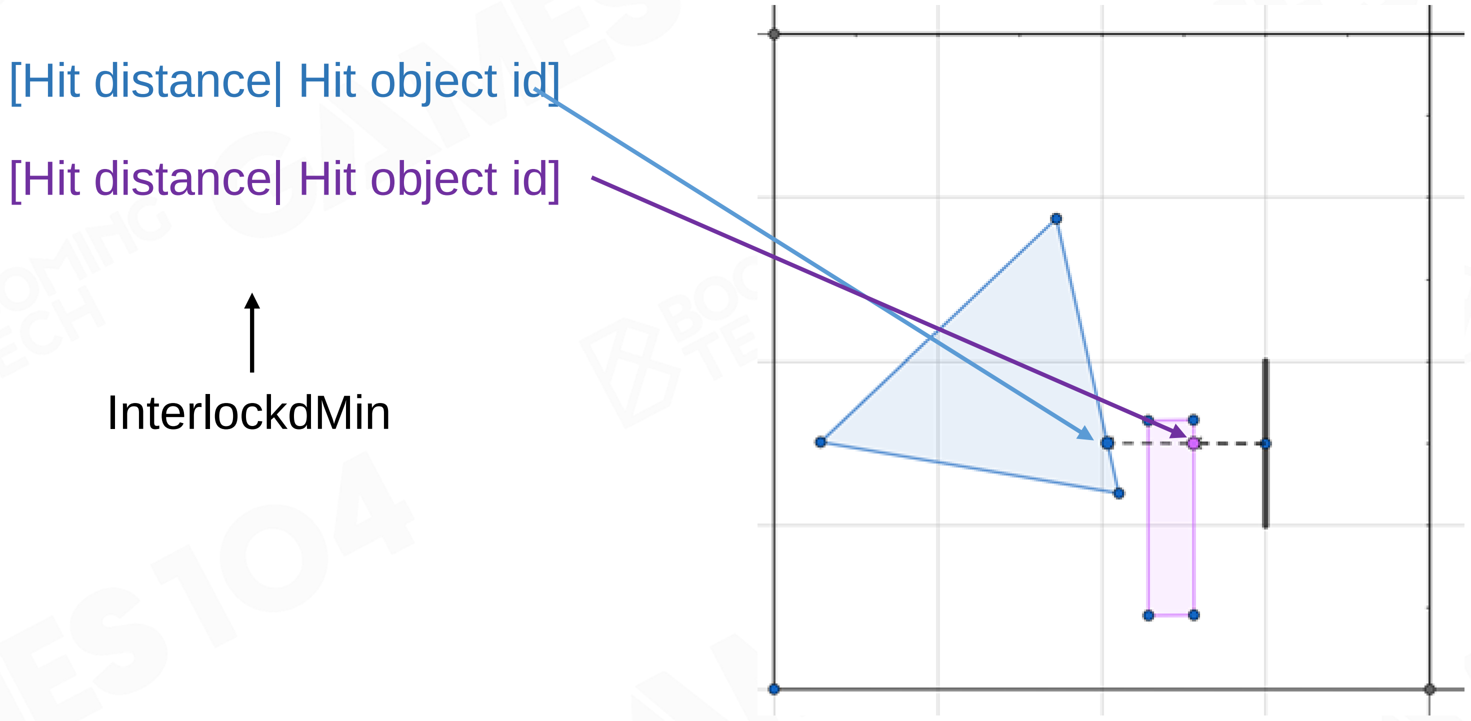


Tile (4x4x4 voxels)

Filter Most Object Out by 4x4x4 Tiles

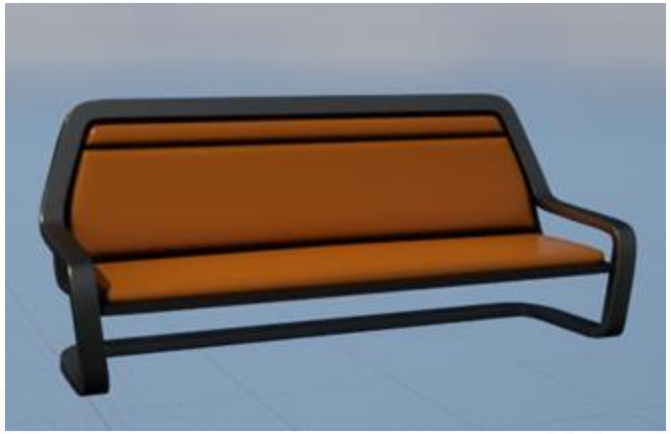


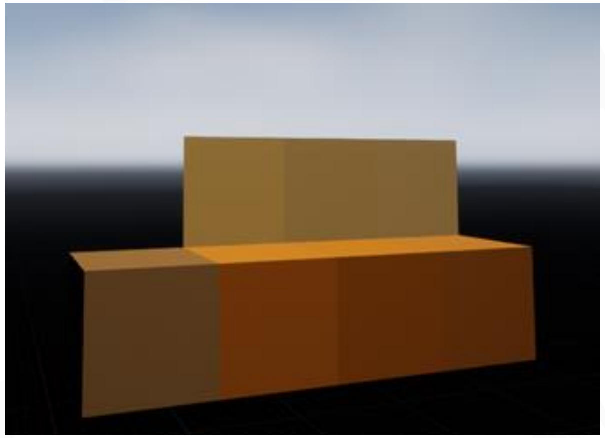
Tile (4x4x4 voxels)

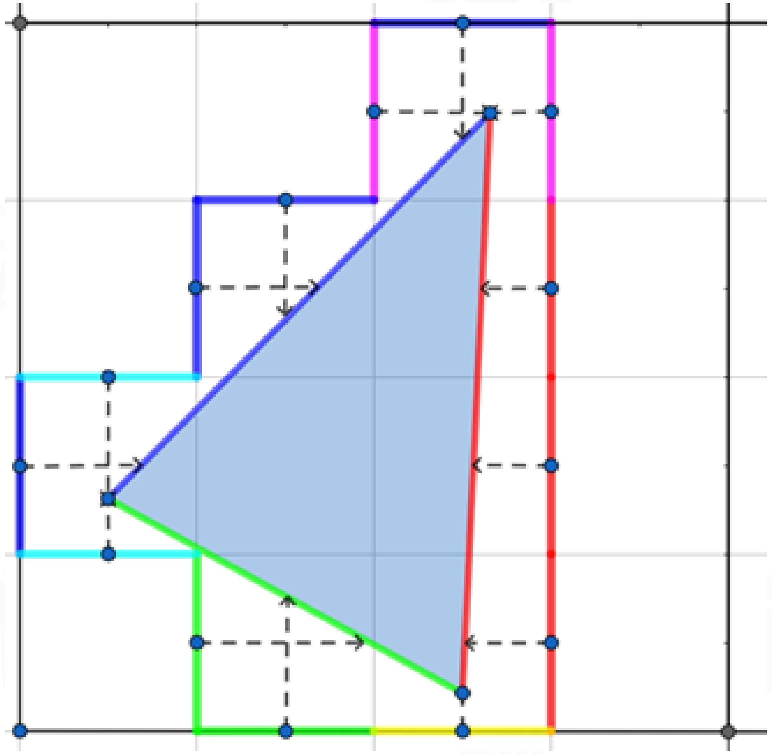


Inject light into clipmap

- Clear all voxel lighting in entire Clipmap
- Compact all valid VisBuffer in Clipmap
- Sampling FinalLighting from VisBuffer and inject lighting



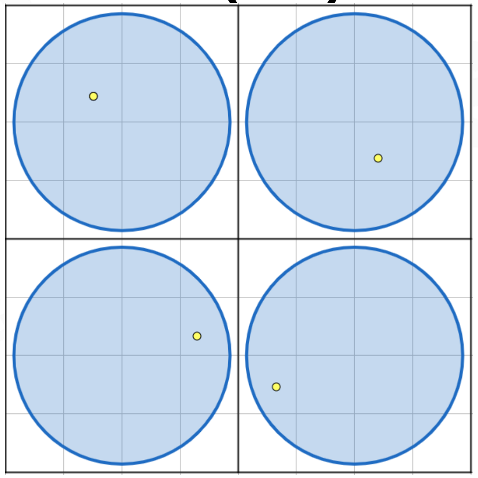




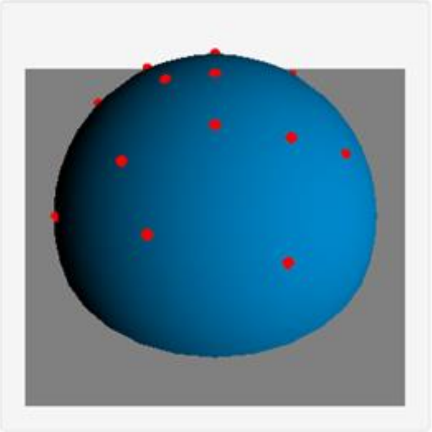
Indirect Lighting

- Place 2x2 probes on each tile - each probe cover 4x4 texels
- Trace 16 rays from heimisphere per probe
- Jitter probe placement and ray directions

Tile(8x8)



probe

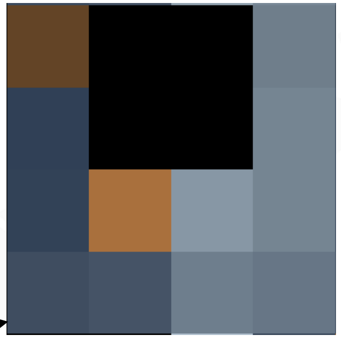


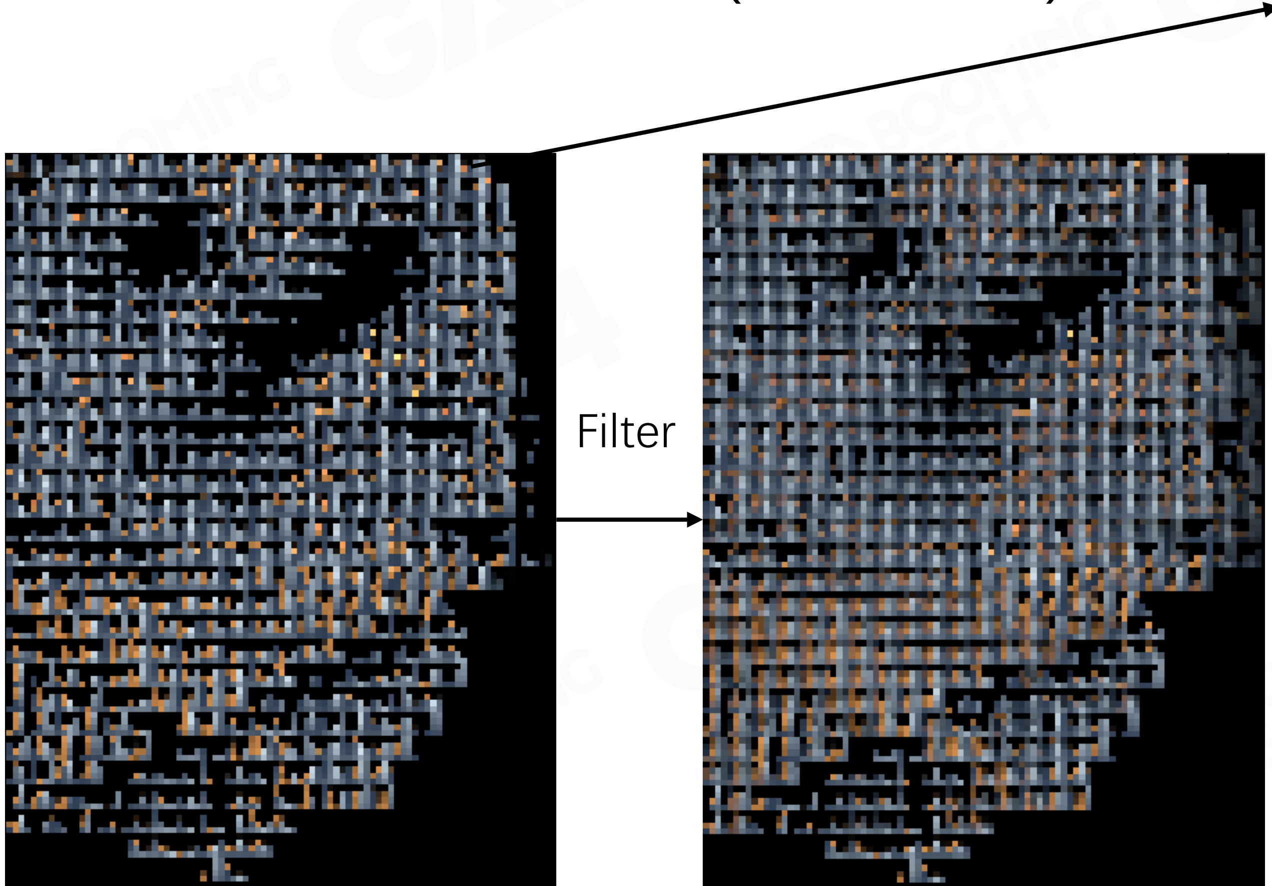
16 rays

Indirect Lighting

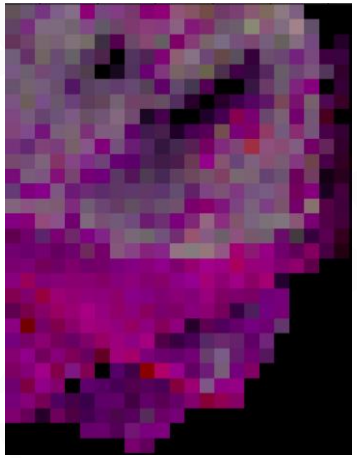
- Spatial filtering between probes
- Convert to TwoBandSH(store in half4)

4x4 radiance altas per probe

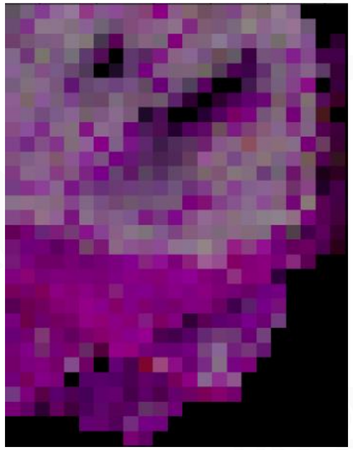




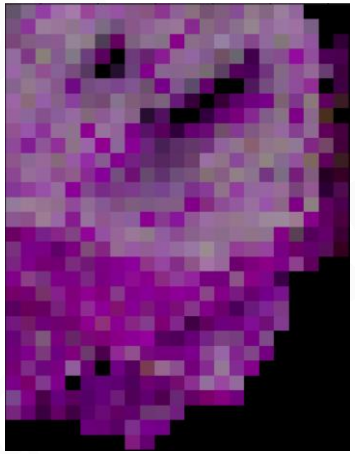
ProbeSHRed



ProbeSHGreen



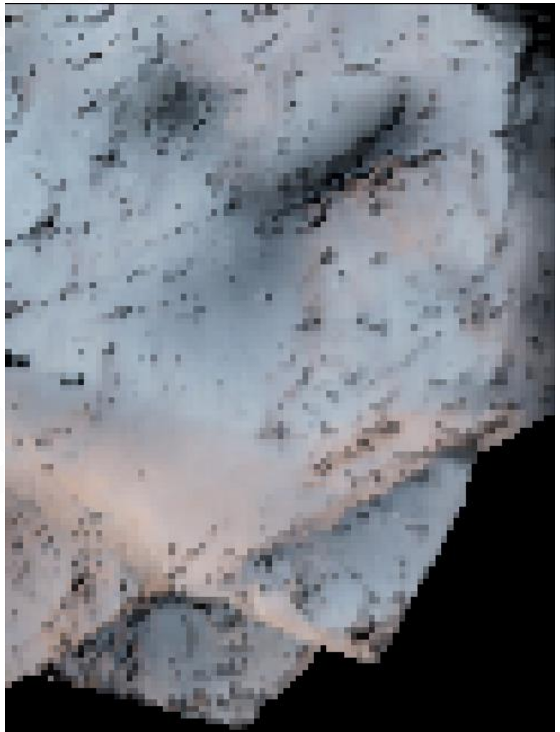
ProbeSHBlue



Per-Pixel Indirect Lighting with 4 Probe Interpolation

- Integrate on pixel - bilinear interpolation of 4 neighbor probes

IndirectLighting



DirectLighting

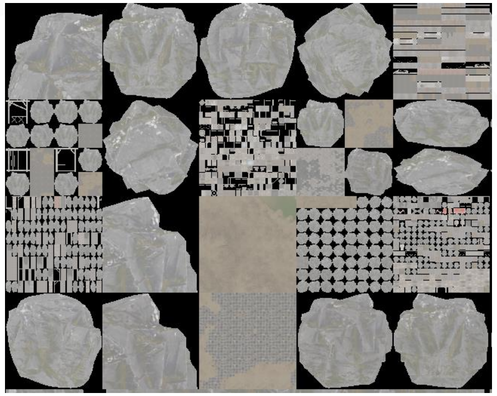
Albedo

FinalLighting

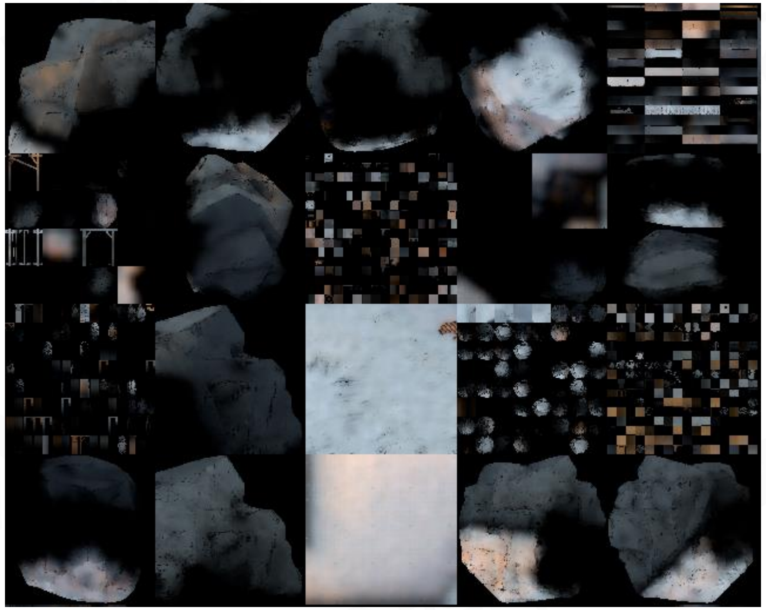
Combine Lighting

FinalLighting = (DirectLighting + IndirectLighting) * Diffuse_Lambert(Albedo) + Emissive;

Albedo



DirectLighting (HDR)



IndirectLighting (LDR)

FinalLighting

Ligting Update Strategy

Fix budget

- 1024x1024 texels for direct lighting
- 512x512 texels for indirect lighting
- Select pages to update based on Priority = LastUsed - LastUpdated

Priority queue using bucket sort

- 128 buckets
- Update buckets with priority until reaching budget

Phase 3 : Build a lot of Probes with Different Kinds

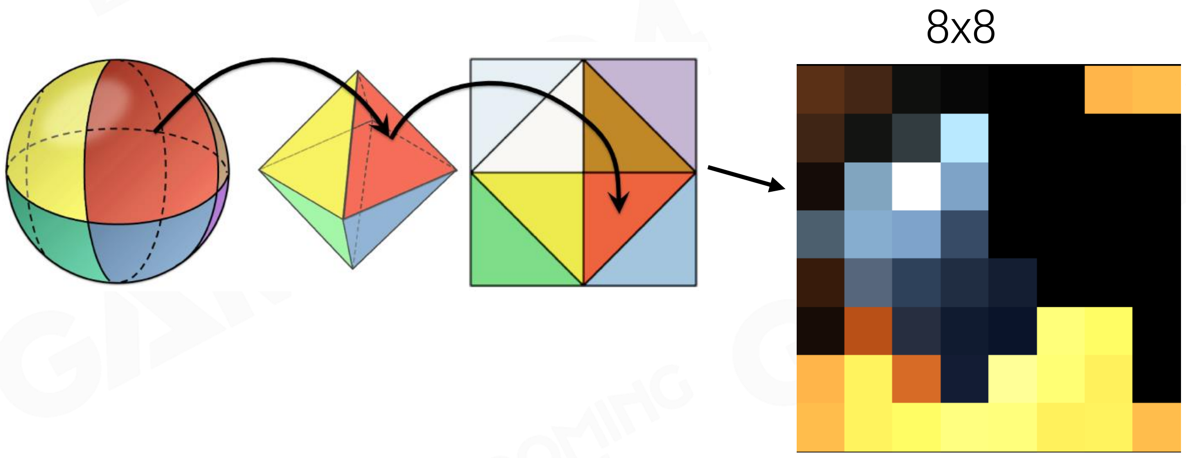
Screen Space Probe

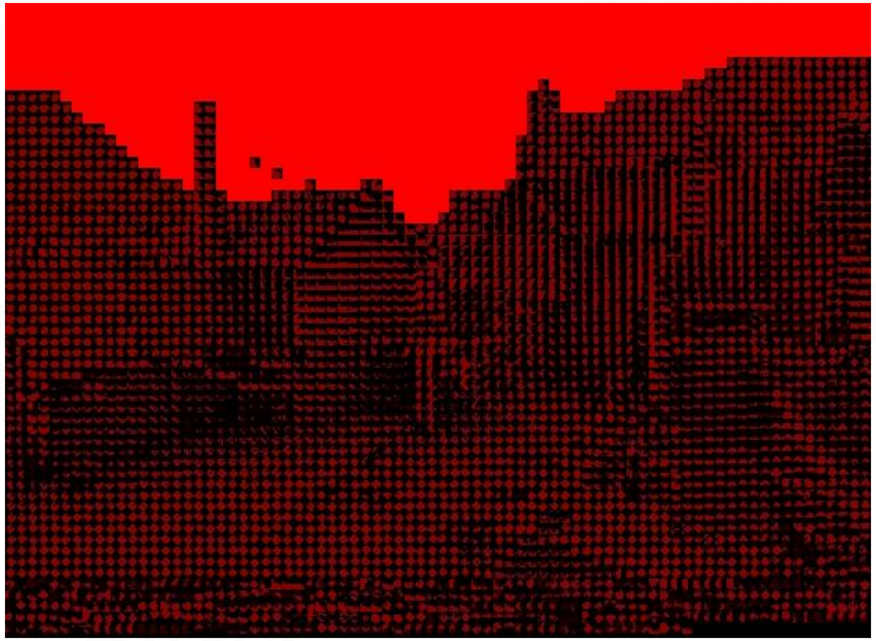
Screen Probe structure

Octahedral atlas with border

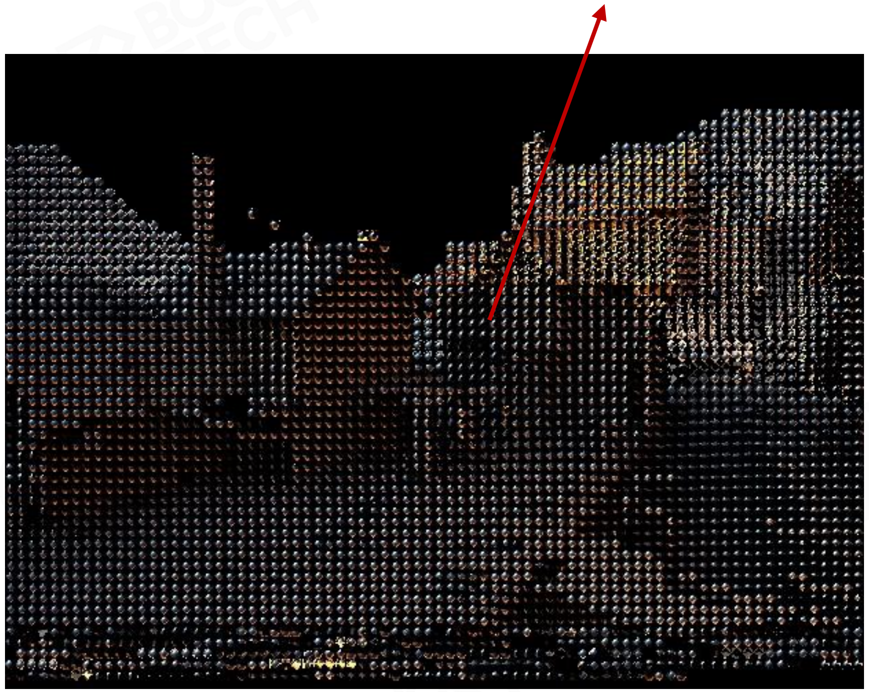
- Typically 8x8 per probe
- Uniformly distributed world space directions
- Neighbors have matching directions

Radiance and HitDistance in 2d atlas



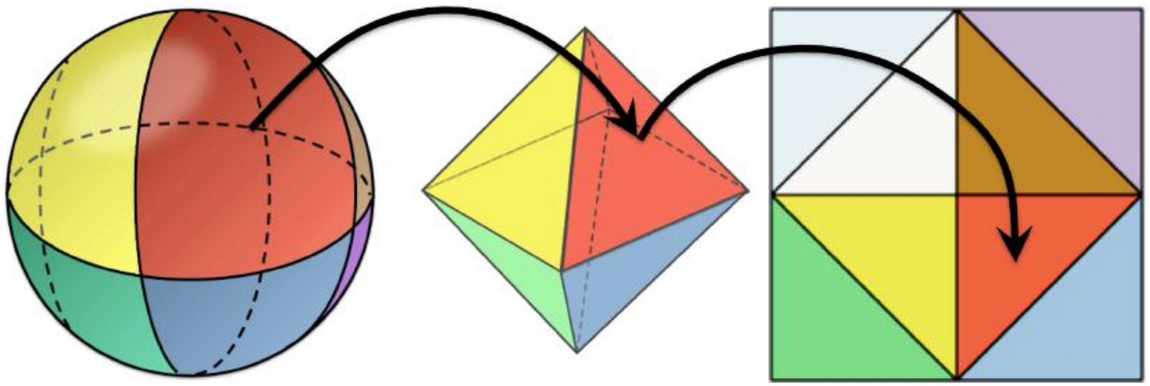


Hit Distance



Radiance Altas

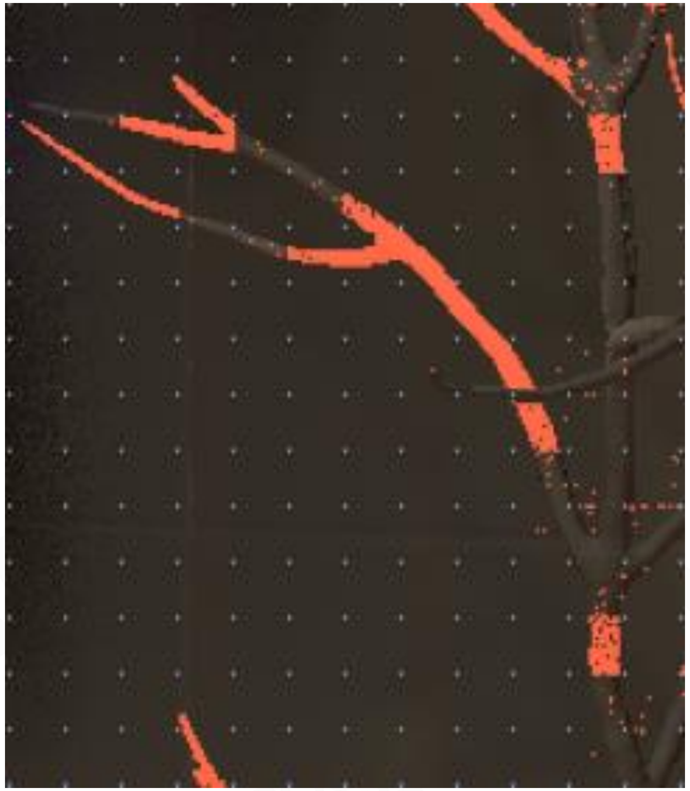
Octahedron mapping



```
float2 unitVectorToOctahedron(float3 N)
{
    N.xy /= dot(1, abs(N));
    if (N.z <= 0)
    {
        x_factor = N.x >= 0? 1.0 : -1.0;
        y_factor = N.y >= 0 ? 1.0 : -1.0;
        N.xy = (1 - abs(N.yx)) * float2(x_factor, y_factor);
    }
    return float2(N.xy);
}
```

Screen Probe Placement

- Adaptive placement with Hierarchical Refinement
- Iteratively place where interpolation fails

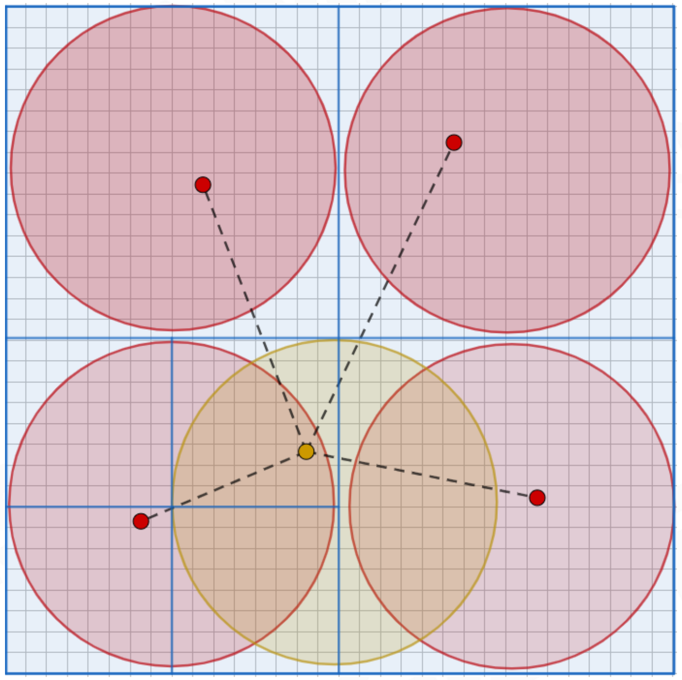


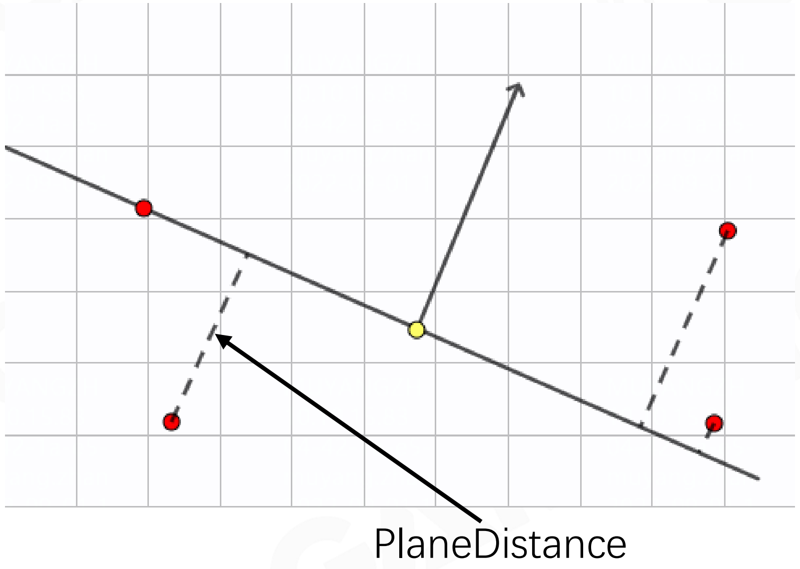
16x16

8x8

4x4

Plane distance weighting of Probe Interpolation



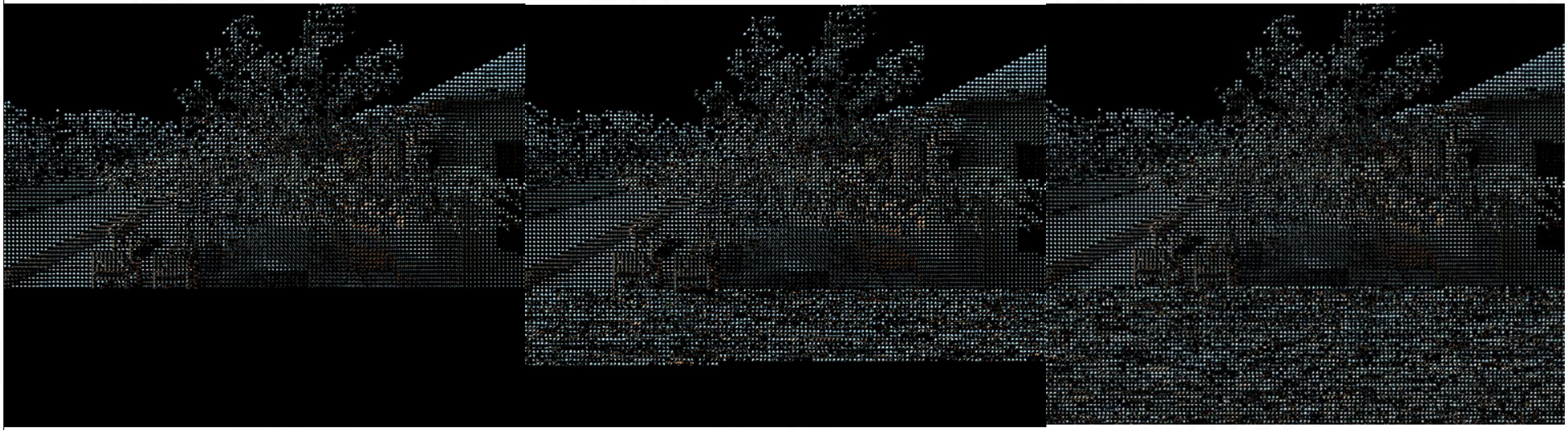


Detect Non-Interpolatable Cases

```
·float4·PlaneDistances;
·PlaneDistances.x·=·abs(dot(float4(Position00,·-1),·ScenePlane));
·PlaneDistances.y·=·abs(dot(float4(Position10,·-1),·ScenePlane));
·PlaneDistances.z·=·abs(dot(float4(Position01,·-1),·ScenePlane));
·PlaneDistances.w·=·abs(dot(float4(Position11,·-1),·ScenePlane));

·float4·RelativeDepthDifference·=·PlaneDistances·/·SceneDepth;

·DepthWeights·=·CornerDepths·>·0·?·exp2(-10000.0f·*·(RelativeDepthDifference·*·RelativeDepthDifference))·:·0;

InterpolationWeights·=·float4(
····(1·-·BilinearWeights.y)·*·(1·-·BilinearWeights.x),
····(1·-·BilinearWeights.y)·*·BilinearWeights.x,
····BilinearWeights.y·*·(1·-·BilinearWeights.x),
····BilinearWeights.y·*·BilinearWeights.x);
InterpolationWeights·*=·DepthWeights;

float·Epsilon·=·.01f;
ScreenProbeSample.Weights·/=·max(dot(ScreenProbeSample.Weights,·1),·Epsilon);

float·LightingIsValid·=·(dot(ScreenProbeSample.Weights,·1)·<·1.0f·-·Epsilon)·?·0.0f·:·1.0f;
```

Screen Probe Atlas

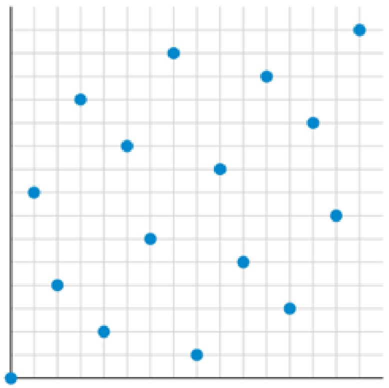
- Atlas have upper limit for real-time
- Place adaptive probes at the bottom of the atlas



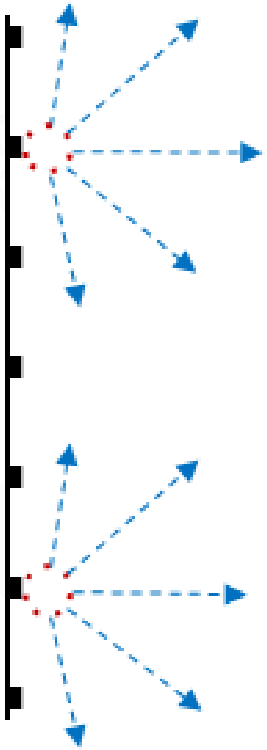
16x16 8x8 4x4

Screen Probe Jitter

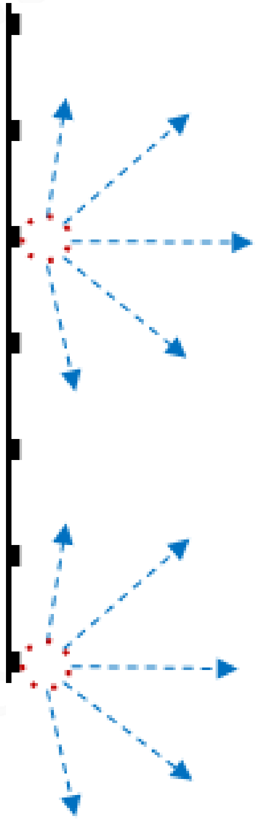
- Place probe directly on pixels
- Temporally jitter placement and direction
- Use Hammersley points in [0, 15]



Hammersley Points in [0-15]



Frame 1



Frame 2

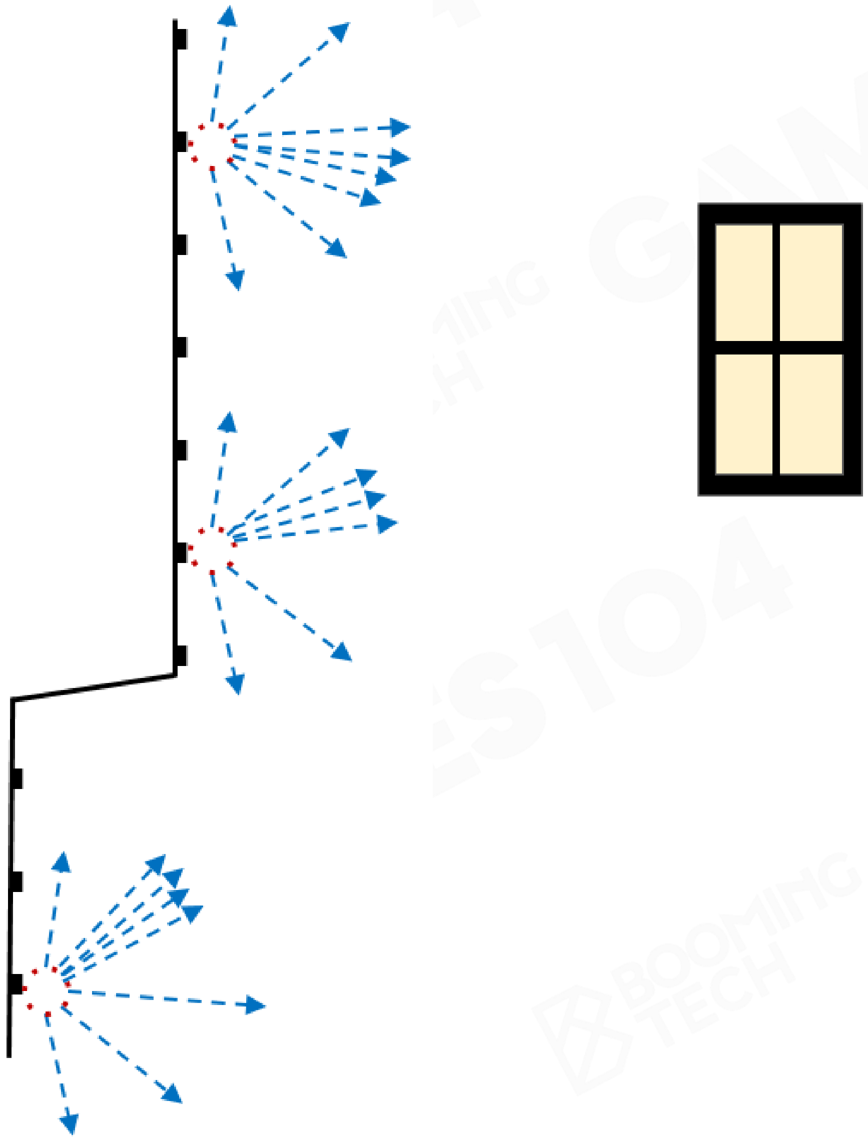
Frame 0

Temporal accumulation

Importance Sampling

But too much noise at ½ ray per pixel

Better sampling - importance sample incoming lighting and BRDF



Importance Sampling

$$\lim_{N\to\infty} \frac{1}{N} \sum_{k=1}^{N} \frac{\boxed{L_i(l)}\,\boxed{f_s(l \to v)\cos(\theta l)}}{P_k}$$

We would like to distribute rays proportional to the integrand

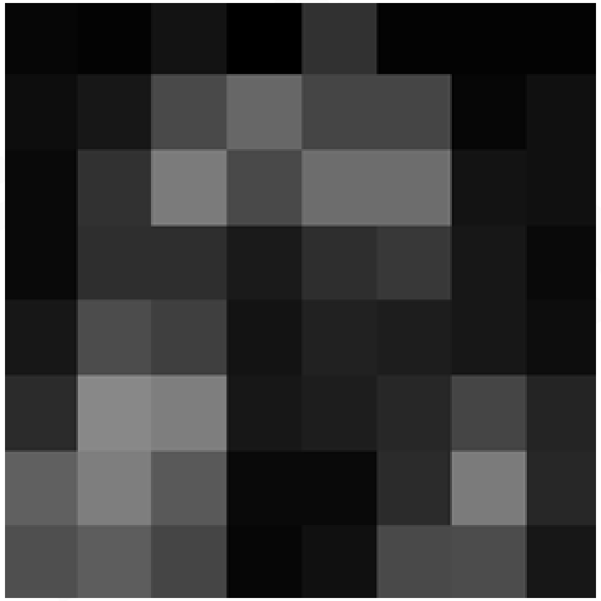
How can we estimate these?

Approximate Radiance Importance from Last Frame Probes

$$\lim_{N\to\infty}\frac{1}{N}\sum_{k=1}^{N}\frac{\boxed{L_i(l)}\,f_s(l\to v)\cos(\theta l)}{P_k}$$

Incoming Radiance:

- Reproject to last frame and average the four neighboring Screen Probes Radiance
- No need to do an expensive search, as rays already indexed in octahedral atlas
- Fallback to World Space Probe Radiance if neighboring probes are occluded



Accumulate Normal Distribution Nearby

$$\lim_{N\to\infty}\frac{1}{N}\sum_{k=1}^{N}\frac{L_i(l)\boxed{f_s(l\to v)\cos(\theta l)}}{P_k}$$

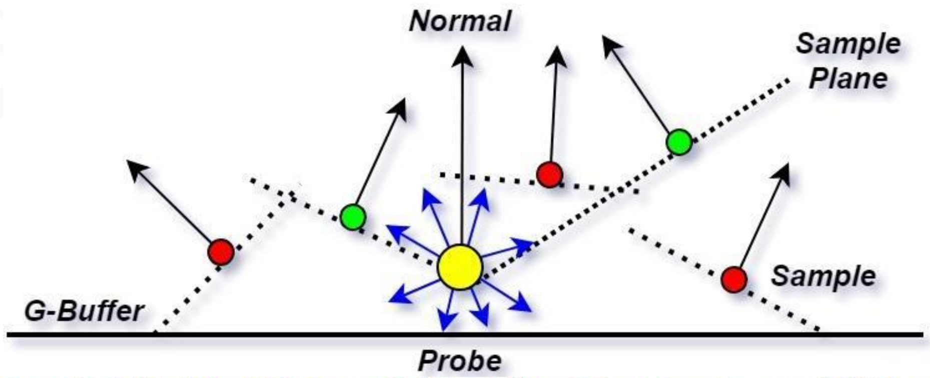
BRDF:

- For a probe that's placed on a flat wall, about half of its sphere having a zero BRDF
- Accumulate from pixels that will use this Screen Probe

Nearby Normal Accumulation

- Gathter 64 neighbor pixels around current probe's pixel in a 32x32 pixel range
- Accept pixel if its depth weight > 0.1
- Accumulate these pixels' world normal into SH

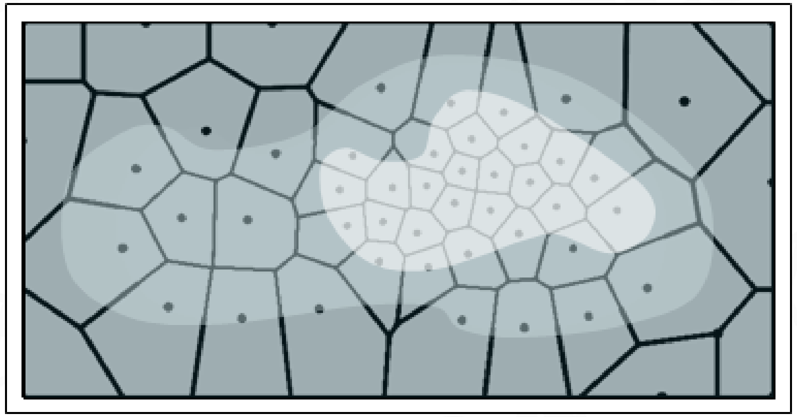
```
if (DepthWeight > .1f || bCenterSample)
{
    uint Index;
    InterlockedAdd(NumSphericalHarmonics, 1, Index);

    FThreeBandSHVector BRDF;
    if (HasSphericalVisibility(Material))
    {
        // Avoid culling directions that the shading models will sample
        BRDF = (FThreeBandSHVector)0;
        BRDF.V0.x = 1.0f;
    }
    else
    {
        BRDF = CalcDiffuseTransferSH3(Material.WorldNormal, 1.0f);
    }
    WriteGroupSharedSH(BRDF, Index);
}
```




```
float3 PixelPosition = GetWorldPositionFromScreenUV(PixelScreenUV, Material.SceneDepth);
float4 PixelPlane = float4(Material.WorldNormal, dot(Material.WorldNormal, PixelPosition));
float3 ProbeWorldPosition = GetWorldPositionFromScreenUV(ScreenUV, ProbeSceneDepth);

float PlaneDistance = abs(dot(float4(ProbeWorldPosition, -1), PixelPlane));
float RelativeDepthDifference = PlaneDistance / ProbeSceneDepth;
float DepthWeight = exp2(-10000.0f * (RelativeDepthDifference * RelativeDepthDifference));
```

Structured Importance Sampling

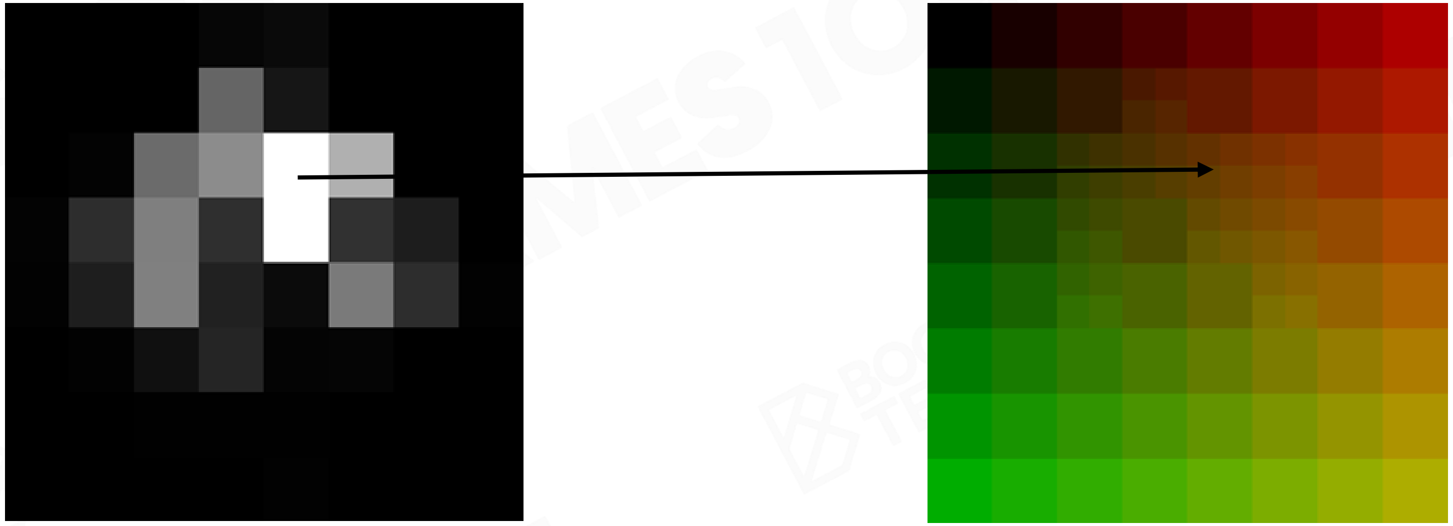
- Assigns a small number of samples to hierarchically structured areas of the Probability Density Function (PDF)
- Achieves good global stratification
- Sample placement requires offline algorithm



Maps perfectly to Octahedral mip quadtree!

PDF

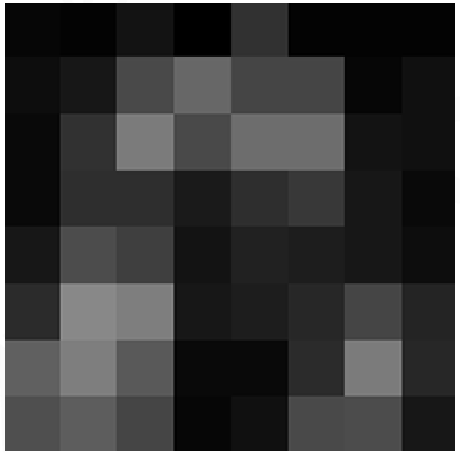
Subdivided Octahedral texels



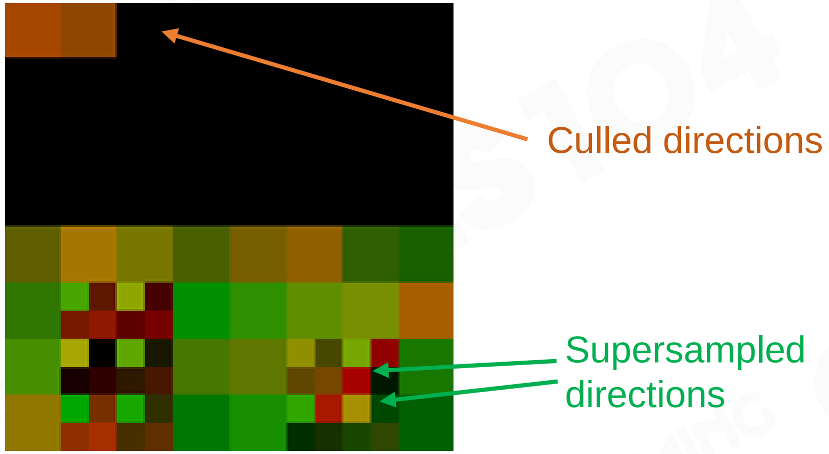
Fix Budget Importance Sampling based on Lighting and BRDF

- Start with uniformly distributed probe ray directions
- Fixed probe tracing ray count=64
- Calculate BRDF PDF * Lighting PDF for each Octahedral texel
- Sort rays by PDF from low to high
- For every 3 rays with PDF below cull threshold, supersample the matching highest PDF ray

BRDF PDF



Lighting PDF



uniform ray directions

ray directions after importance sampling

Denoising and Spatial Probe Filtering

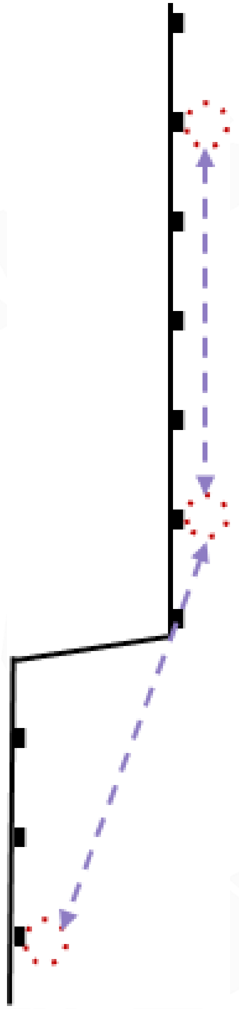
Denoise: Spatial filtering for Probe

Large spatial filter for cheap

- Each probe cover 16x16 pixels, 3x3 filtering kernel in probe space equals 48x48 in screen space

Can ignore normal differences between spatial neighbors

- Only depth weighting



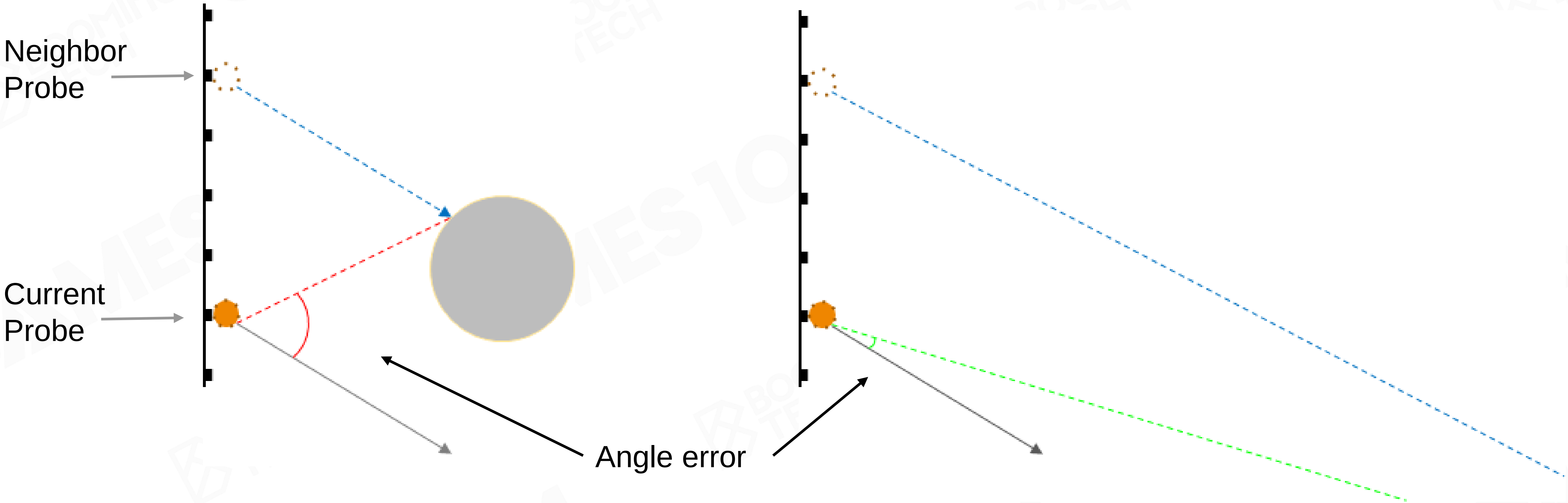
```
float GetFilterPositionWeight(float ProbeDepth, float SceneDepth)
{
    float DepthDifference = abs(ProbeDepth - SceneDepth);
    float RelativeDepthDifference = DepthDifference / SceneDepth;
    return ProbeDepth >= 0 ? exp2(-SpatialFilterPositionWeightScale * (RelativeDepthDifference * RelativeDepthDifference)) : 0;
}
```

Denoise: Gather Radiance from neighbors

Gather radiance from matching Octahedral cell in neighbor probes

Error weighting:

- Angle error from reprojected neighbor ray hits (less than 10 degree)
- Filters distant lighting, preserves local shadowing

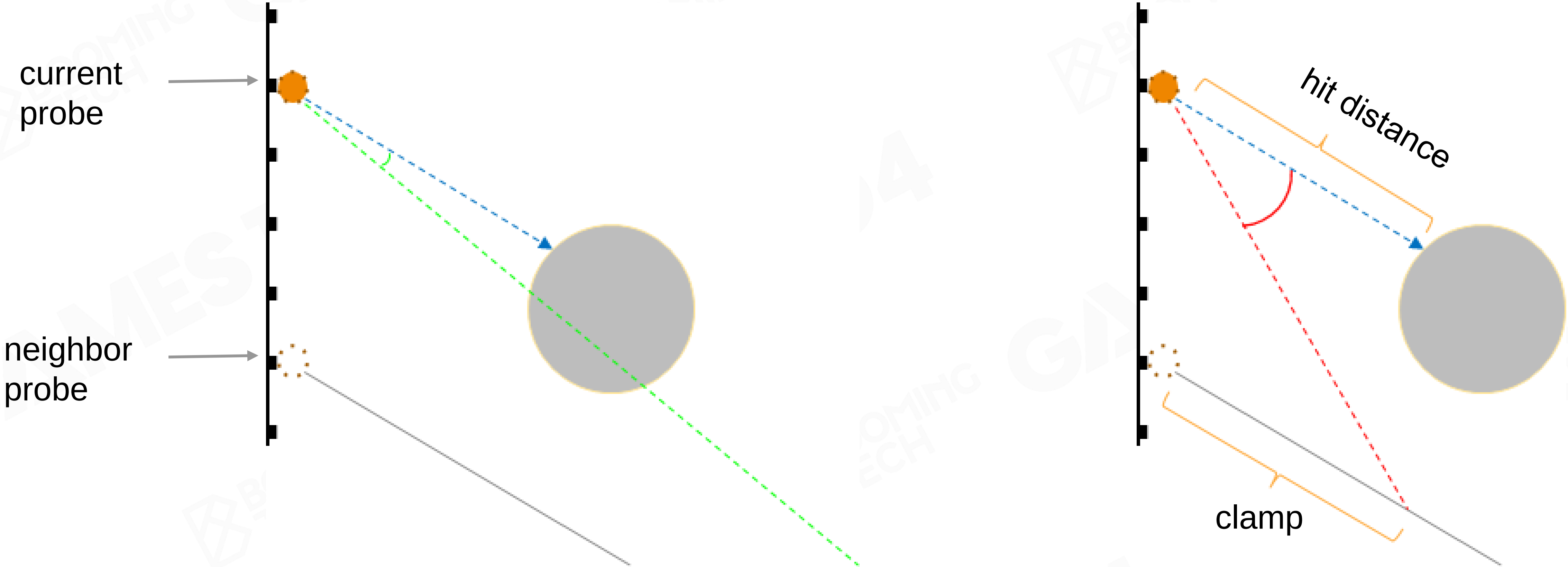


Clamp Distance Mismatching

Angle error biases toward distant light = leaking

- Distant light has no parallax and never gets rejected

Solution: clamp neighbor hit distance to our own before reprojection



Final filtering compare

World Space Probes and Ray Connecting

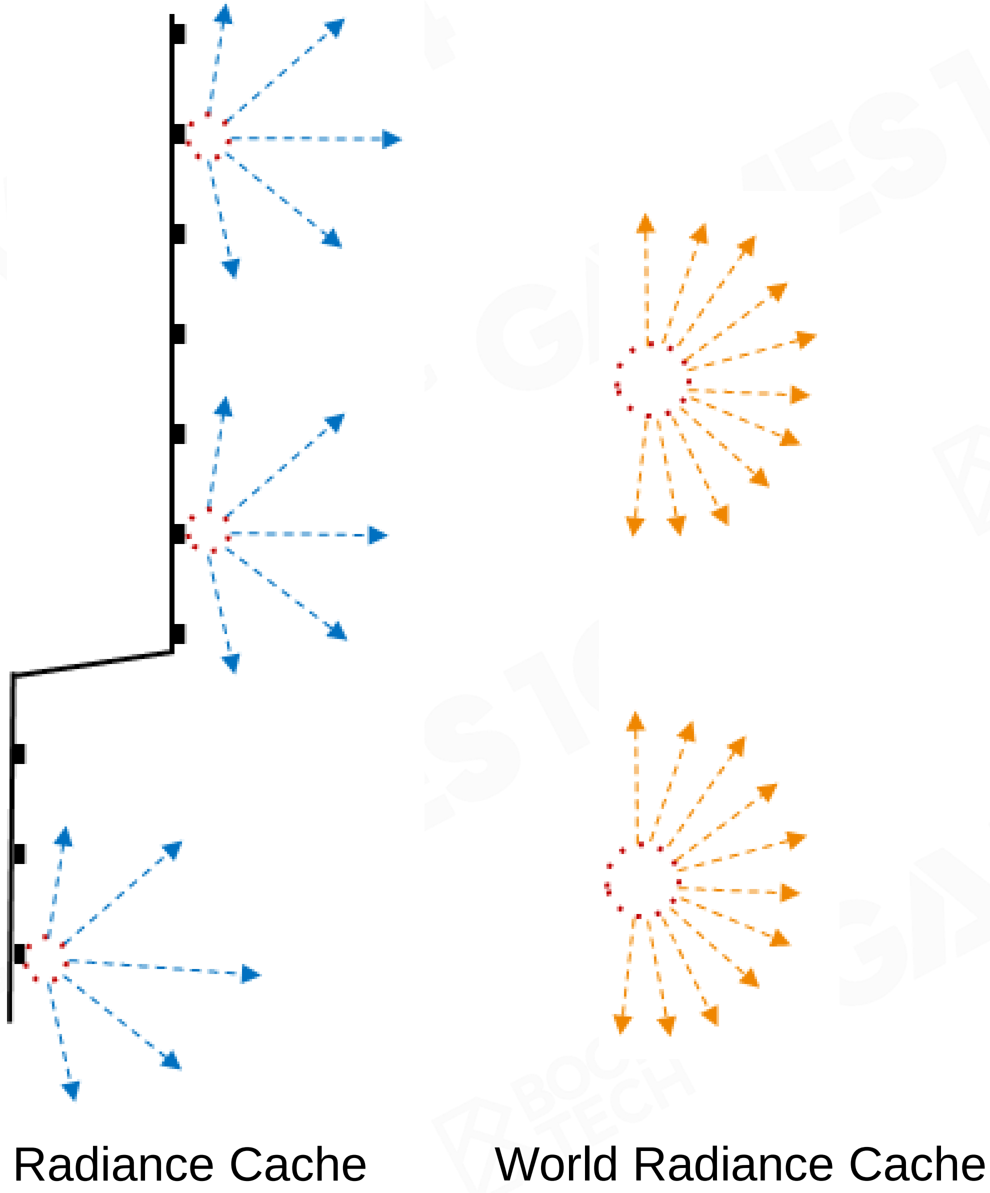
World Space Radiance Cache

Problem: distant lighting

- Noise from small bright feature increases with distance
- Long incoherent traces are slow
- Distant lighting is changing slowly - opportunity to cache
- Redundant operations for nearby Screen Probes

Solution: separate sampling for distant Radiance

- World space Radiance Caching for distant lighting
- Stable error since world space - easy to hide



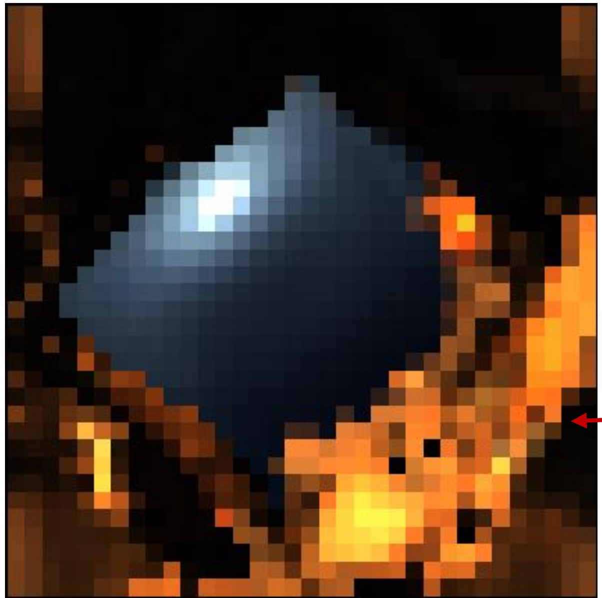
World Space Radiance Cache

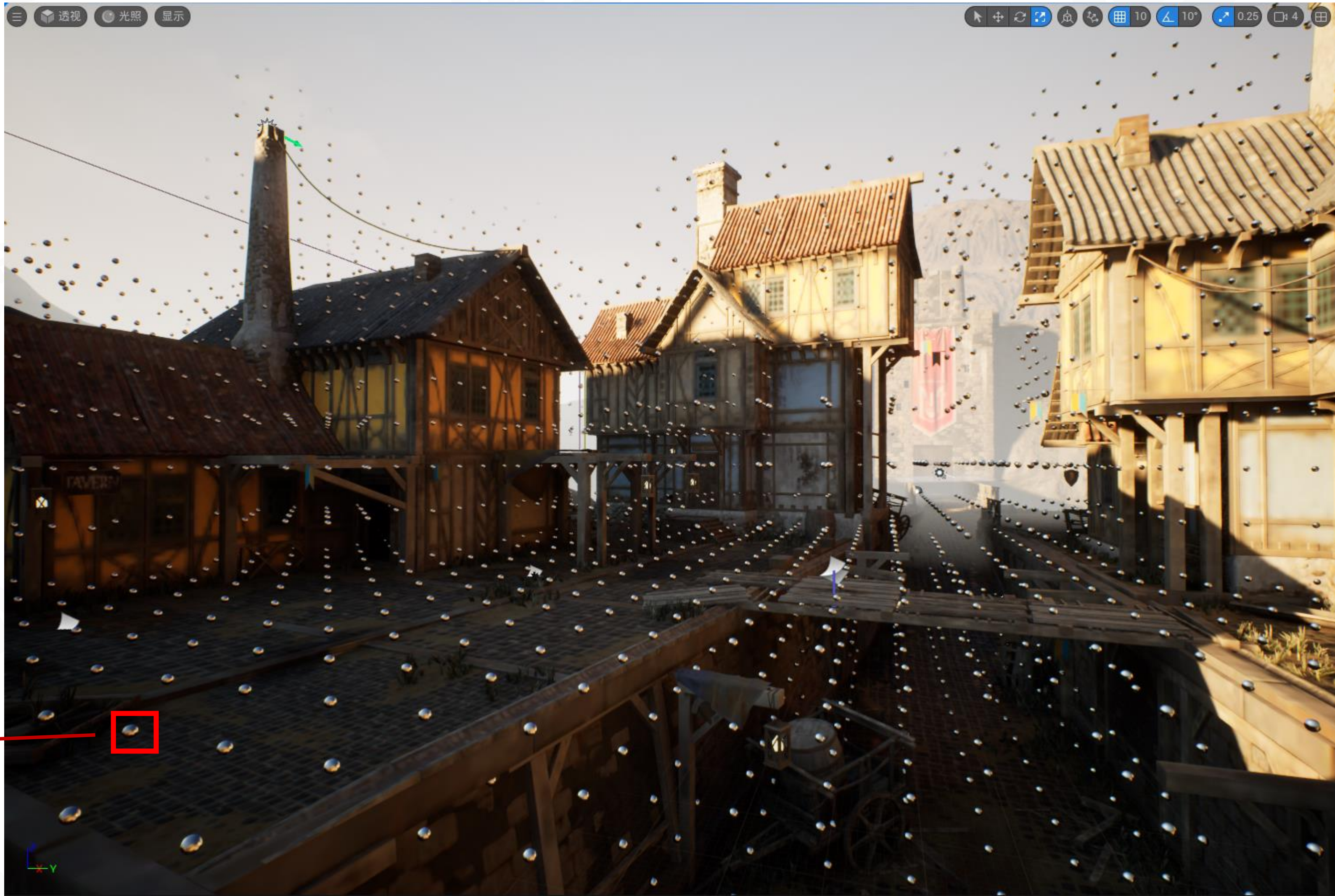
Placement

- 4 level clipmaps around camera
- default resolution is 48^3
- clipmap 0 size is 50m^3

Radiance

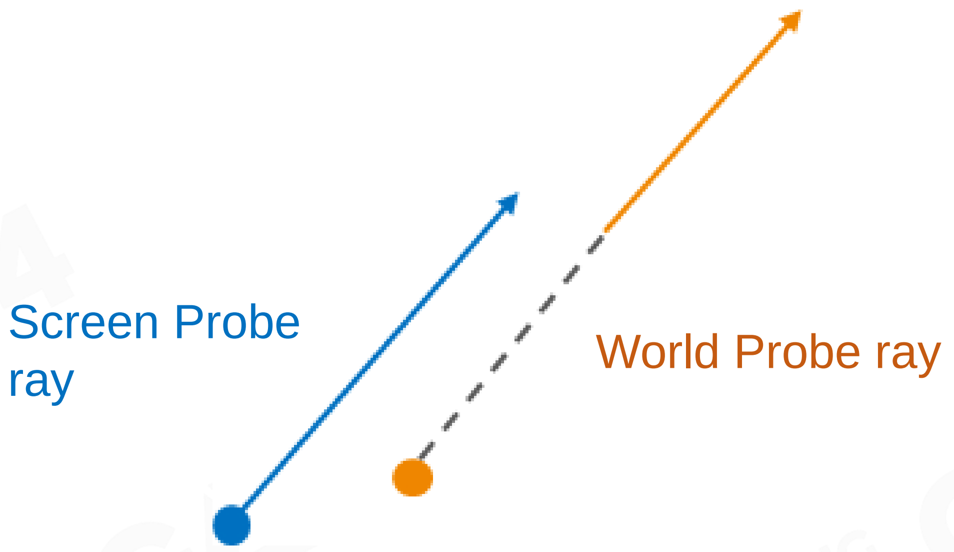
- 32x32 atlas a per probe





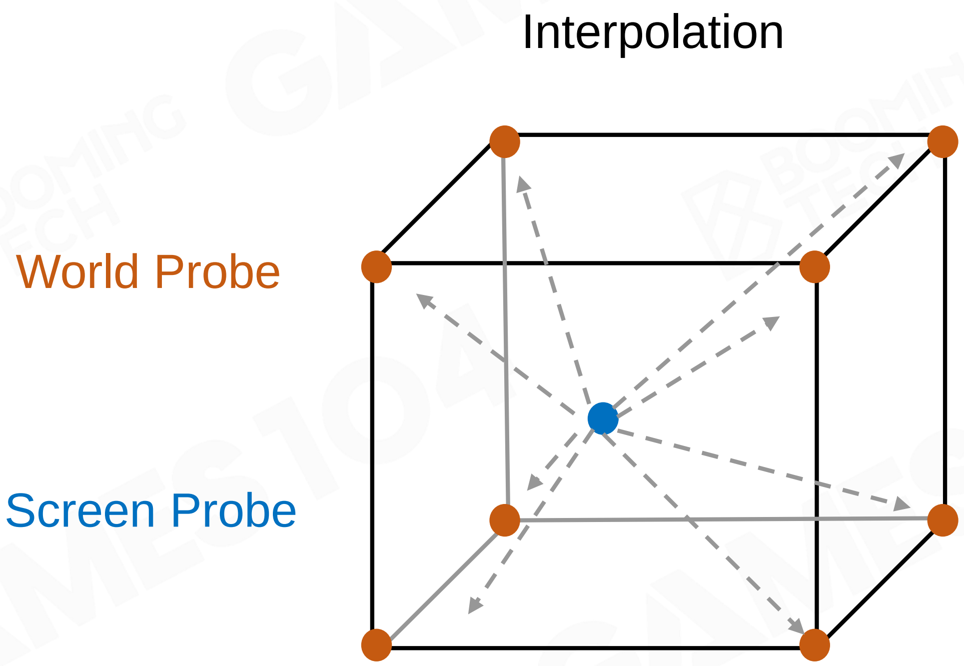
Connecting rays

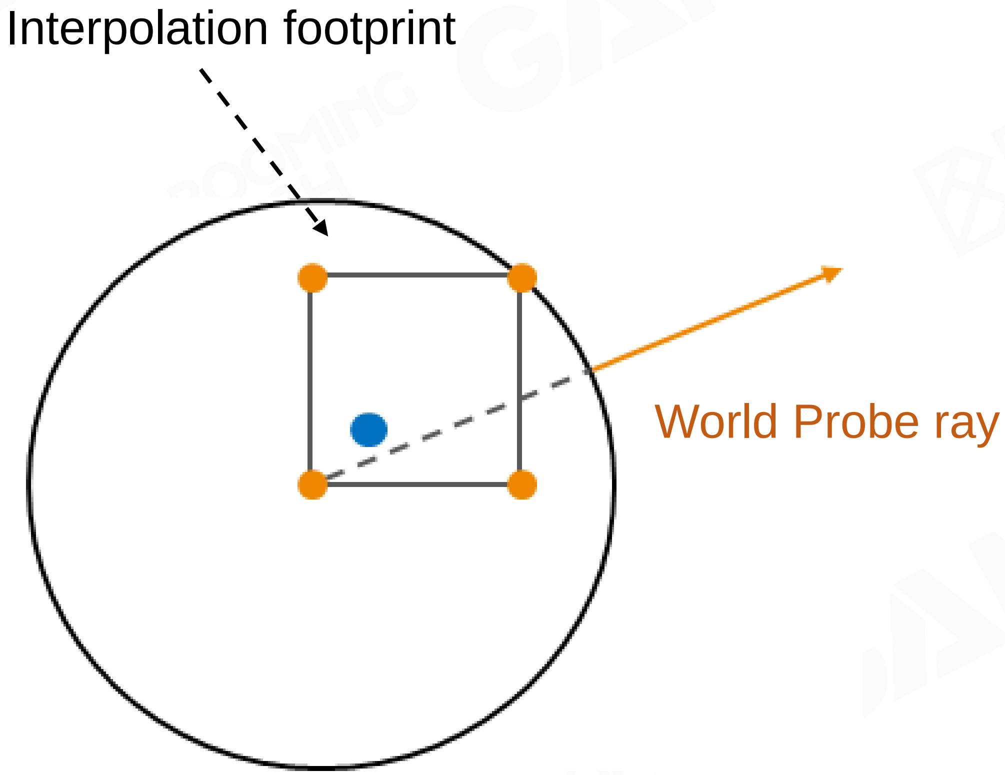
- How to connect Screen Probe ray and World Probe ray



Connecting rays

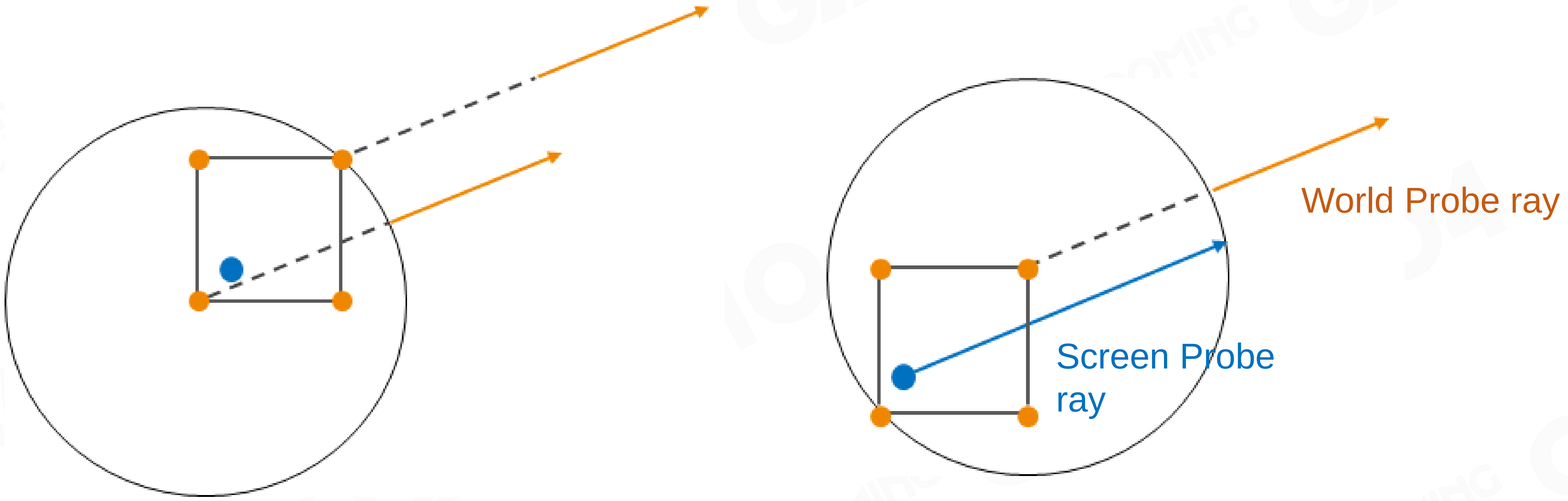
- World Probe ray must skip the interpolation footprint





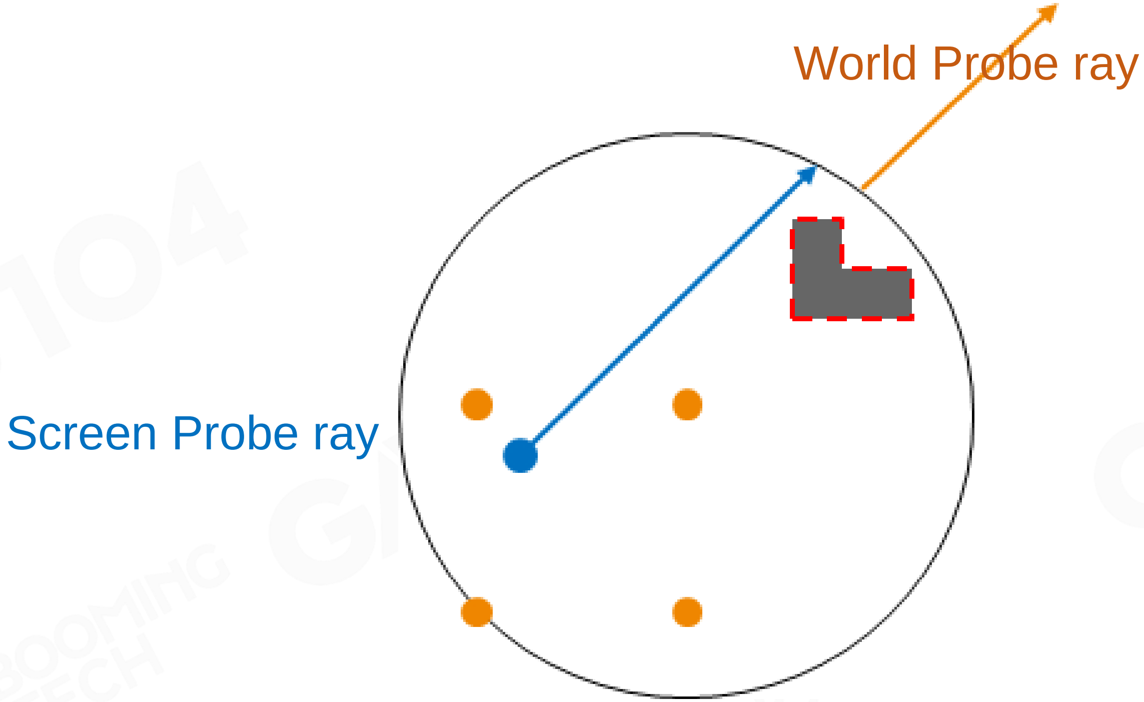
Connecting rays

- Screen Probe ray must cover interpolation footprint + skipped distance



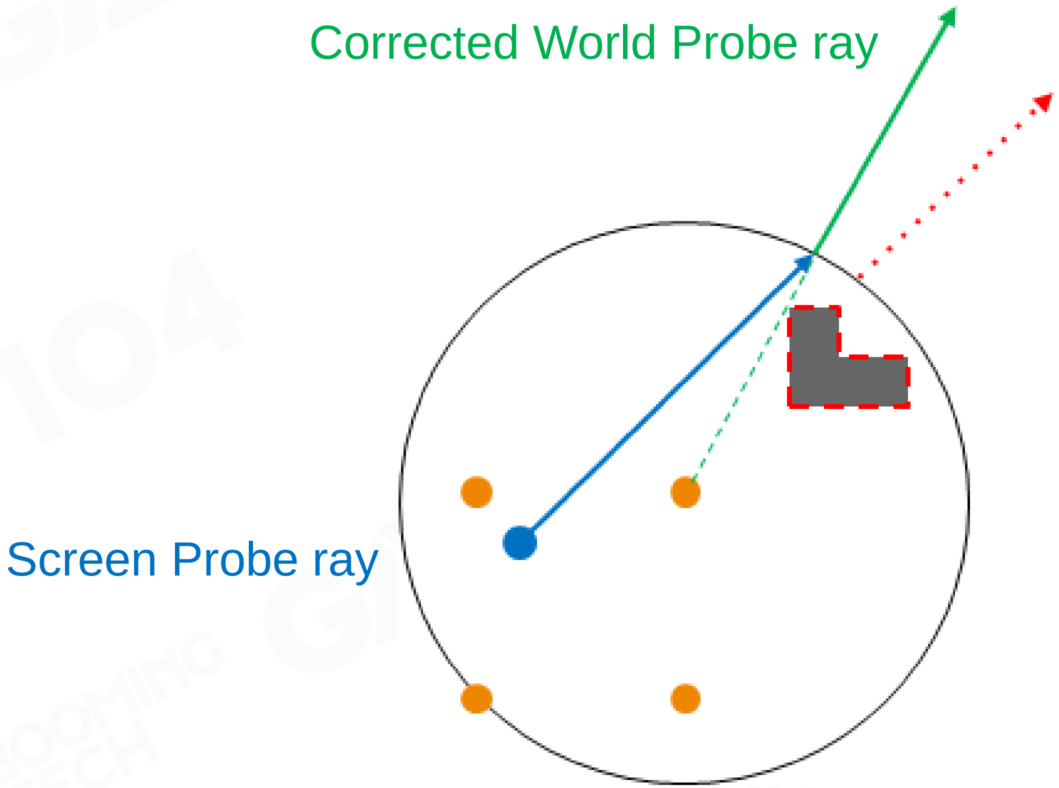
- Problem: leaking!

- World probe radiance should have been occluded
  - But wasn't due to incorrect parallax



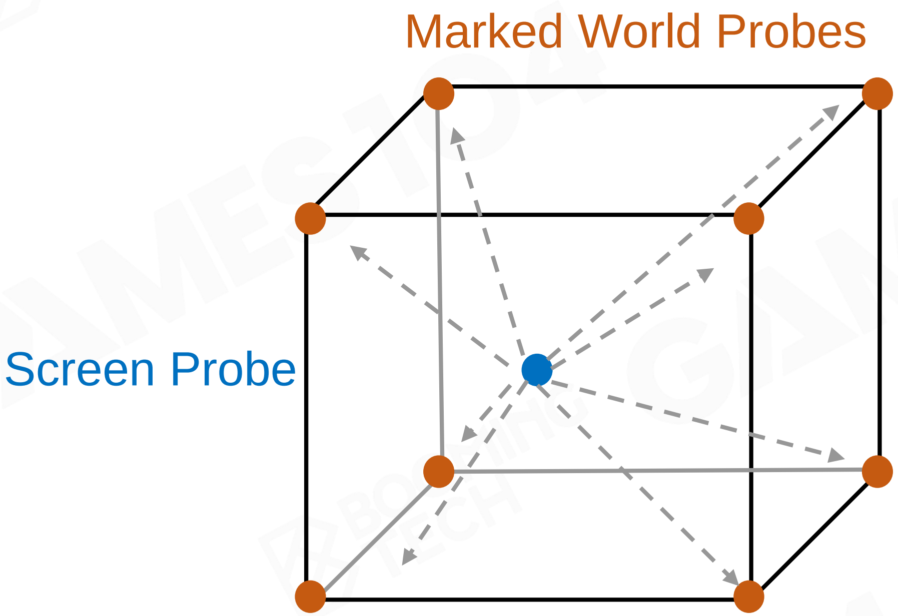
- Solution: simple sphere parallax

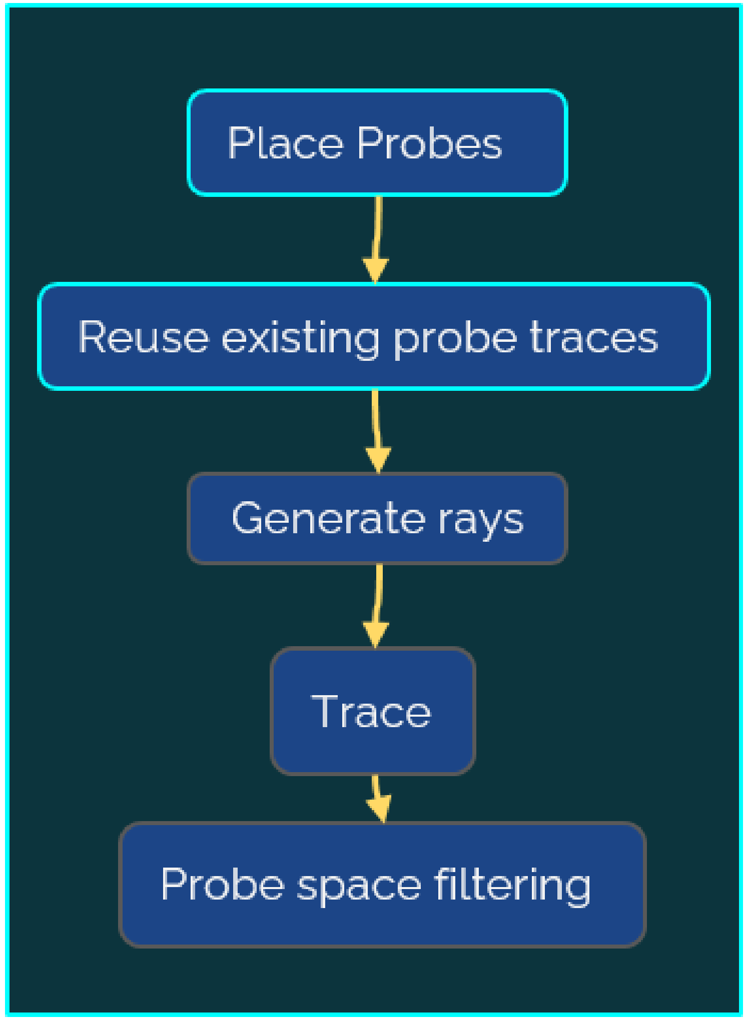
- Reproject Screen Probe ray intersection with World Probe sphere



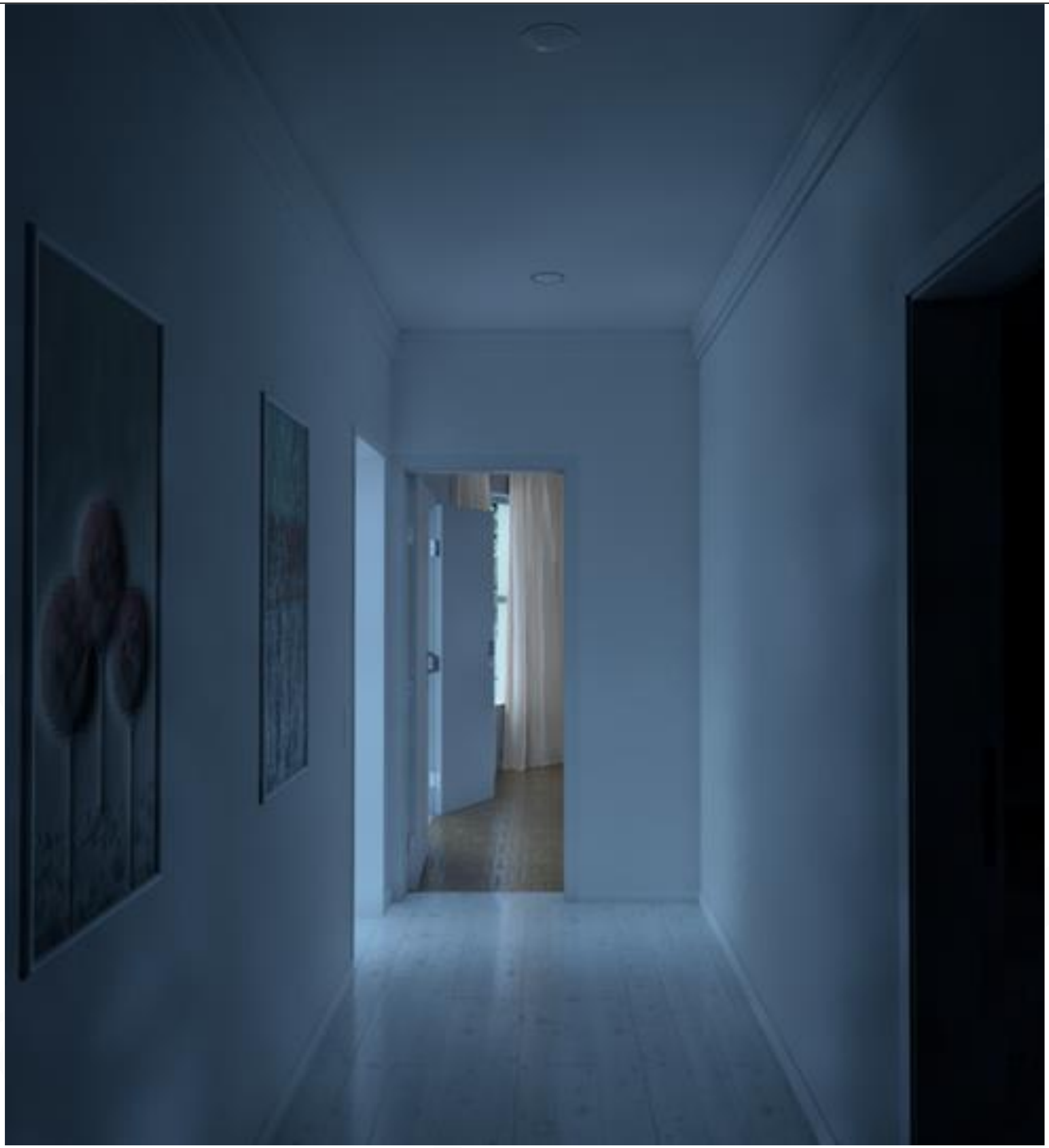
Placement and caching

- Mark any position that we will interpolate from later in clipmap indirections
- For each marked world probe:
  - Reuse traces from last frame, or allocate new probe index
  - Re-trace a subset of cache hits to propagate lighting changes

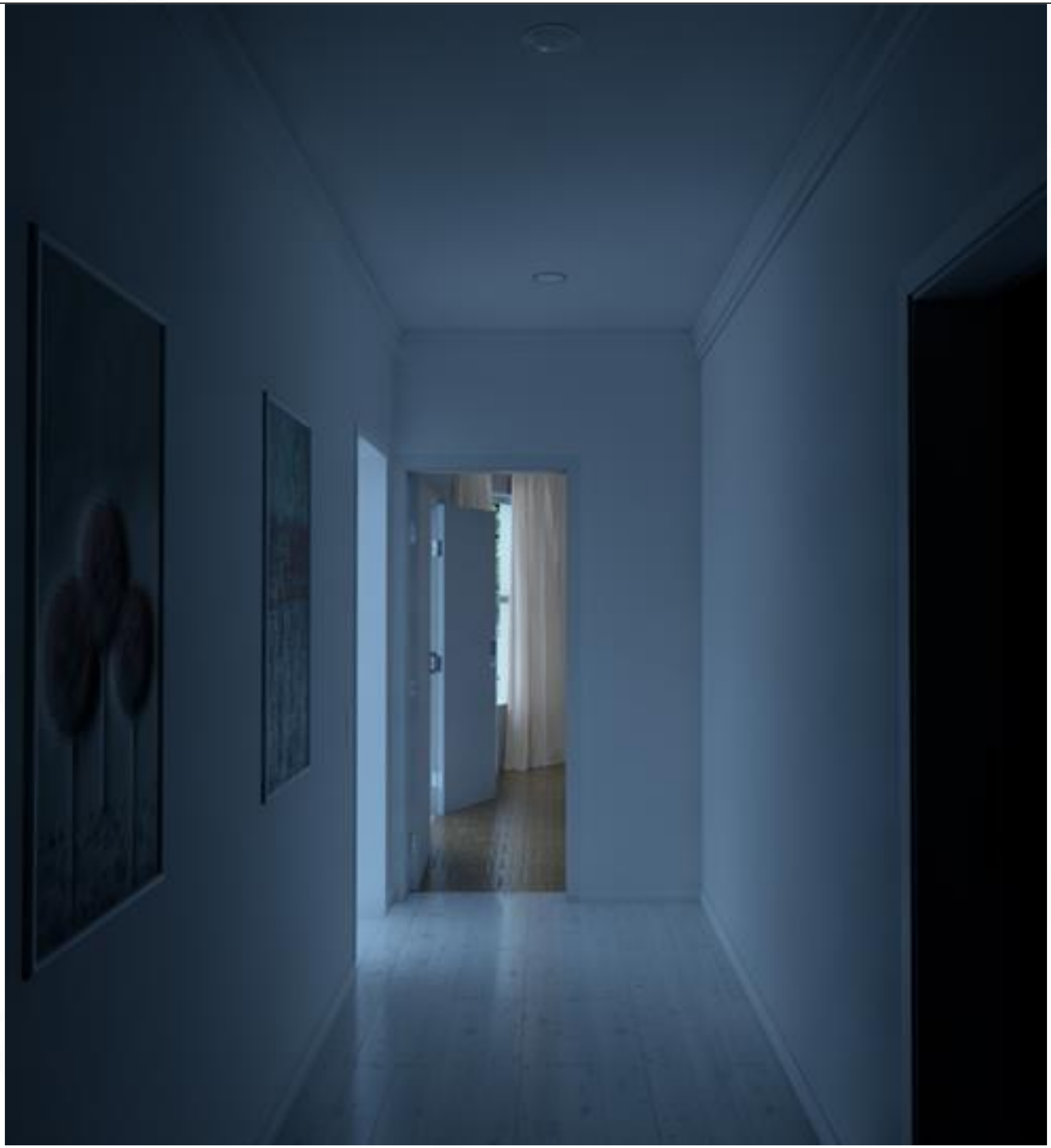




- Without World Space Probes



- Screen Radiance Cache for the first 2 meters
- World Radiance Cache for any lighting further than that.



Phase 4 : Shading Full Pixels with Screen Space Probes

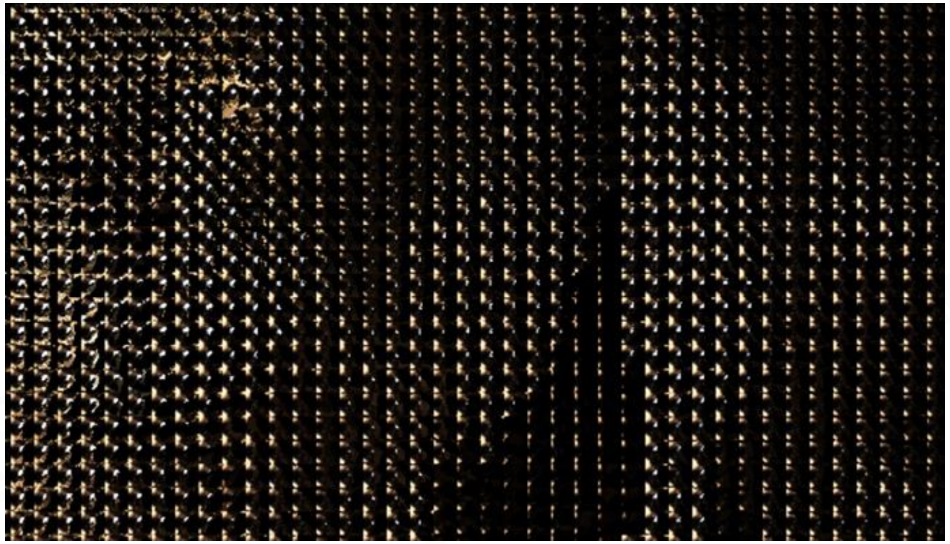
Convert Probe Radiance to 3rd order Spherical Harmonic:

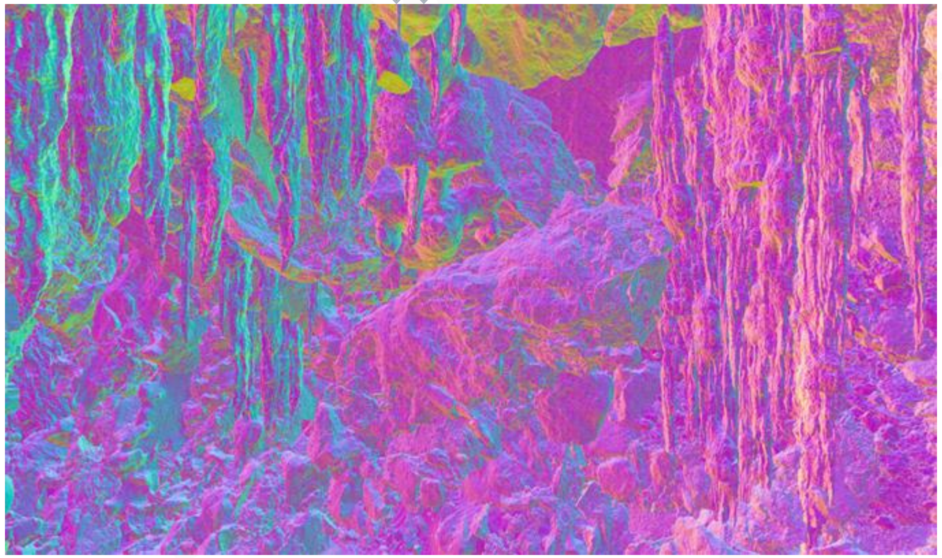
- SH is calculated per Screen Probe
- Full res pixels load SH coherently
- SH Diffuse integration cheap and high quality

importance sample the BRDF to get ray directions, and then sample the Radiance Cache.

Spherical Harmonic

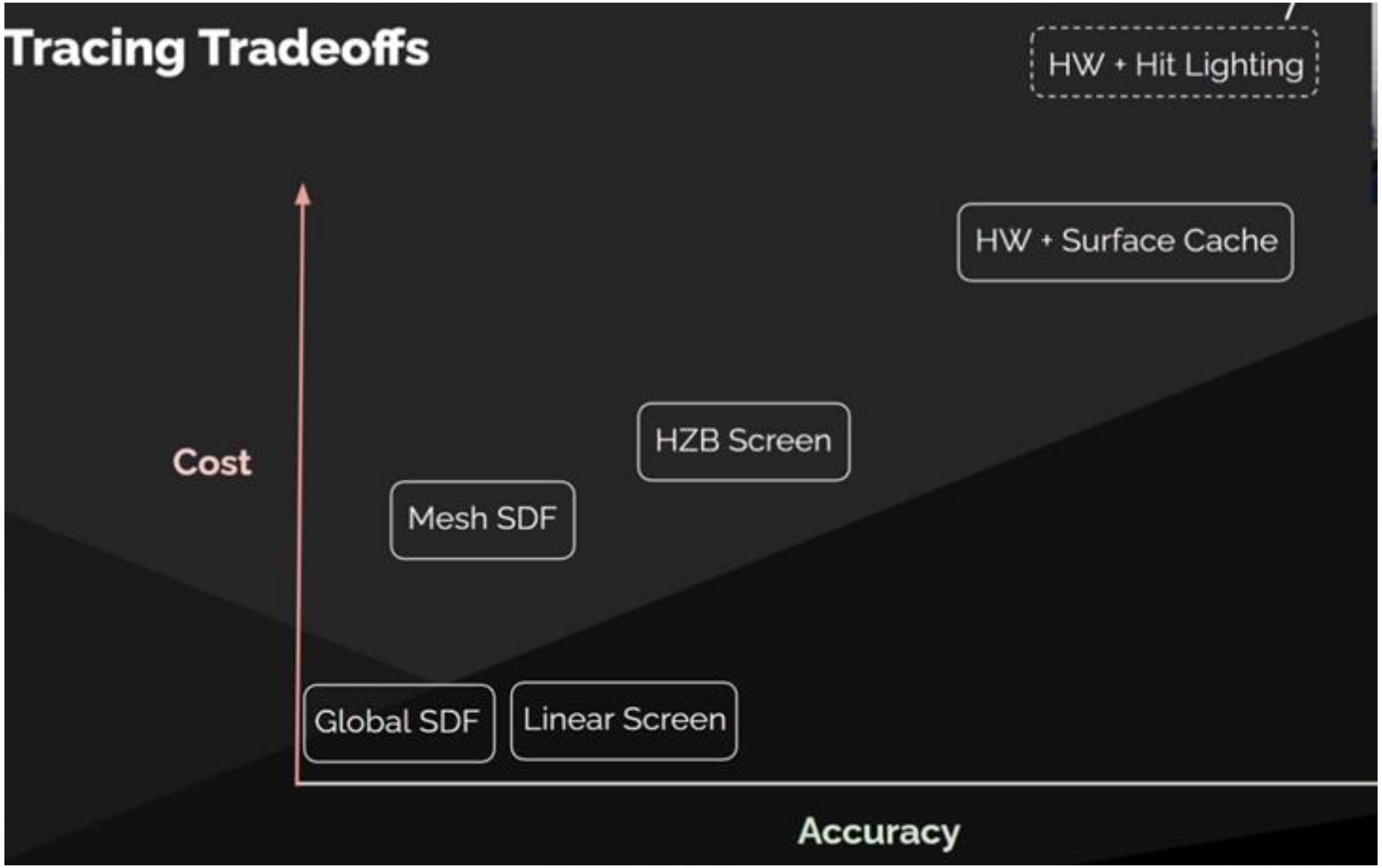
Final integration with SH





Overall, Performance and Result

Speed of Different Tracing Methods



Red – Screen Space Trace

fail to hit

Green – Mesh SDF Trace

fail to hit

Blue – Global SDF Trace

Trace Method	Trace Parameter	Hit Infromation	Sampling
Screen Space Trace	HZB Max Step=50	Hit Screen UV	Previous Frame SceneColorTexture

Fail to hit

Mesh SDF Trace	Max Trace Distance= 1.8m Position in 40m Radius of Camera	Mesh ID Hit World Position Normal	Final Lighting

Fail to hit

Global SDF Trace	Max Trace Distance = 200m	Hit World Position Normal	Voxel Lighting

Fail to hit

Cubemap	Infinite	N/A	Sky Cube Color

SSGI OFF

SSGI ON

Performance

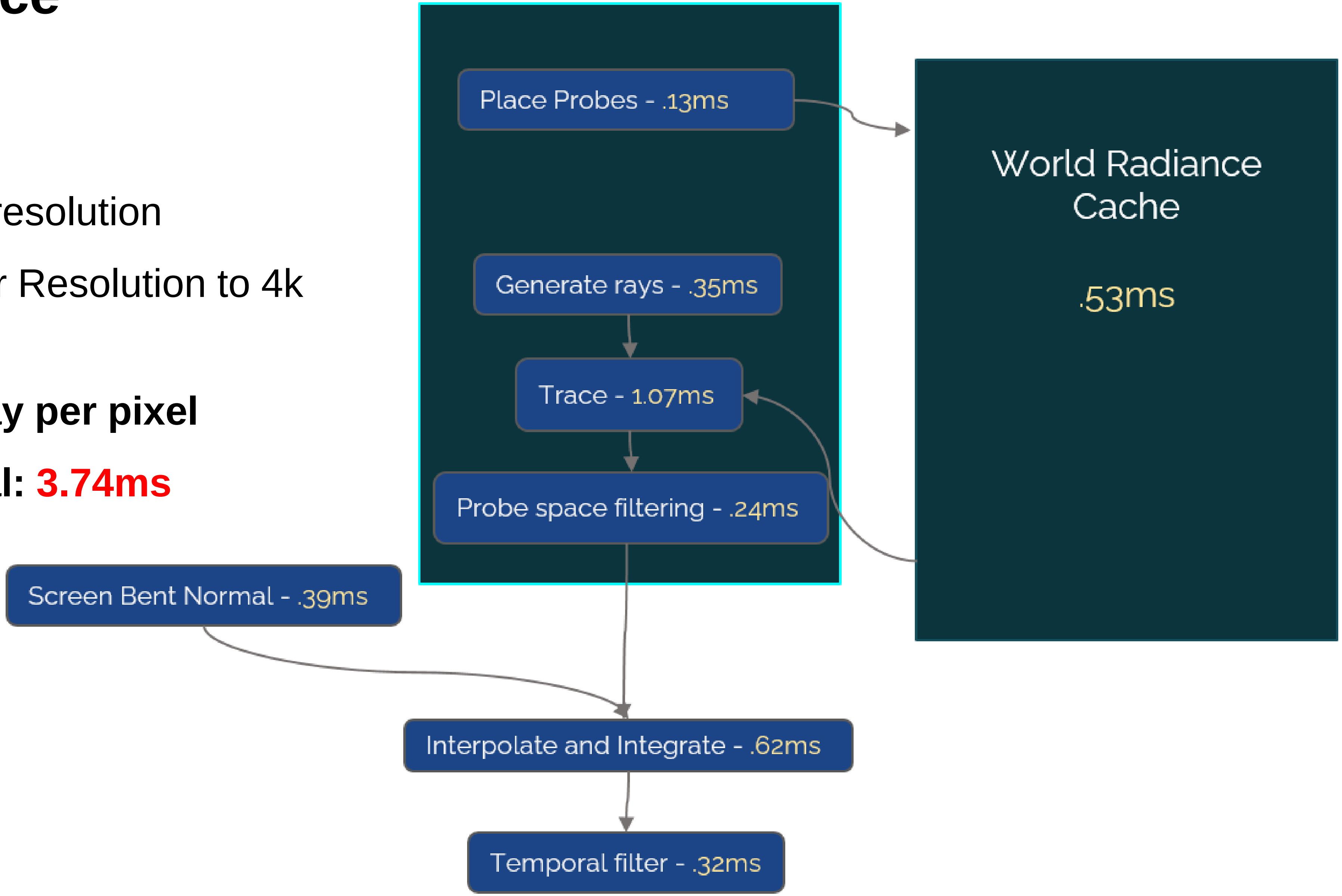
Playstation 5

1080p internal resolution

Temporal Super Resolution to 4k

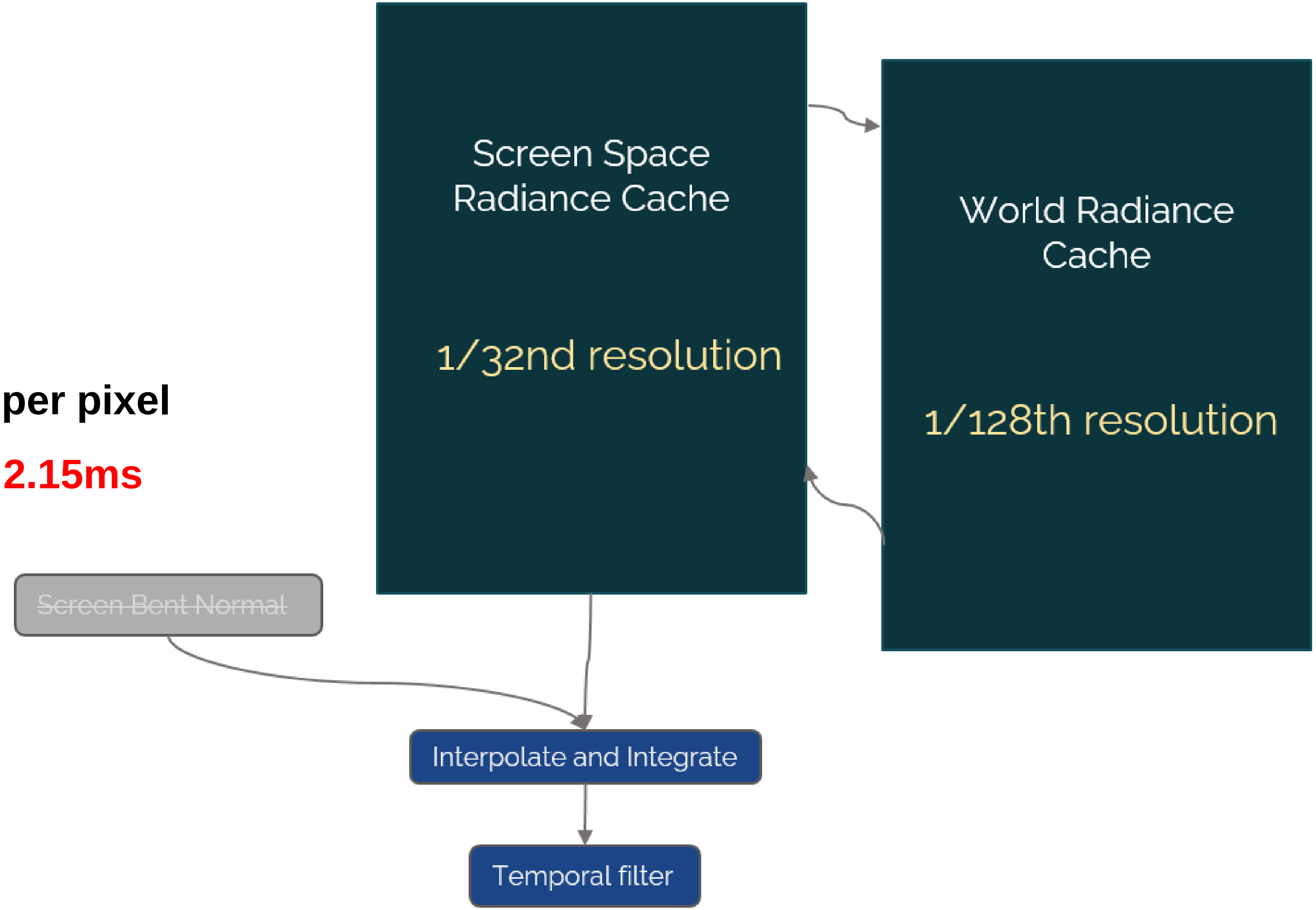
½ ray per pixel

Total: 3.74ms



⅛ ray per pixel

Total: 2.15ms

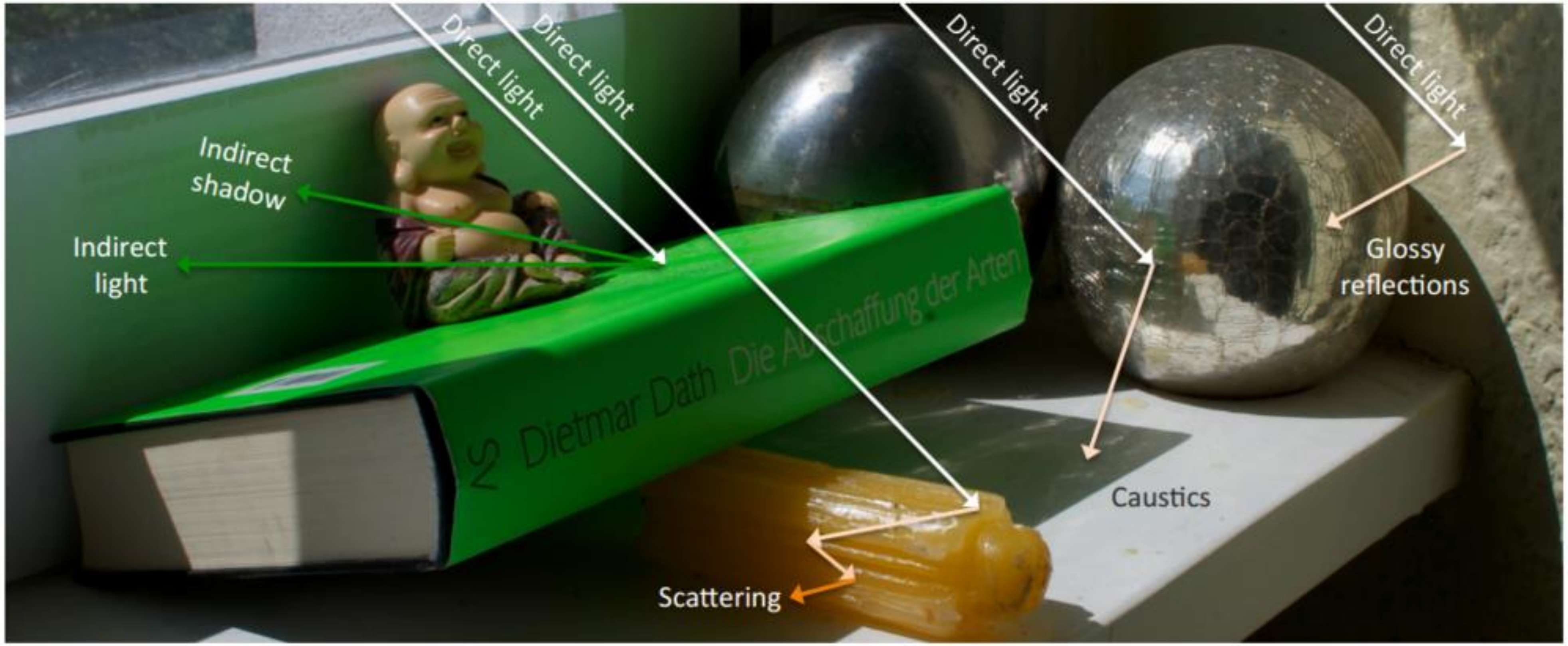



⅛ ray per pixel
2.15ms Final Gather


½ ray per pixel
3.74ms Final Gather

Conclusion

Complexity of Real Rendering



$$L_o(\boldsymbol{x},\omega_o) = L_e(\boldsymbol{x},\omega_o) + \int_{H^2} f_r(\boldsymbol{x},\omega_o,\omega_i) L_i(\boldsymbol{x},\omega_i)\cos\theta_i d\omega_i$$

References

Monte Carlo Integration

- Importace Sampling: https://patapom.com/blog/Math/ImportanceSampling/
- Notes on importance sampling: https://www.tobias-franke.eu/log/2014/03/30/notes_on_importance_sampling.html
- Chinagraph 2020会前课程 3:https://www.bilibili.com/video/BV1my4y1z76s?p=3&vd_source=5e38e5c84aa6ff6cff3b802b3eecb8bc
- Probability density function: https://en.wikipedia.org/wiki/Probability_density_function
- Microfacet Models for Refraction through Rough Surfaces: https://www.cs.cornell.edu/~srm/publications/EGSR07-btdf.pdf

GI

- SSGI: https://www.ea.com/frostbite/news/stochastic-screen-space-reflections
- DDGI: https://zhuanlan.zhihu.com/p/404520592
- DDGI: https://developer.download.nvidia.com/video/gputechconf/gtc/2019/presentation/s9900-irradiance-fields-rtx-diffuse-global-illumination-for-local-and-cloud-graphics.pdf
- DDGI: https://www.jcgt.org/published/0008/02/01/paper-lowres.pdf

GI

- Reflective Shadow Maps: https://users.soe.ucsc.edu/~pang/160/s13/proposal/mijallen/proposal/media/p203-dachsbacher.pdf
- Light Propagation Volumes: https://ericpolman.com/
- VXGI: https://on-demand.gputechconf.com/gtc/2014/presentations/S4552-rt-voxel-based-global-illumination-gpus.pdf
- Lumen: https://advances.realtimerendering.com/s2021/Radiance%20Caching%20for%20real-time%20Global%20Illumination%20(SIGGRAPH%202021).pptx

Hardware Ray Tracing

- Ray Tracing Gems II: https://link.springer.com/content/pdf/10.1007/978-1-4842-7185-8.pdf
- The six levels of ray tracing acceleration: https://f.hubspotusercontent10.net/hubfs/2426966/Gated%20Files/imagination-raytracing-primer-sept2020.pdf
- Hybrid Rendering for Real-Time Ray Tracing: https://link.springer.com/content/pdf/10.1007/978-1-4842-4427-2_25.pdf
- Ray Traced Reflections in 'Wolfenstein: Youngblood': https://www.gdcvault.com/play/1026723/Ray-Traced-Reflections-in-Wolfenstein
- DirectX Raytracing (DXR) Functional Spec: https://microsoft.github.io/DirectX-Specs/d3d/Raytracing.html
- VulkanRayTracingFinalSpecificationRelease

Signed Distance Field

- Dynamic Occlusion With Signed Distance Fields: http://advances.realtimerendering.com/s2015/DynamicOcclusionWithSignedDistanceFields.pdf
- Lectrue5 Real-time Environment Mapping: https://www.bilibili.com/video/BV1YK4y1T7yY?p=5&vd_source=5e38e5c84aa6ff6cff3b802b3eecb8bc
- DX12渲染管线(5) - 距离场 : 建场: https://zhuanlan.zhihu.com/p/89701518?utm_id=0
- GPU Gems 2 - Chapter 8.Per-Pixel Displacement Mapping with Distance Functions: https://developer.nvidia.com/gpugems/gpugems2/part-i-geometric-complexity/chapter-8-pixel-displacement-mapping-distance-functions
- Unreal Engine 5.0 Documentation: https://docs.unrealengine.com/5.0/en-US/mesh-distance-fields-in-unreal-engine/
- Advances in Real-Time Rendering in Games: Part I - SIGGRAPH 2022: https://advances.realtimerendering.com/s2022/index.htmlf

Lecture 20 Contributors

- 砚书
- 坤
- 小老弟
- 峻铖
- 灵鑫
- 光哥

Q&A

Enjoy;)
Coding

Course Wechat

Please follow us for further information