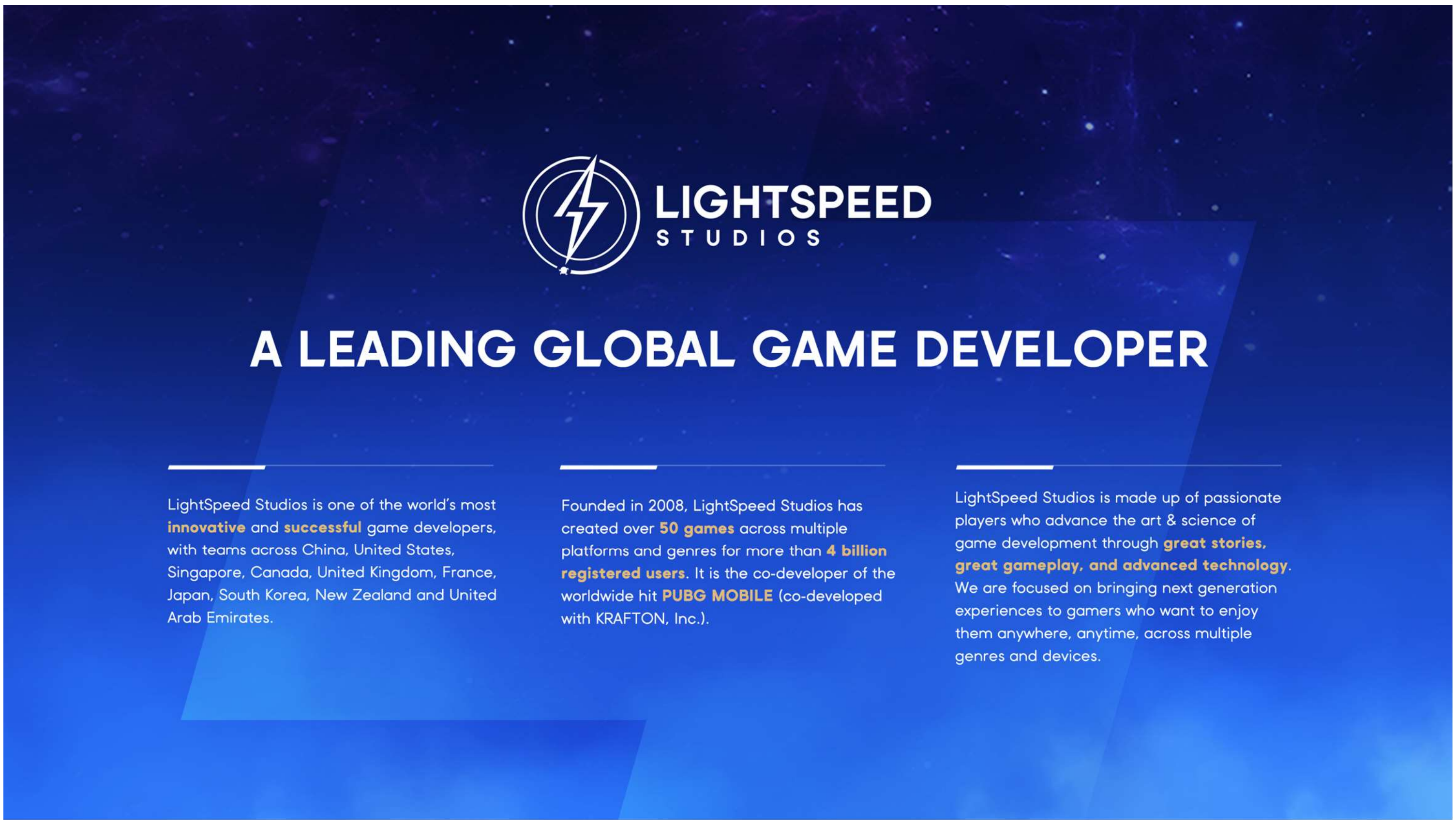
LIGHTSPEED STUDIOS | GDC

Decoding Light: Neural Compression of Global Illumination

Luyan Cao, LIGHTSPEED STUDIOS

My presentation focuses on the application of neural networks in pre-baked global illumination data. I'd like to thank my advisor, Amy Ackermann, for helping me devise a more apt title: "Decoding Light: Neural Compression of Global Illumination."



LIGHTSPEED STUDIOS

A LEADING GLOBAL GAME DEVELOPER

LightSpeed Studios is one of the world's most **innovative** and **successful** game developers, with teams across China, United States, Singapore, Canada, United Kingdom, France, Japan, South Korea, New Zealand and United Arab Emirates.

Founded in 2008, LightSpeed Studios has created over **50 games** across multiple platforms and genres for more than **4 billion registered users**. It is the co-developer of the worldwide hit **PUBG MOBILE** (co-developed with KRAFTON, Inc.).

LightSpeed Studios is made up of passionate players who advance the art & science of game development through **great stories, great gameplay, and advanced technology**. We are focused on bringing next generation experiences to gamers who want to enjoy them anywhere, anytime, across multiple genres and devices.

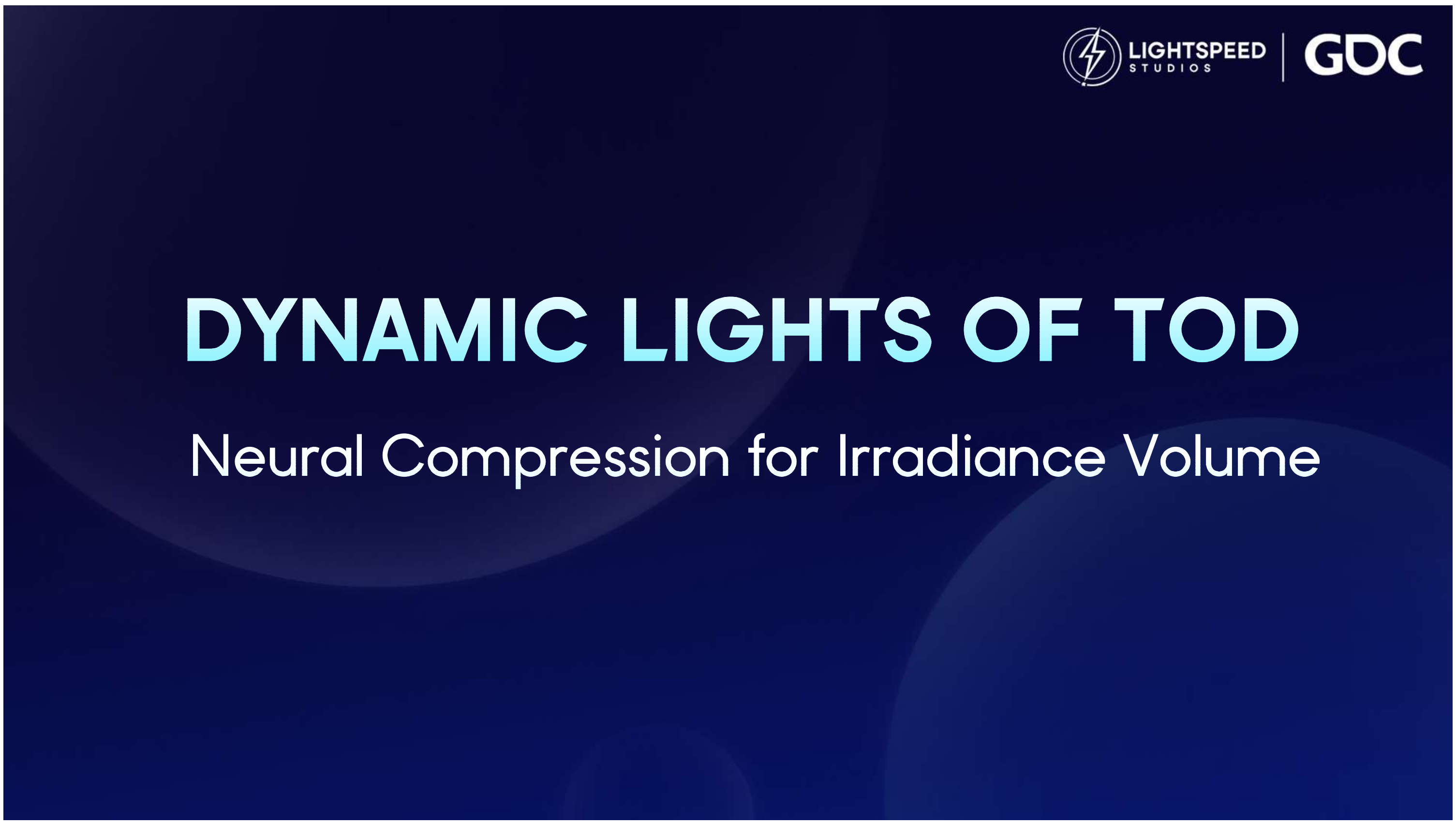
I am from LightSpeed Studios, a global game developer with over fifty games created.

Speaker Bio

- **Luyan Cao**
- LightSpeed Tech Department
- Recently Focus on Global Illumination

my name is Luyan Cao. And I have dedicated over ten years to game development, focusing mainly on rendering. I have also worked at Netease and Bytedance, where I participated in projects such as "Where Winds Meet" and "Earth: Revival." Recently, my main focus has been on global illumination.

As a member of the Lightspeed Tech Department, which is a central technology group, my solution has been applied in several titles. I will share some videos from one of these titles to demonstrate.



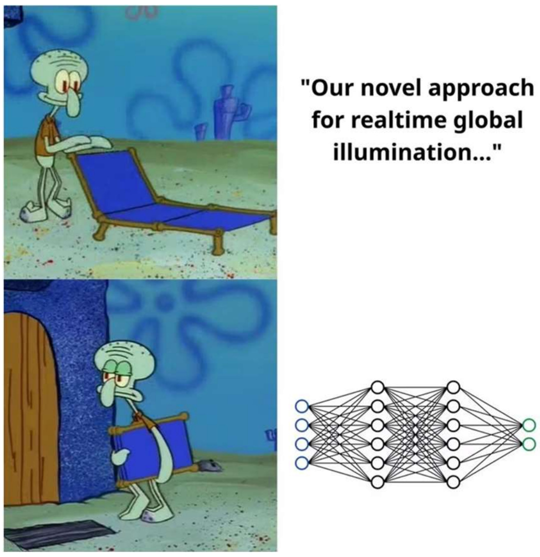
DYNAMIC LIGHTS OF TOD

Neural Compression for Irradiance Volume

The main part of this talk focuses on using neural networks to compress the irradiance volume to achieve the effects of Time of Day Global Illumination.

New method?

- Not Brand New



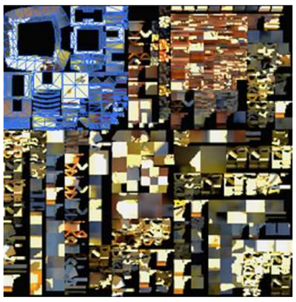
https://research.nvidia.com/publication/2021-06_real-time-neural-radiance-caching-path-tracing
https://www.activision.com/cdn/research/Neural_Light_Grid.pdf

In the last few years, neural networks have become very popular. You might be tired of hearing about their use in seemingly everything. Nevertheless, it's still valuable to explore how these networks can solve tangible problems encountered in game developing. I hope my exploration can provide some inspiration for practical applications.

Why Probe GI?

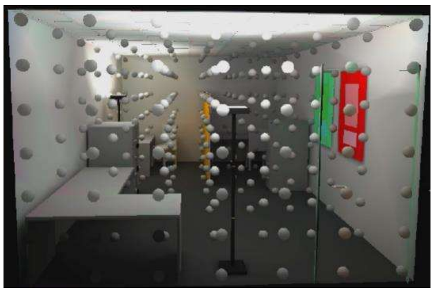
Lightmap

- For Static Objects
- Need 2UV
- Bigger Size/Memory
- Higher Precision



Irradiance Volume

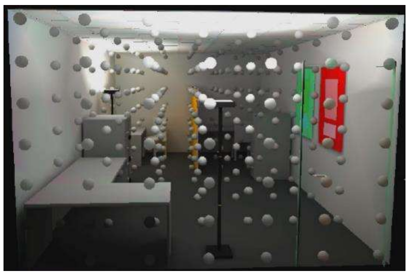
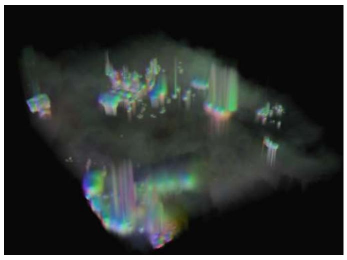
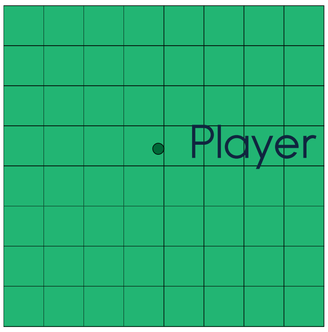
- Both Static and Dynamic
- Simpler Workflow no 2UV
- Smaller Size/Memory
- Light Leaking



For stability and reduced runtime consumption, we employ a pre-baked GI approach. Compared to the more commonly used Lightmap, Irradiance Volume consumes less package space and memory, and can be applied to both static and dynamic models. Considering workflow efficiency, the baking process with probes is also much faster. Additionally, Lightmap requires a separate second UV mapping, whereas the probe-based approach does not.

Relighting

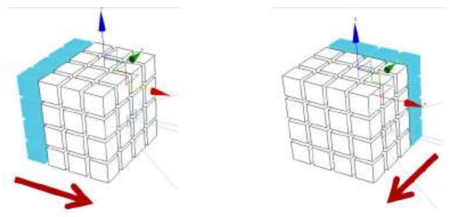
- 3DTex: 256 x 256 x 256 (Different sizes on different platforms)
- 1 probe/2m x 2m x 2m

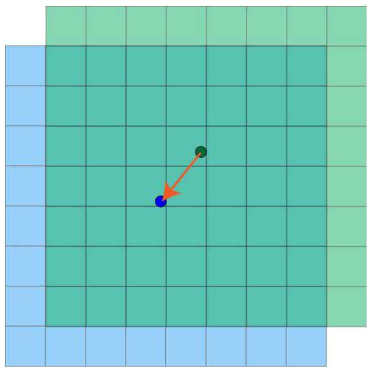




We use a large 3D texture for the final relighting step. The size of this texture varies based on the platform. For example, on PCs, we utilize a 3D texture that measures 256 per side, encompassing an area of 512 meters per side centered around the player. On mobile devices, we switch to a smaller 3D texture.

Streaming

- **Clipmap Scrolling of 64m x 64m sectors**
- Tex: 256 x 256 x 256 (Different sizes on different platforms)
- 1 probe/2m x 2m x 2m



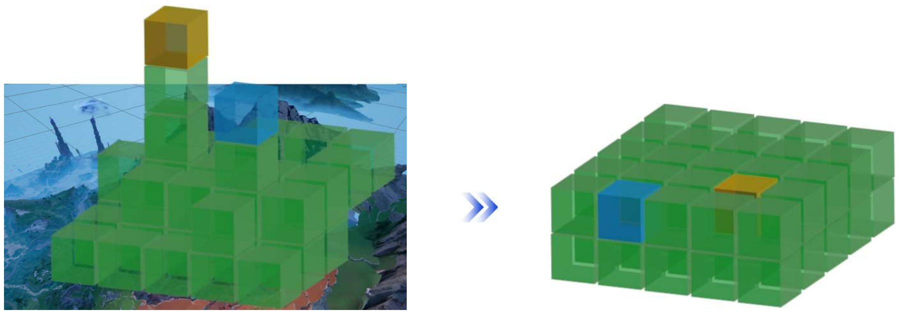


https://www.gdcvault.com/play/1015326/Deferred-Radiance-Transfer-Volumes-Global

For streaming, we utilize a scrolling clipmap technique inspired by the FarCry 3 presentation, where we update only the farthest row of sectors as the player moves.

3D Virtual Texture

- Less Memory, Like Unreal Volumetric Lightmap
- Need Deal with seam issues
- An Extra Sampler for Indexing
- →Back to **Unwrapped 3D Texture**

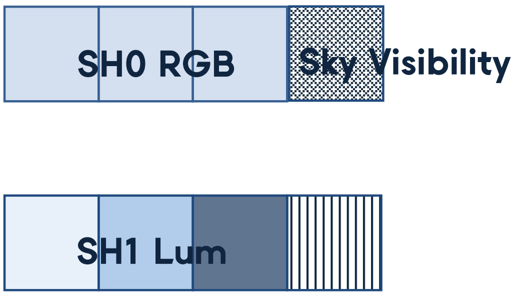


Additionally, we experimented with using a Virtual Texture to manage sparse probe data and reduce memory usage, similar to the volumetric lightmap approach in Unreal Engine. Nevertheless, we prefer using a fully unwrapped 3D texture. The unwrapped texture simplifies the handling of seam issues through trilinear interpolation and removes the need for an extra sampler for indexing.

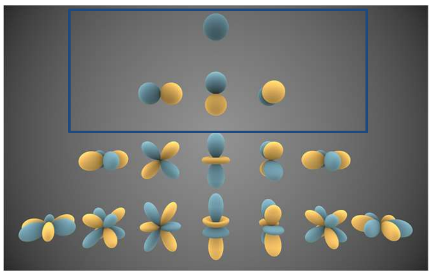
Probe GI Format

LIGHTSPEED STUDIOS | GDC

- **Indirect Diffuse**
  - 2nd Order Spherical Harmonics
  - 6 Bytes
- **Sky Visibility**
  - 1 Byte
- Other Attributes
  - 1 Byte



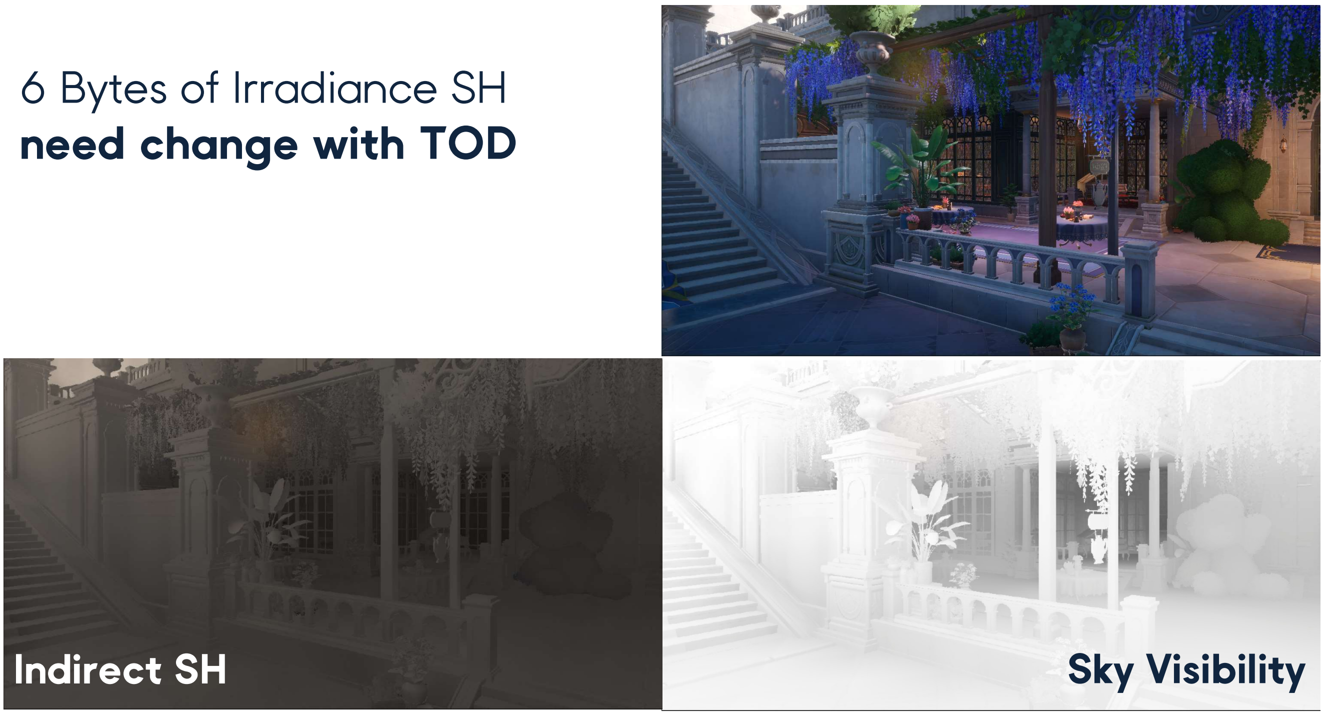
4 Coefficients X 3 RGB
= **12 Floats**



The storage format for probe global illumination data is highly efficient, utilizing just 8 bytes per probe. Typically, color information for second-order spherical harmonics requires 12 floats; Nonetheless, we optimize this by using only 6 bytes for indirect diffuse lighting. We allocate 3 bytes for the RGB values of the zeroth order spherical harmonics. For the first order spherical harmonics' three coefficients, we compress each RGB into a single byte for luminance, totaling three bytes. Thus, 6 bytes are used for storing the irradiance in spherical harmonics. Additionally, one byte is dedicated to sky visibility, allowing for artistic adjustments of ambient light influenced by varying weather conditions. The final byte is used for other attributes of the probe.

Probe GI Format

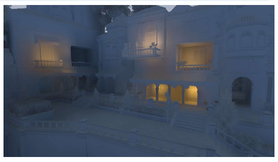
- 6 Bytes of Irradiance SH
  need change with TOD

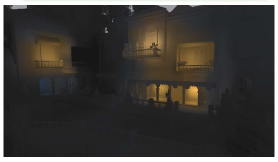


For the scene in the top right image, the component on the bottom left shows the baking result of the indirect light from Spherical Harmonics. The component on the bottom right shows the baking result of SkyVisibility. It transitions more smoothly compared to ambient occlusion .

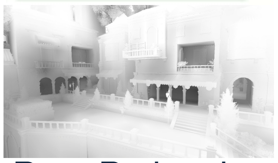
Probe GI

Indirect Diffuse = Irradiance SH + Sky Visibility X Ambient Light





Pre-Baked



Pre-Baked

Sky Color
Equator Color
Ground Color

By Artist

- Not Effective for Time Of Day

  Light Change in direction, intensity, color
- Multiple data sets needed

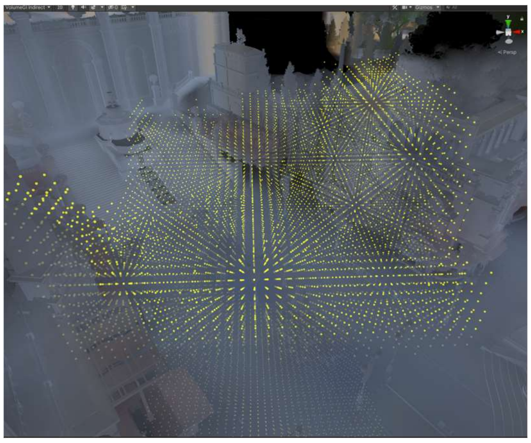
The formula for calculating the final irradiance is: Result = Irradiance SH + Sky Visibility x Ambient Light.

Both the Indirect SH and Sky Visibility components are pre-baked, while the Ambient Light settings can be adjusted by artists. This flexibility allows artists to tailor ambient light to enhance environmental atmospheres and has proven effective in several released titles. However, variations due to different times of day (TOD), changes in light direction, intensity and color can reduce the effectiveness of pre-baked data.

To address this, multiple data sets can be baked to represent the different times of day accurately. On the other hand, the data volume will increase by multiples with the number of baking moments

Time of Day

- Compression of Baked Data
  - Moving Basis Decomposition
  - ZSTD
  - …
  - **Neural Compression**



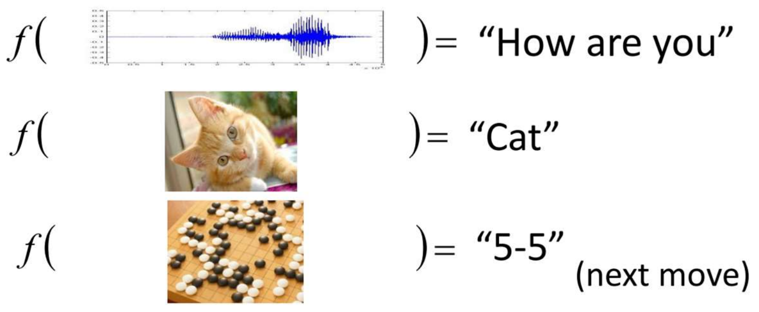
https://facebook.github.io/zstd

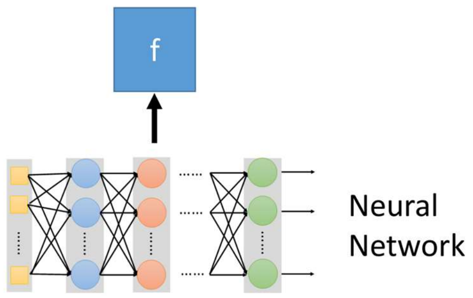
https://research.activision.com/publications/2021/09/moving-basis-decomposition-for-precomputed-light-transport

We initially tried some traditional compression schemes, including both lossy and lossless algorithms, but the compression rates didn't quite meet our requirements.

Compression Function

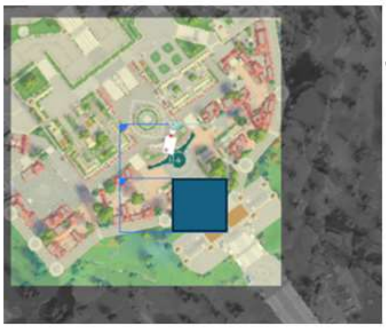
- F (Probe Info, time) = Irradiance

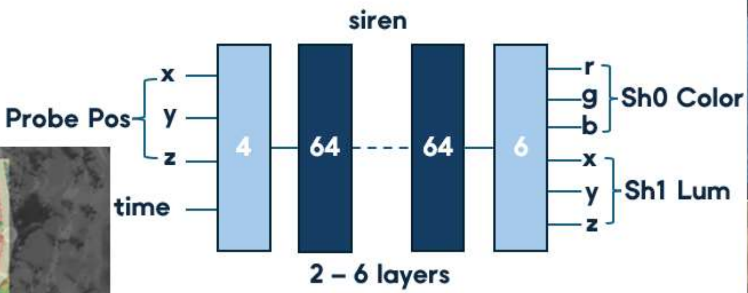




Our requirement can be viewed as a function. Our objective is to identify a function that, given inputs such as probe information, produces outputs like irradiance. It's natural to think of using neural networks as the function.

Multilayer Perceptron(MLP)





Input: position(3), time (1) ,encoded

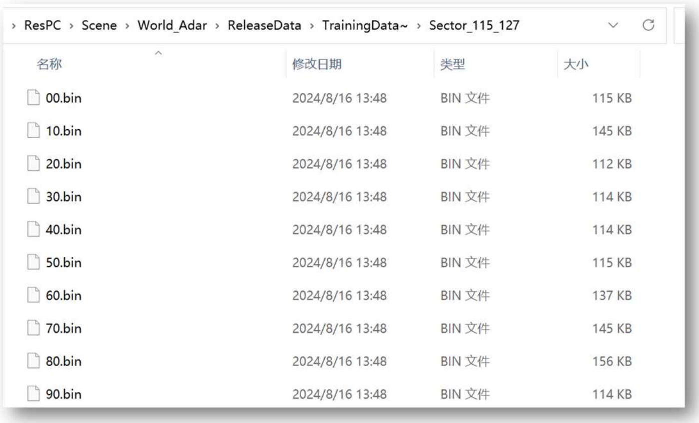
Output: SH0_RGB(3), SH1_Lum(3)

The inputs required for our model consist of four values: the x, y, z coordinates of the Probe Position and a linear time value, all normalized to the range [0, 1]. The output is composed of six bytes, which include the RGB values for the zeroth order of spherical harmonics and three luminance values for the first order of spherical harmonics.

We employ hidden layers with 64 nodes. Depending on the complexity, different numbers of MLP layers may be optimal for various sectors. Furthermore, we have compared the activation functions ReLU and Sin. We found that the Sin-based activation function, known as SIREN, results in smaller errors.

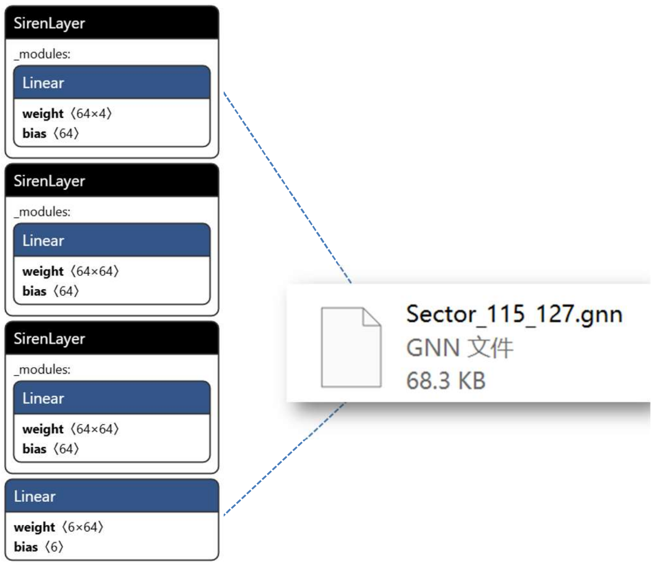
Neural Compression

- Compress per sector(64m x 64m)



› ResPC › Scene › World_Adar › ReleaseData › TrainingData~ › Sector_115_127

名称	修改日期	类型	大小
00.bin	2024/8/16 13:48	BIN 文件	115 KB
10.bin	2024/8/16 13:48	BIN 文件	145 KB
20.bin	2024/8/16 13:48	BIN 文件	112 KB
30.bin	2024/8/16 13:48	BIN 文件	114 KB
40.bin	2024/8/16 13:48	BIN 文件	114 KB
50.bin	2024/8/16 13:48	BIN 文件	115 KB
60.bin	2024/8/16 13:48	BIN 文件	137 KB
70.bin	2024/8/16 13:48	BIN 文件	145 KB
80.bin	2024/8/16 13:48	BIN 文件	156 KB
90.bin	2024/8/16 13:48	BIN 文件	114 KB



10M $\xrightarrow{\text{Run-Length}}$ **1.23MB(1267KB)** $\xrightarrow{\text{MLP}}$ **~70KB**

The compression rate is satisfactory. For example, the 1.23MB of baked data for 10 different times was already run-length encoded. However, after applying neural compression, we were able to reduce it to 70KB.

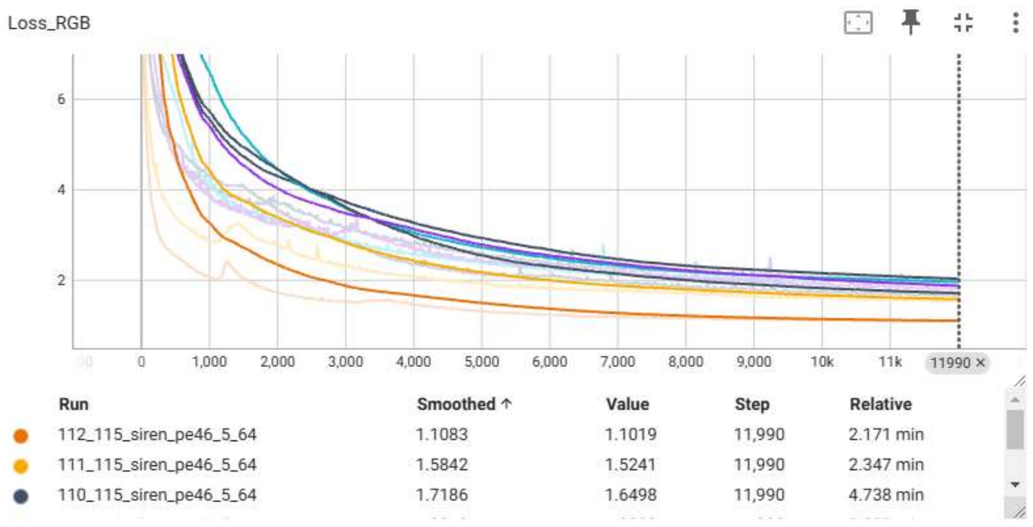
Neural Compression

- MLP Layer number adjusted base on scene complexity
- Below 10M per 1km x 1km
- Little Loss

The layer number of the MLP can be adjusted based on scene complexity of the sector or the number of valid probes. Generally, for every 1km by 1km scene, the storage size needed for the GI data can be kept under 10MB with minimal error.

Neural Compression

- Average RGB loss below 2%

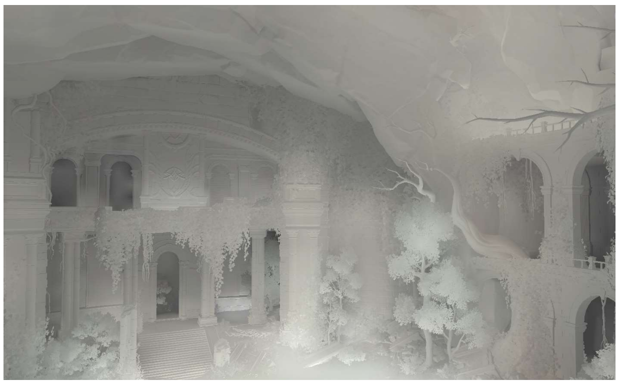


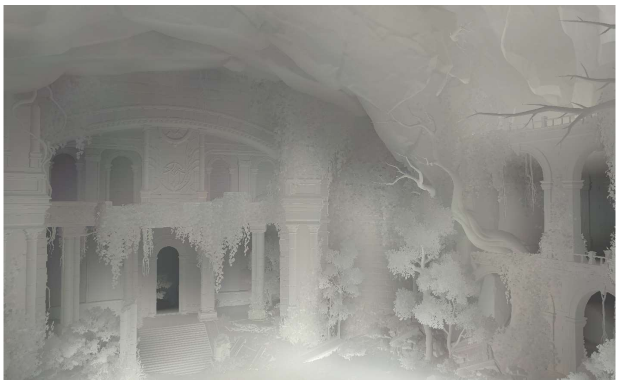
	Run	Smoothed ↑	Value	Step	Relative
●	112_115_siren_pe46_5_64	1.1083	1.1019	11,990	2.171 min
●	111_115_siren_pe46_5_64	1.5842	1.5241	11,990	2.347 min
●	110_115_siren_pe46_5_64	1.7186	1.6498	11,990	4.738 min

This figure shows the gradual decrease of training error as iterations go on. The iteration count was set to about 10,000 cycles to reach a sub-2% RGB color accuracy threshold.

Neural Compression

- Average RGB loss below 2%


Baked SH


Inferred SH

As a result, The left image displays baked lighting results, the right shows the inference outputs. Visually they appear almost identical. Under closer inspection, you will notice subtle color variations in detailed regions – particularly inside the doorway arch

Neural Compression

- Average RGB loss below 2%

  Hard to tell the difference while shaded

Baked

Inferred

When comparing the final composite result, the errors are even harder to spot.

Neural Compression

- MLP Layers adjusted based on scene complexity
- Below 10M per 1km x 1km
- Little Loss
- Problems

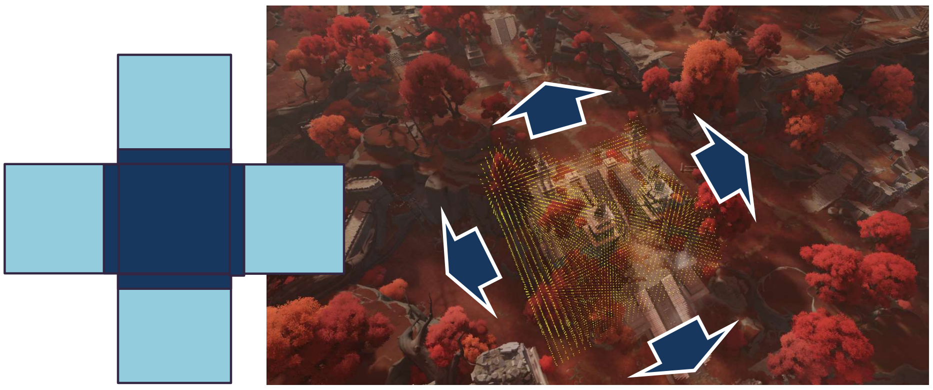
  Seam Issues between sectors

After the compression rates and error levels met our expectations, we encountered some finer details that required attention, such as seam issues between sectors.

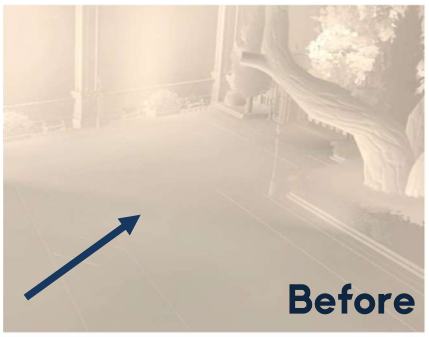
Seam Issues

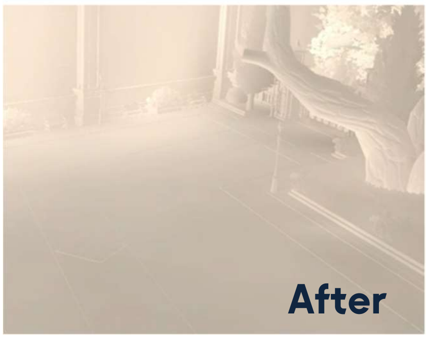
LIGHTSPEED STUDIOS | GDC

- Including **Neighbor Sectors** while training



■ **Training Data**





Because both encoding and decoding are conducted sector by sector, though discrepancies are small, seams between sectors are inevitable. To address this issue early and avoid additional runtime cost, we incorporate data from surrounding sectors during training. We use two extra layers of probes in training to assist. This effectively prevents seam issues. The top image shows the error effect without introducing surrounding probes, and the bottom image shows the effect after solving the error.

Neural Compression

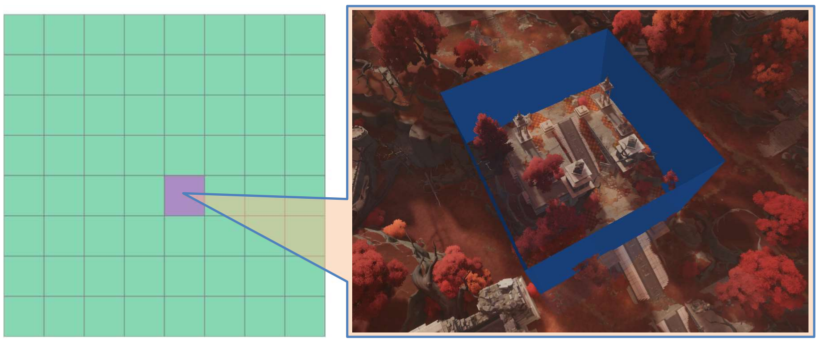
- MLP Layers adjusted based on scene complexity
- Below 10M per 1km x 1km
- Little Loss, no seam issues
- Problems

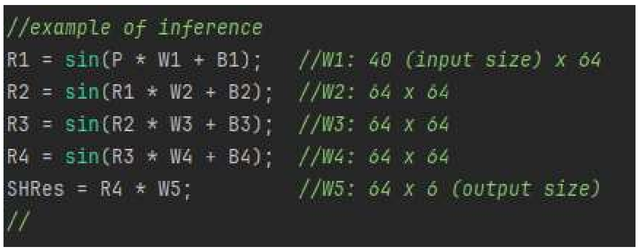
Runtime decode on the device

We've addressed many challenges, such as compression rates and seam issues. Nonetheless, the question I'm most frequently asked is, "How did you manage to run the neural network on the device?" Despite the model being very small, there is still pressure to reduce overhead.

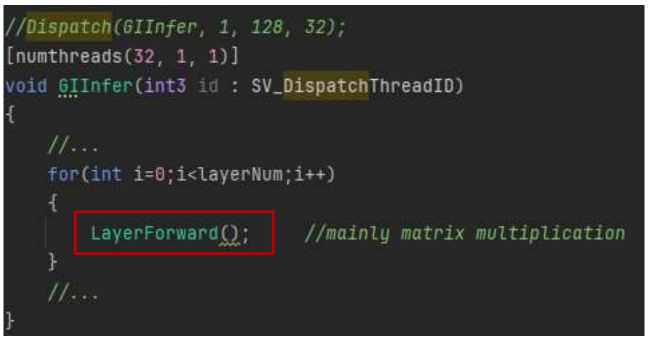
Runtime Decode

- Infer With **Compute Shader**
  Per Sector (64m x 64m)





```
//example of inference
R1 = sin(P * W1 + B1);   //W1: 40 (input size) x 64
R2 = sin(R1 * W2 + B2);  //W2: 64 x 64
R3 = sin(R2 * W3 + B3);  //W3: 64 x 64
R4 = sin(R3 * W4 + B4);  //W4: 64 x 64
SHRes = R4 * W5;         //W5: 64 x 6 (output size)
//
```



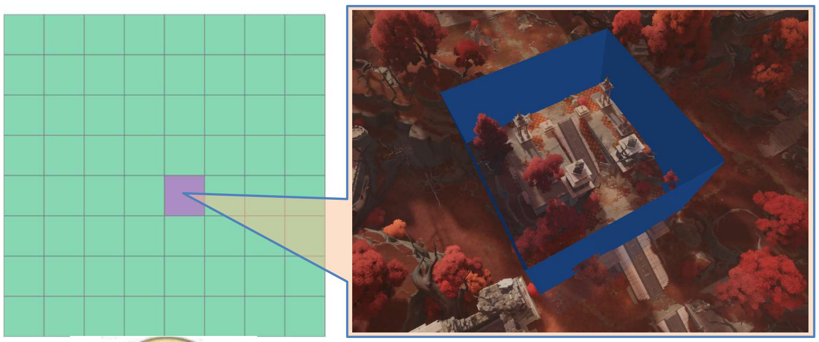
```
//Dispatch(GIInfer, 1, 128, 32);
[numthreads(32, 1, 1)]
void GIInfer(int3 id : SV_DispatchThreadID)
{
    //...
    for(int i=0;i<layerNum;i++)
    {
        LayerForward();     //mainly matrix multiplication
    }
    //...
}
```

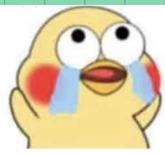
Writing neural network inference with a compute shader is straight forward.   For instance, in our initial setup, we divided threads based on probes. Each probe handled the full inference process, which includes five matrix multiplications.

Runtime Decode

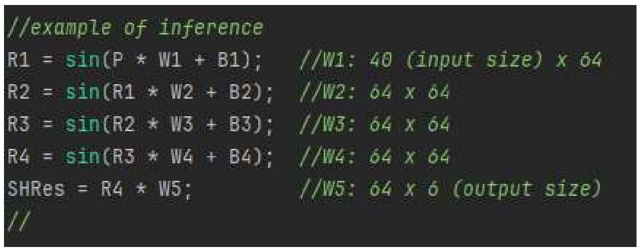
- Infer With Compute Shader

  10ms per sector(64m x 64m)

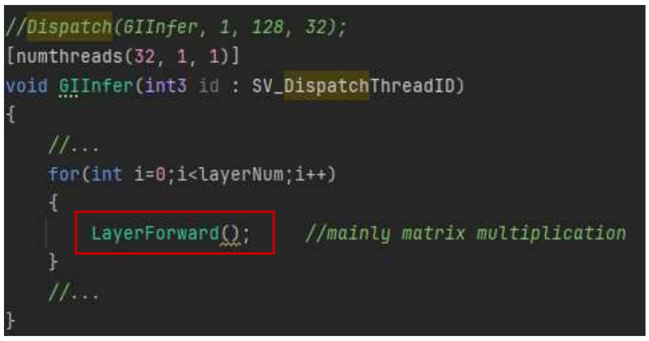




RTX3090 Hits 80°C!



```
//example of inference
R1 = sin(P * W1 + B1);   //W1: 40 (input size) x 64
R2 = sin(R1 * W2 + B2);  //W2: 64 x 64
R3 = sin(R2 * W3 + B3);  //W3: 64 x 64
R4 = sin(R3 * W4 + B4);  //W4: 64 x 64
SHRes = R4 * W5;         //W5: 64 x 6 (output size)
//
```



```
//Dispatch(GIInfer, 1, 128, 32);
[numthreads(32, 1, 1)]
void GIInfer(int3 id : SV_DispatchThreadID)
{
    //...
    for(int i=0;i<layerNum;i++)
    {
        LayerForward();     //mainly matrix multiplication
    }
    //...
}
```

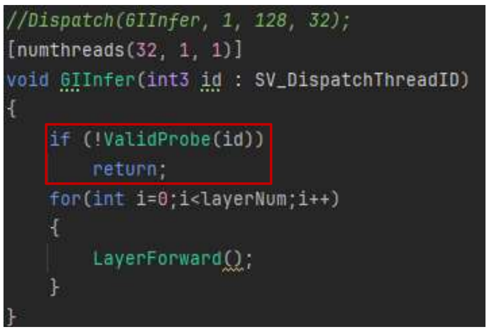
After setting up the compute shader this way, the inference time for each sector hits 10ms on RTX 3090. As shown on the earlier streaming page, we need to process 64 sectors to complete the 3D texture, which is required for the near distance irradiance. Consequently, the system really heated up, reaching 80°C. We really need to find ways to keep things cool.

Runtime Decode

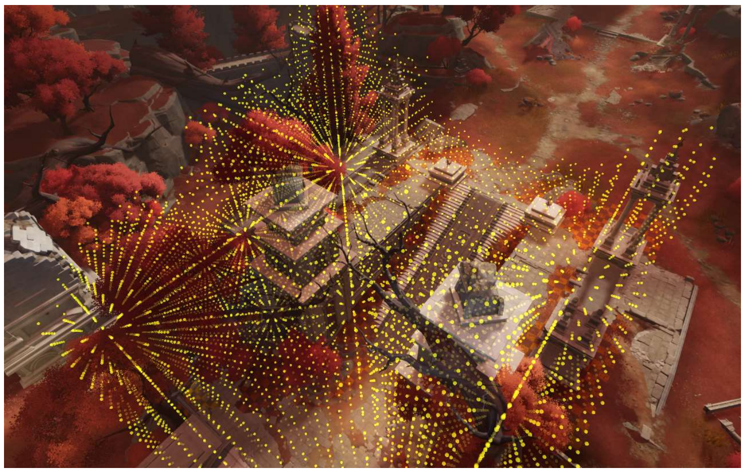
- Optimization

With Valid Mask

3ms per sector



```
//Dispatch(GIInfer, 1, 128, 32);
[numthreads(32, 1, 1)]
void GIInfer(int3 id : SV_DispatchThreadID)
{
    if (!ValidProbe(id))
        return;
    for(int i=0;i<layerNum;i++)
    {
        LayerForward();
    }
}
```



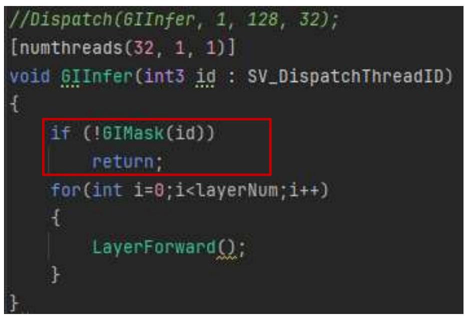
First, by examining how the probes are positioned. We quickly realized that not every probe at every position needs inference. We only need to focus on the valid probes near the surface of the objects. By introducing a valid mask to filter things out, we managed to reduce the inference time per sector down to 3ms, but still too much.

Runtime Decode

- Optimization

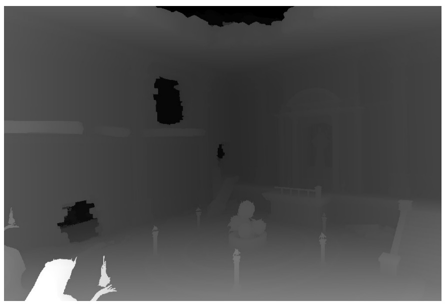
Only Consider Probes near Screen Space Depth

1ms per sector



```
//Dispatch(GIInfer, 1, 128, 32);
[numthreads(32, 1, 1)]
void GIInfer(int3 id : SV_DispatchThreadID)
{
    if (!GIMask(id))
        return;
    for(int i=0;i<layerNum;i++)
    {
        LayerForward();
    }
}
```

Valid Mask



Screen Depth

Then, we use the screen space G-buffer depth. Now we only consider probes near the screen depth. Probes that are off screen or occluded do not need to be processed in the current frame. Initially, we record the probes to be inferenced with priority onto another 3D texture. Then choose the probes to deal with in the current frame. They are indicated by the red pixels in the middle texture. As you can see, the number of probes requiring processing is minimal. We can deal with other probes when the GPU is idle. This sorting and selecting probe approach reduces the inference time to just 1ms per frame.

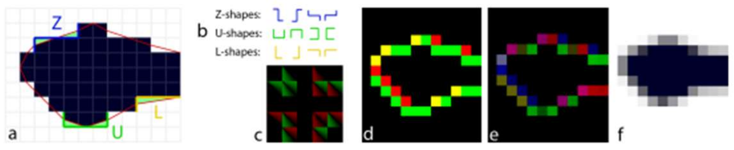
Runtime Decode

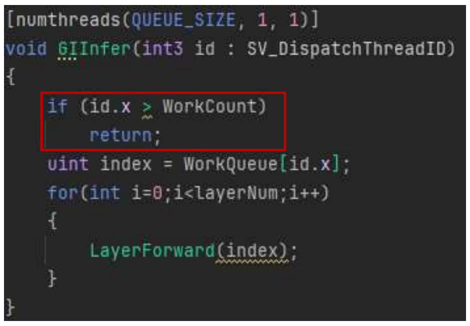
- Optimization

  Only Consider Probes near Screen Space Depth

  Organize Objects for GPU Latency Hiding

  0.6-0.7ms





```
[numthreads(QUEUE_SIZE, 1, 1)]
void GIInfer(int3 id : SV_DispatchThreadID)
{
    if (id.x > WorkCount)
        return;
    uint index = WorkQueue[id.x];
    for(int i=0;i<layerNum;i++)
    {
        LayerForward(index);
    }
}
```

https://www.iryoku.com/smaa/downloads/SMAA-Enhanced-Subpixel-Morphological-Antialiasing.pdf

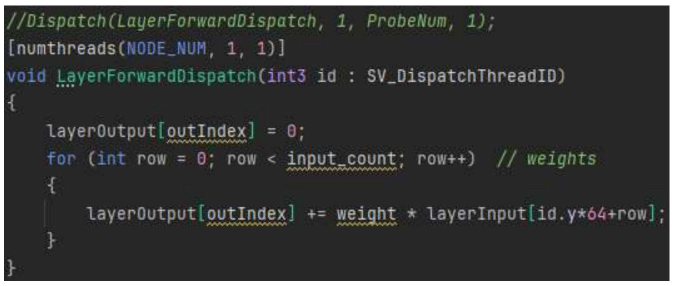
Experienced engineers may have noticed that dynamic judgment can damage GPU latency hiding. Inspired by the SMAA method, we reorganize the necessary pixels into a queue. This approach further compresses the time to 0.7ms per sector

Runtime Decode

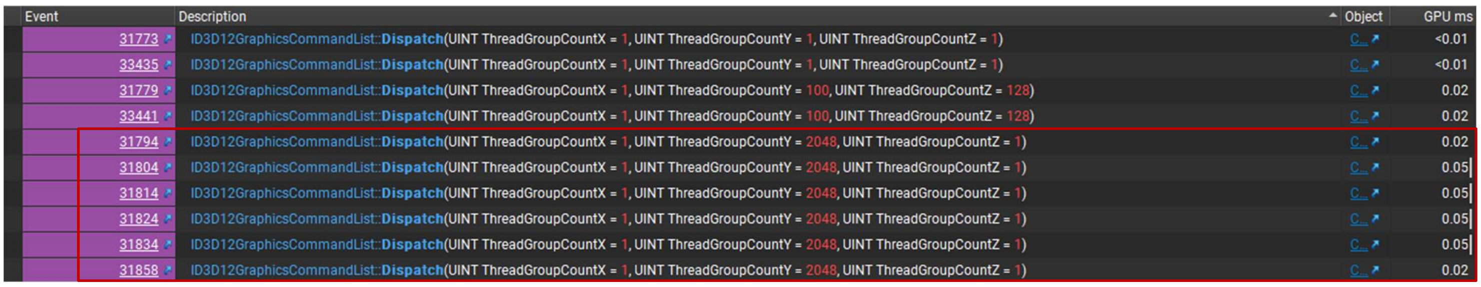
- Optimization

Dispatch Each Layer Forward

0.3ms in total



```
//Dispatch(LayerForwardDispatch, 1, ProbeNum, 1);
[numthreads(NODE_NUM, 1, 1)]
void LayerForwardDispatch(int3 id : SV_DispatchThreadID)
{
    layerOutput[outIndex] = 0;
    for (int row = 0; row < input_count; row++)  // weights
    {
        layerOutput[outIndex] += weight * layerInput[id.y*64+row];
    }
}
```

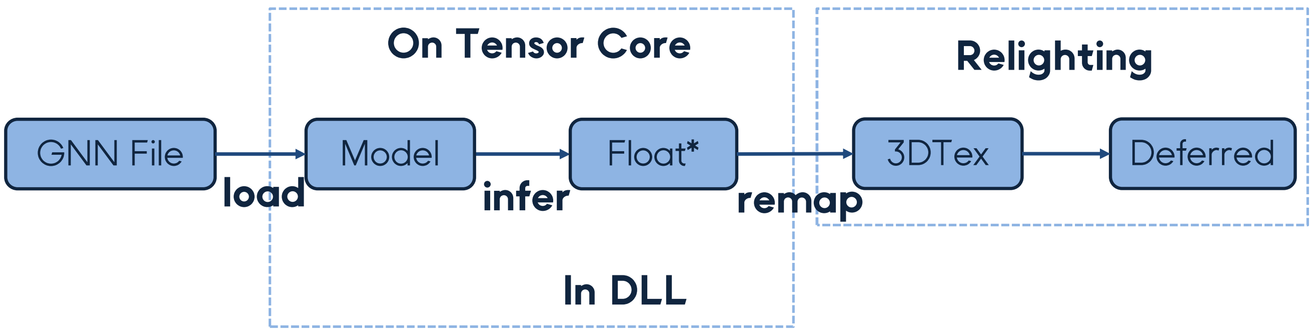


Event	Description	Object	GPU ms
31773	ID3D12GraphicsCommandList::**Dispatch**(UINT ThreadGroupCountX = 1, UINT ThreadGroupCountY = 1, UINT ThreadGroupCountZ = 1)	C...	<0.01
33435	ID3D12GraphicsCommandList::**Dispatch**(UINT ThreadGroupCountX = 1, UINT ThreadGroupCountY = 1, UINT ThreadGroupCountZ = 1)	C...	<0.01
31779	ID3D12GraphicsCommandList::**Dispatch**(UINT ThreadGroupCountX = 1, UINT ThreadGroupCountY = 100, UINT ThreadGroupCountZ = 128)	C...	0.02
33441	ID3D12GraphicsCommandList::**Dispatch**(UINT ThreadGroupCountX = 1, UINT ThreadGroupCountY = 100, UINT ThreadGroupCountZ = 128)	C...	0.02
31794	ID3D12GraphicsCommandList::**Dispatch**(UINT ThreadGroupCountX = 1, UINT ThreadGroupCountY = 2048, UINT ThreadGroupCountZ = 1)	C...	0.02
31804	ID3D12GraphicsCommandList::**Dispatch**(UINT ThreadGroupCountX = 1, UINT ThreadGroupCountY = 2048, UINT ThreadGroupCountZ = 1)	C...	0.05
31814	ID3D12GraphicsCommandList::**Dispatch**(UINT ThreadGroupCountX = 1, UINT ThreadGroupCountY = 2048, UINT ThreadGroupCountZ = 1)	C...	0.05
31824	ID3D12GraphicsCommandList::**Dispatch**(UINT ThreadGroupCountX = 1, UINT ThreadGroupCountY = 2048, UINT ThreadGroupCountZ = 1)	C...	0.05
31834	ID3D12GraphicsCommandList::**Dispatch**(UINT ThreadGroupCountX = 1, UINT ThreadGroupCountY = 2048, UINT ThreadGroupCountZ = 1)	C...	0.05
31858	ID3D12GraphicsCommandList::**Dispatch**(UINT ThreadGroupCountX = 1, UINT ThreadGroupCountY = 2048, UINT ThreadGroupCountZ = 1)	C...	0.02

Now let's deal with thread management. As you know, most of neural network inference includes heavy calculation, especially matrix multiplication. Initially, we assigned one thread per probe. However, we realized we weren't fully using the parallel processing ability of matrix multiplication. So we shifted to allocate one thread per element in the matrix and assign one dispatch for each matrix multiplication. Although this means more dispatches, our total processing time drops down to just 0.3ms.

Runtime Decode

- With Tensor Core
- <1ms for 1 sector, Asynchronous Thread
- Update Whole 3D Texture in 2 frames



https://github.com/davidAlgis/InteropUnityCUDA

On machines equipped with computational acceleration units like NVIDIA Tensor Cores, we can enhance inference performance by utilizing CUDA for parallel computing. All loading processes are managed through DLLs, and after inference, the results are remapped into the engine.

This allows for the simultaneous processing of all probes within a sector, with each sector taking less than 1ms. As a result, the frequency of inference and memory mapping is reduced, ensuring optimal performance on machines with Tensor Cores.

Moreover, the entire process can be executed asynchronously. Whenever there is a change in weather, we can obtain updated results for the entire 3D texture within two frames.

Runtime Decode

- With Compute Shader

Probe Num	Platform	Infer Time
512	Snapdragon 888	1.3ms
1024	Dimensity 9200	1.2ms
2048	GTX 1060	<0.3ms
8192	RTX 3090	<0.3ms

- With Tensor Core

Probe Num	Platform	Infer Time
197632	RTX 4070	0.9ms

On mobile devices, we will reduce the size of 3D textures and the number of probes per frame for inference. Based on performance of different mobile GPUs, we will keep the inference time per frame under 1.5 milliseconds for all devices. With the help of Tensor cores or Neural Processing Units, we can handle more probes.

Result

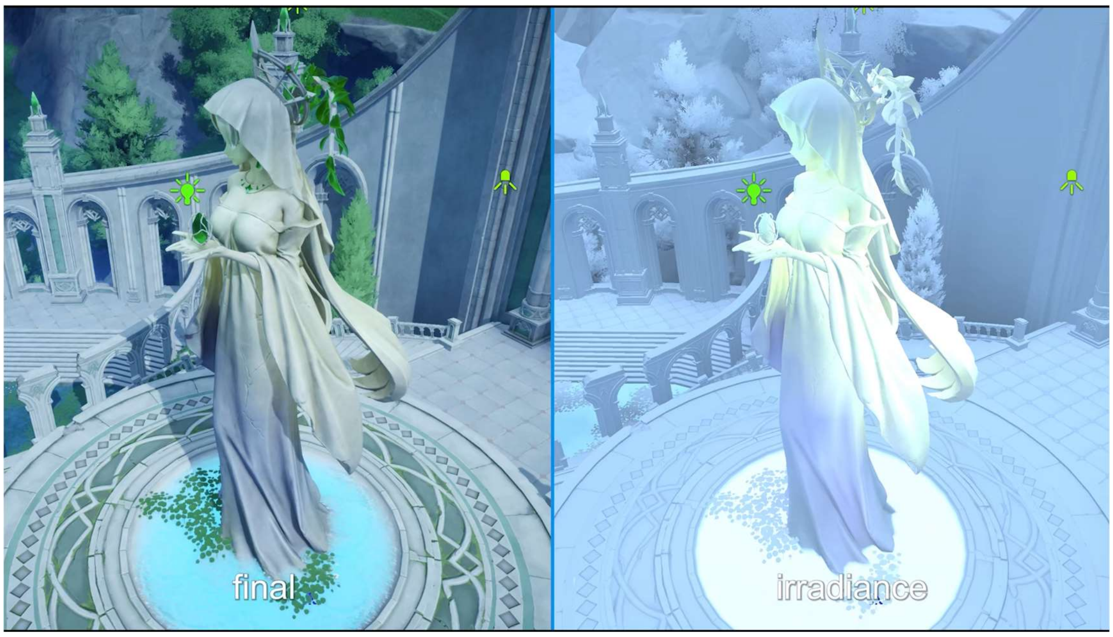
- Irradiance SH Changed by TOD with no real-time light

Let me show some results. In this example, all indirect lighting and some direct lighting changes are handled by NeuralGI. Except for sunlight, there's no real-time lighting calculation here.

Result

LIGHTSPEED STUDIOS | GDC

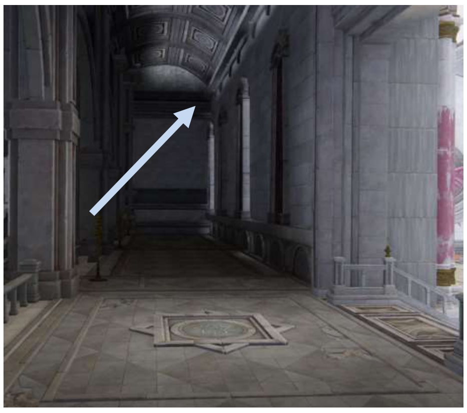
- Nighttime Illuminated Goddess with no real-time light

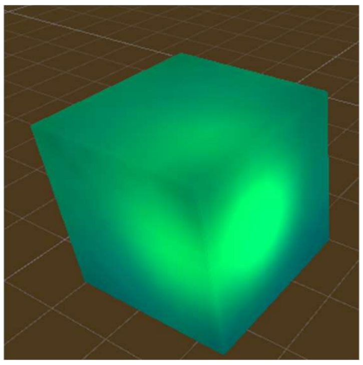


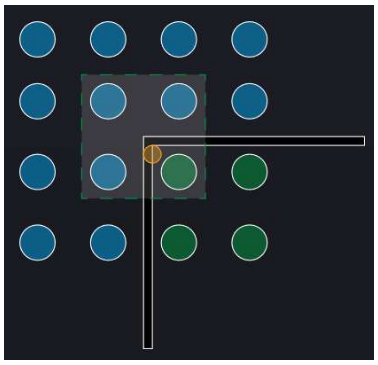
Here’s another example. The lighting changes on the goddess are also obtained from prebaked lighting, trained and inferred, with no need for real-time lighting calculations per pixel.

Light Leaking

- During Trilinear Sampling





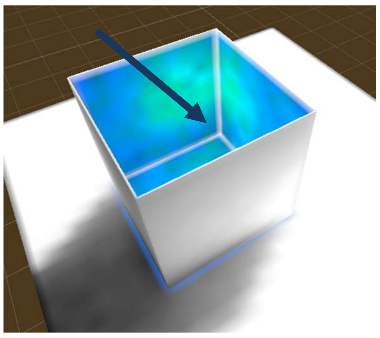


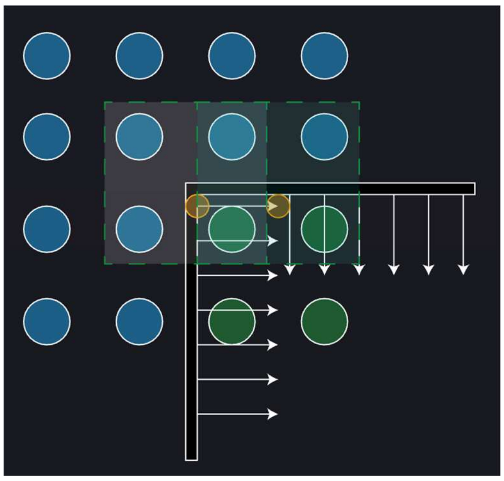
Despite the effectiveness of prebaked irradiance volumes, many probe GI solutions are plagued by light leakage issues during sampling. This is primarily because hardware Trilinear sampling introduces surrounding probes.

Light Leaking

- **Normal Offset**

  Not work at corners

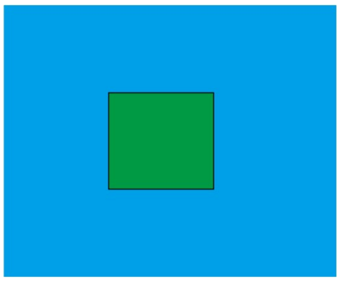


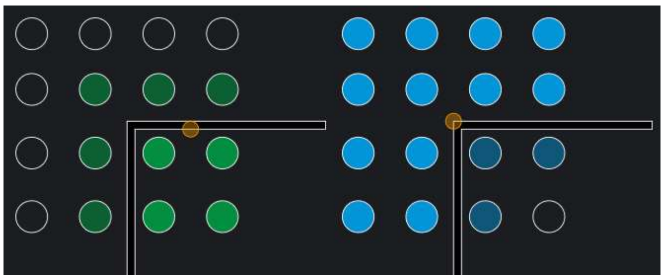


First we offset the sampling position by a distance along the pixel's normal to reduce the influence of probes on the backside. We've been using this approach till now, but it doesn't solve all the issues, especially with pixels located in corners.

Light Leaking

- Normal Offset
- **Interior Volume**

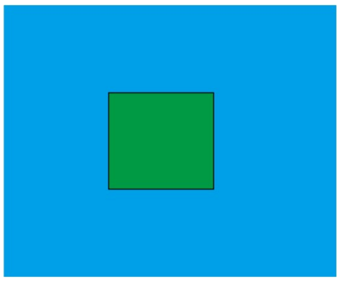

Ideal



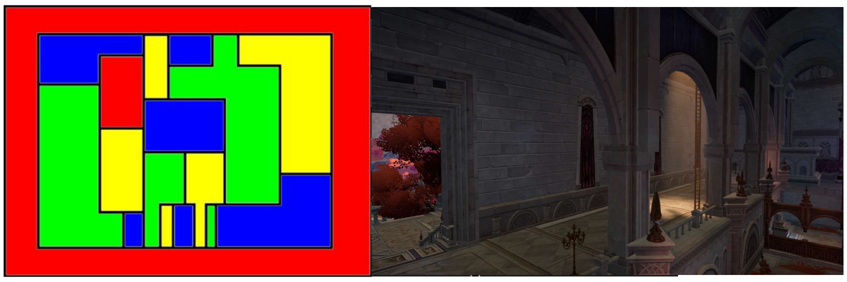
I once tried using interior volumes to distinguish between indoor and outdoor areas,

Light Leaking

- Normal Offset
- **Interior Volume**



Ideal



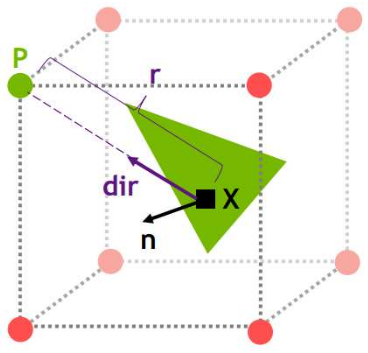
Reality

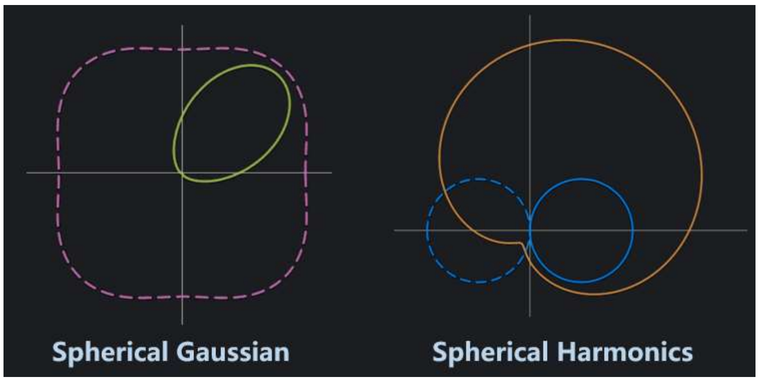
but the adjacency between rooms is much more complex than just a simple indoor-outdoor separation.

Light Leaking

LIGHTSPEED STUDIOS | GDC

- **DDGI → Depth Comparison**





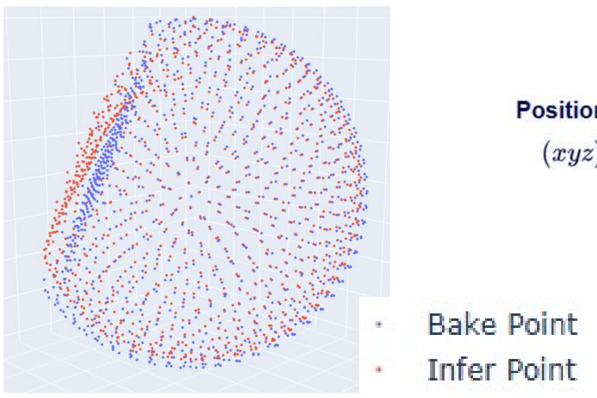
https://developer.download.nvidia.cn/video/gputechconf/gtc/2019/presentation/s9900-irradiance-fields-rtx-diffuse-global-illumination-for-local-and-cloud-graphics.pdf

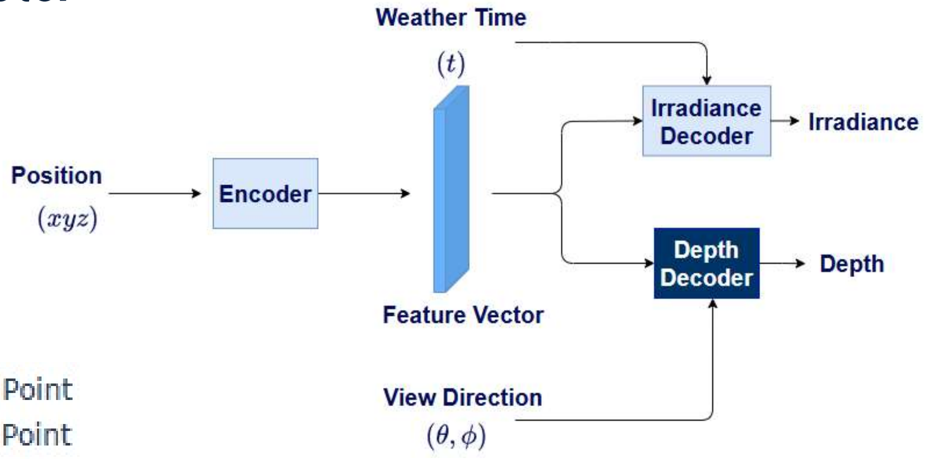
The DDGI method tracks the depth buffer around probes. The weights of eight adjacent probes can be adjusted during interpolation based on visibility. It's effective but requires a lot of space for recording the depth. We've tried to compress the depth buffer using methods like spherical harmonics and spherical Gaussians. However, neither of them can accurately fit the depth shapes and distributions around the probes of various corners.

Light Leaking

- Multi-Task Learning

  Irradiance & **Depth**
- **Only Extra 17kb/sector**





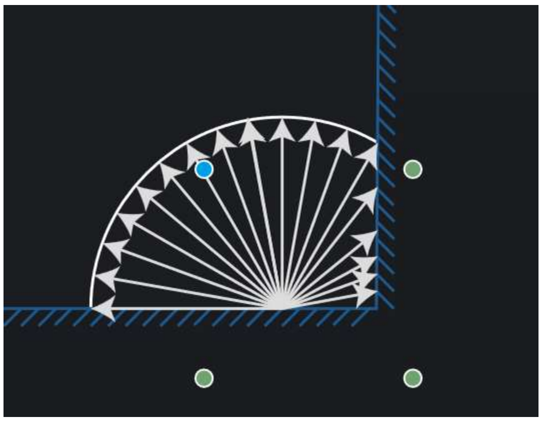
Considering the advantages of neural networks in fitting, we tried to modify the previous neural network to enhance its capability in fitting depth. The modified model remains simple, requiring only an additional 17kb of data per sector to achieve decent fitting results around the probe. On the other hand, while training for irradiance initially took less than one minute per sector, integrating massive depth data extended the duration significantly, with training time nearing 40 minutes on 3090 per sector. Therefore, this method is not very practical for our workflow.

Light Leaking

- Normal Offset
- Interior Volume
- Depth Comparison
- **Screen Space Tracing**

  Screen Probe / 8x8 pixels

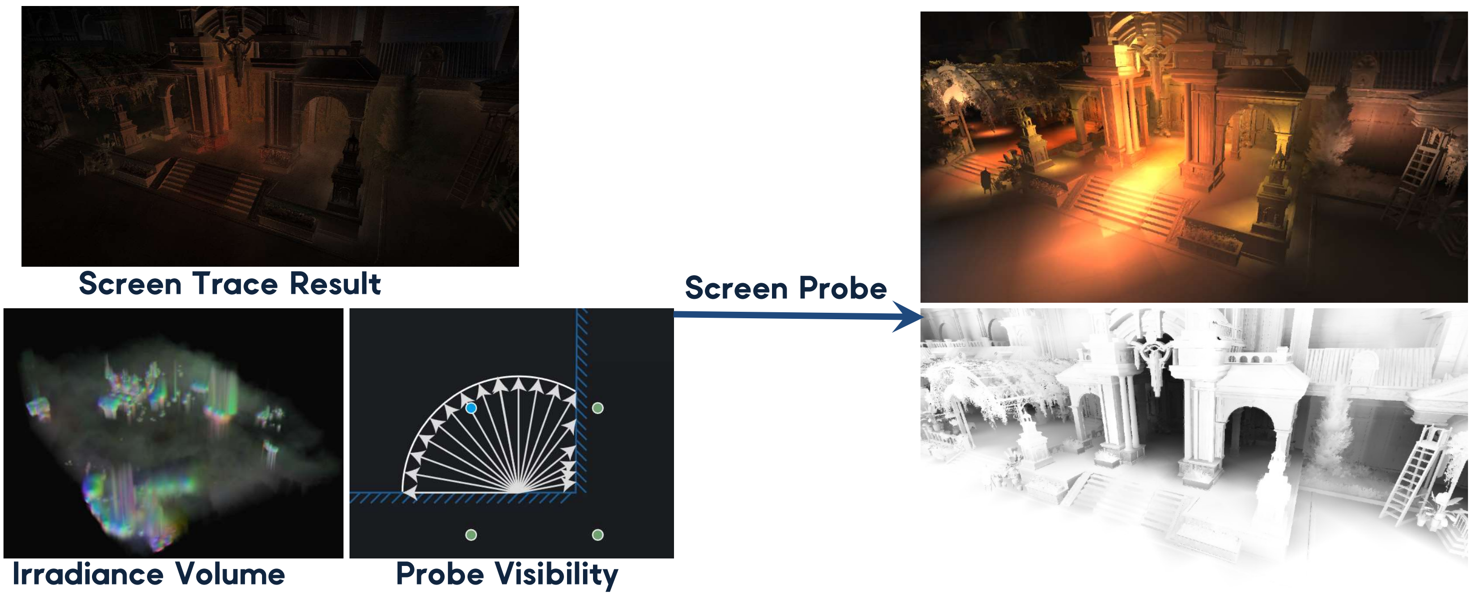
  Weight of Probe by visibility



https://dev.epicgames.com/documentation/en-us/unreal-engine/lumen-global-illumination-and-reflections-in-unreal-engine

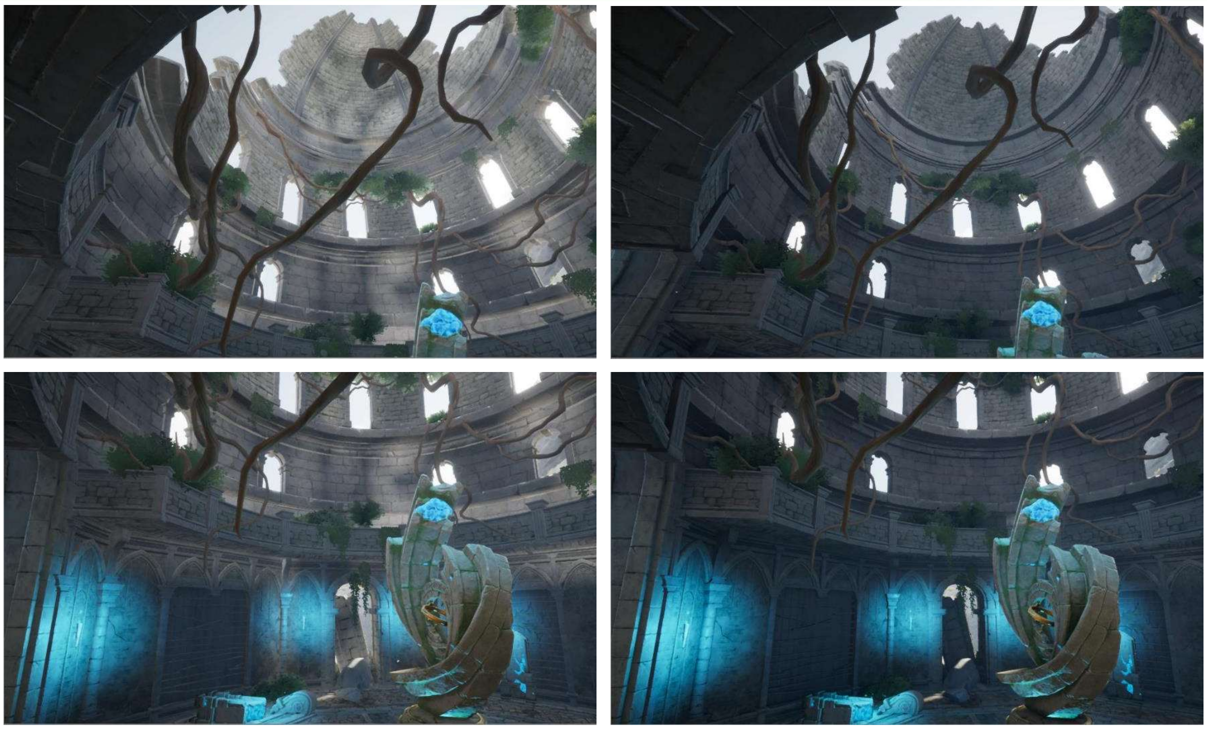
Finally, we returned to the screen space probe gathering method. We employ a pixel density of 8x8 for each screen space probe and accumulate the tracing results into a compact radiance map. During the screen space tracing process, we check if a probe is reachable from each pixel; this helps us decide whether to accumulate results based on that specific probe's visibility.

Screen Space Probe Gather



By screen probe gathering, we integrate the results from both screen space tracing and prebaked Irradiance Volume. This method allows us to combine high-frequency information from screen space and low-frequency baked irradiance to prevent most ghosting. Critically, by adjusting the weights of the probes based on visibility from pixels, we manage to prevent most instances of light leakage.

Light Leaking



The left image displays the results before screen space probe gathering was enabled, while the right image displays the results after its activation. You can clearly see that the issue of light leakage has been significantly mitigated.

Now that we've addressed near-field challenges,

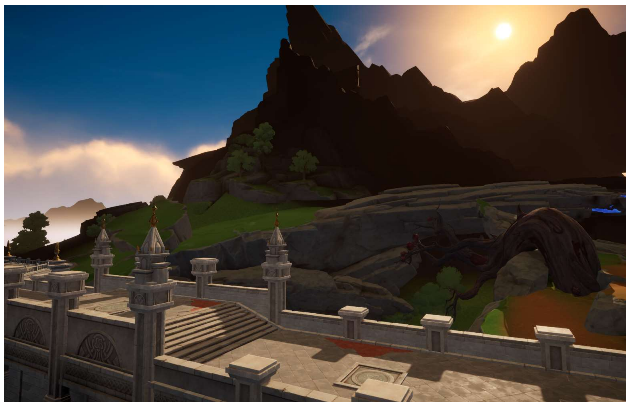
VIDEO-BASED DECODE

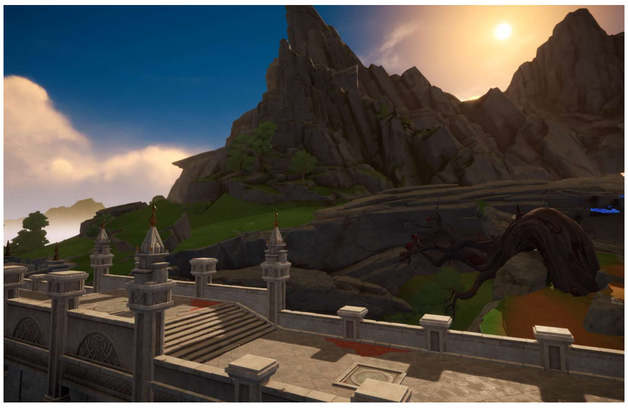
HEVC For Global 2D Map

let’s discuss how to handle distant scene Time of Day Global Illumination.

Global GI Map

- Beyond Irradiance Volume


Without Global Map


With Global Map

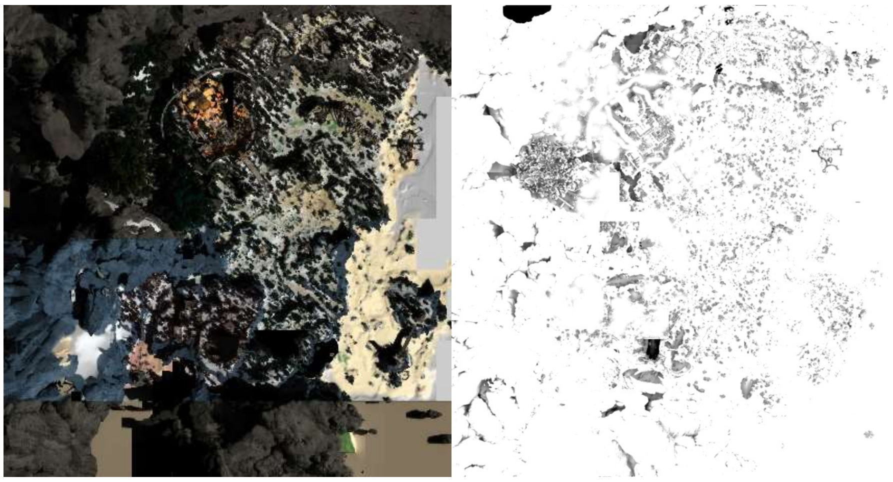
Within the 512m x 512m range, we utilize Irradiance Volume. For areas beyond this, we implement a 2D global map texture for distant views.

Global GI Map

- RGB： Irradiance color

Change With TOD

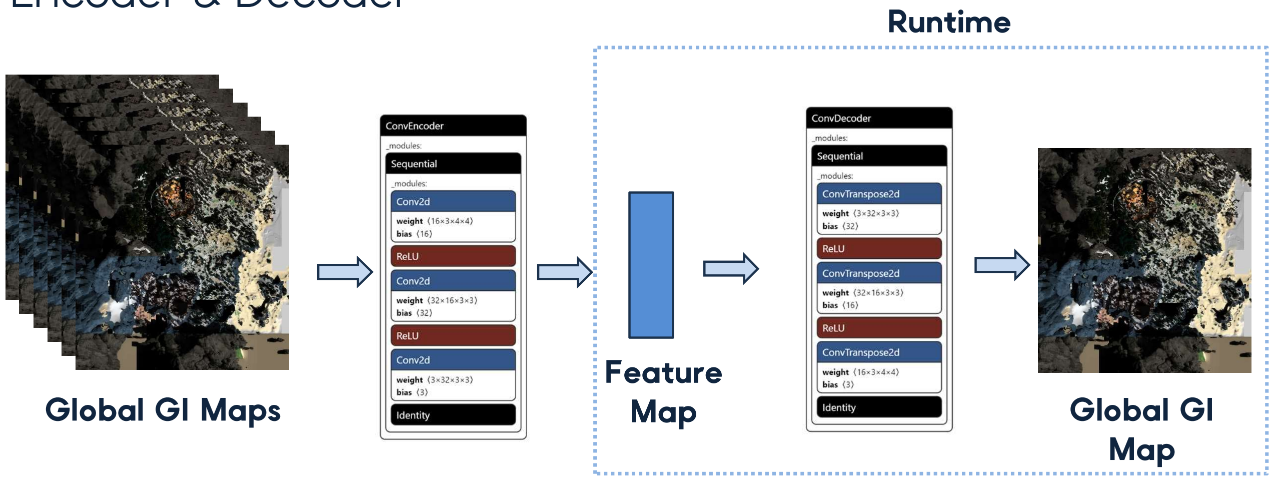
- Alpha： Sky Visibility



Using a GlobalGIMap for distant scenes is a common approach. The RGB channels capture the top-down projected irradiance color, while the alpha channel encodes Sky Visibility. Similar to the previous discussions, Time of Day variations require storing multiple pre-baked GlobalGIMaps. We need to compress them again.

Convolutional Neural Network

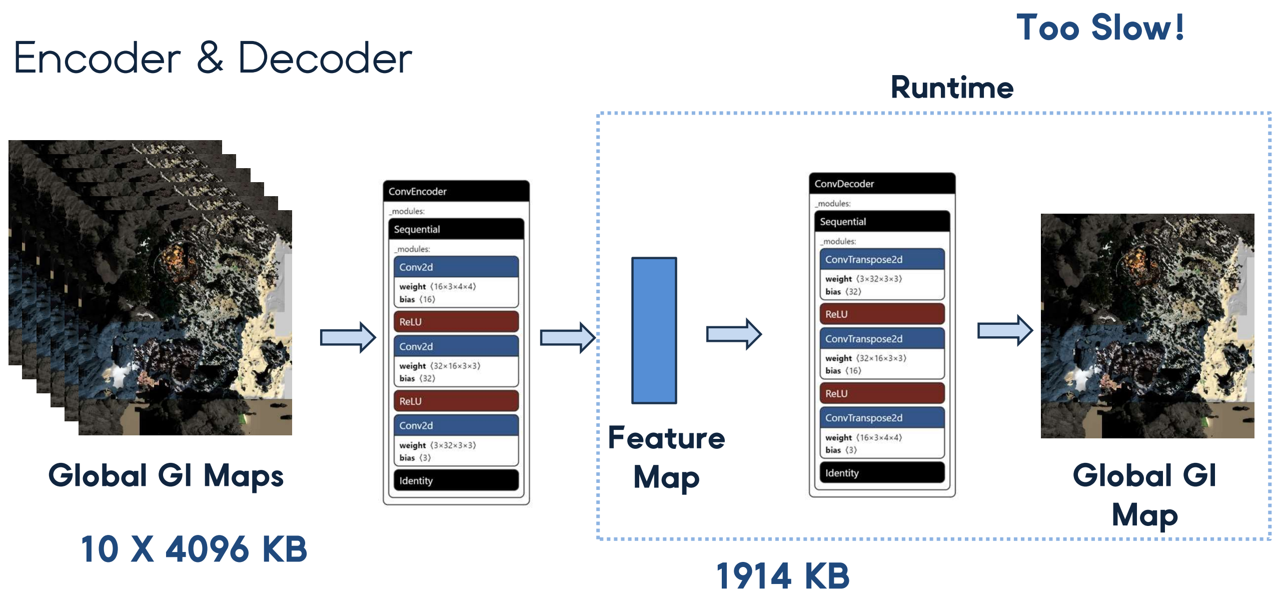
- Encoder & Decoder



While BC7 and ASTC provided a reliable baseline for texture compression, the increasing number of textures motivated us to pursue higher compression ratios. Therefore, we experimented with using Convolutional Neural Networks to achieve this goal.

Convolutional Neural Network

- Encoder & Decoder

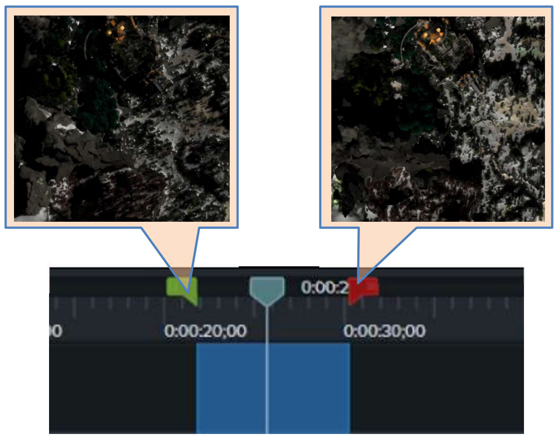


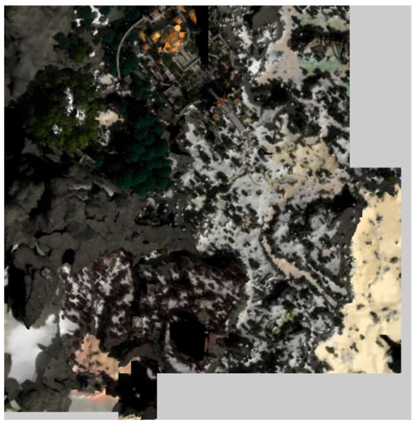
The compression rate of CNN is acceptable, but real-time inference speed became a critical bottleneck. and the precision of this model is also not enough.

Hence we abandoned this solution.

Video-based Compression

- **Multiple Textures→ Video**



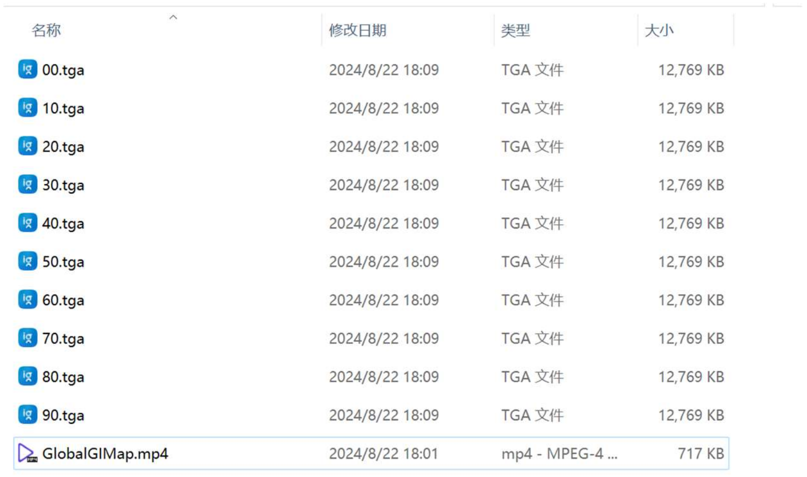


By analyzing the relationships between multiple textures, we turned to video-based compression. Treating textures as video frames allows us to leverage temporal coherence, achieving higher compression ratios without compromising visual quality

HEVC

LIGHTSPEED STUDIOS | GDC

- Using FFMPEG encode in H.265 format
- 1024 x 1024 10 frames
- Color format: YUV420p
- Bitrate type: VBR, 8000k



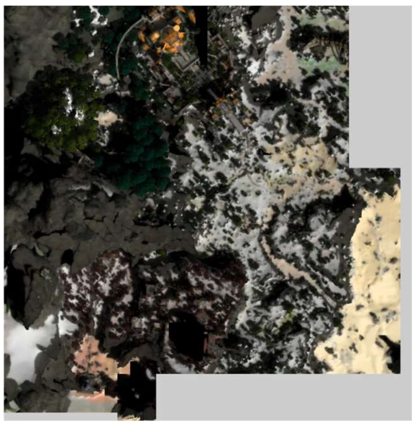
名称	修改日期	类型	大小
00.tga	2024/8/22 18:09	TGA 文件	12,769 KB
10.tga	2024/8/22 18:09	TGA 文件	12,769 KB
20.tga	2024/8/22 18:09	TGA 文件	12,769 KB
30.tga	2024/8/22 18:09	TGA 文件	12,769 KB
40.tga	2024/8/22 18:09	TGA 文件	12,769 KB
50.tga	2024/8/22 18:09	TGA 文件	12,769 KB
60.tga	2024/8/22 18:09	TGA 文件	12,769 KB
70.tga	2024/8/22 18:09	TGA 文件	12,769 KB
80.tga	2024/8/22 18:09	TGA 文件	12,769 KB
90.tga	2024/8/22 18:09	TGA 文件	12,769 KB
GlobalGIMap.mp4	2024/8/22 18:01	mp4 - MPEG-4 ...	717 KB

10 x 4MB $\rightarrow$ 717KB

Let’s take the TOD GlobalGIMap across 10 different time points as an example. We applied ffmpeg to compress ten textures, each 1024x1024 in size, using the YUV420p format. This process resulted in a compressed size of just 717KB.

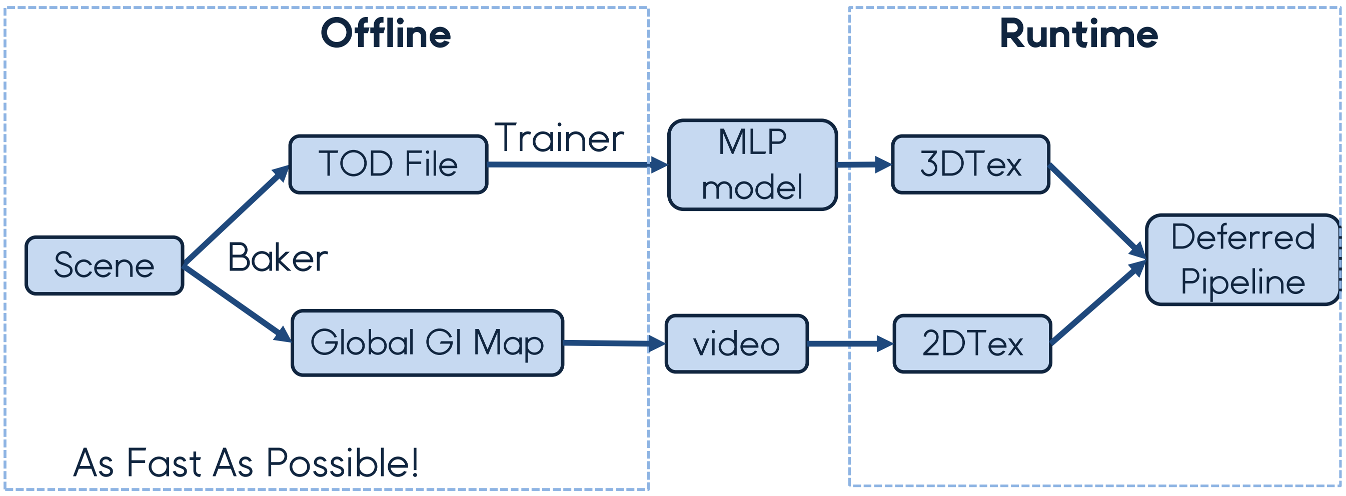
Runtime Decode

- Hardware Decoding
- Called at time change(per minute)
- Asynchronous Threads (in 1 frame)



The decompression process is also quite streamlined. For instance, in Unity, we can utilize the native VideoPlayer to decompress textures frame-by-frame. When there is a change in time, the decompression can be performed asynchronously on a background thread, ensuring that it doesn't block the main thread. This setup allows the decompression to be completed within one frame.

Workflow



During runtime, we decode a 3D texture from the trained model, extract the GlobalGIMap from the video, and then perform a screen-space gather in the deferred pipeline.

In production workflows, it's necessary to complete the baking and training process for multiple times of day. This process needs to be as fast as possible

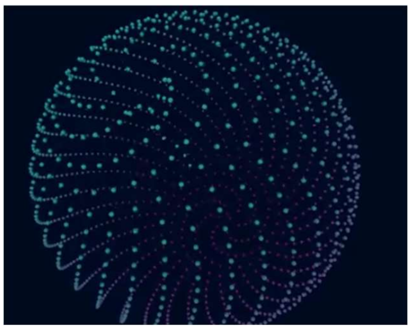
WORKFLOW OPTIMIZATION

HWRT Baker & MLP Trainer

Let's delve into our hardware ray tracing baker and the MLP training workflow.

HWRT Baker

- Probe Based, without 2UV
- DXR based
- 1024 rays in predefined directions
- Accumulate to Spherical Harmonics

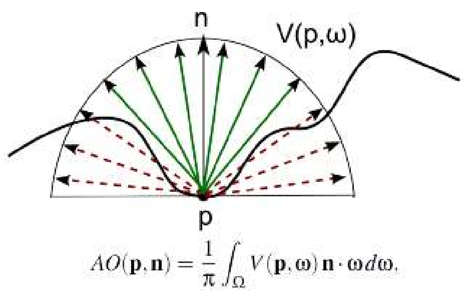

Fibonacci Sampler from a Probe

Since most bakers are based on Lightmaps, they require an additional process of creating a second set of UV maps for objects. We can bypass this step by using probes. Moreover, we can further accelerate by developing our own Hardware Ray Tracing baker, developed with DXR. This baker dispatches multiple rays, such as 1024 rays, from every probe in predefined directions, gathers the outcomes of various bounces, and projects them onto Spherical Harmonics (SH).

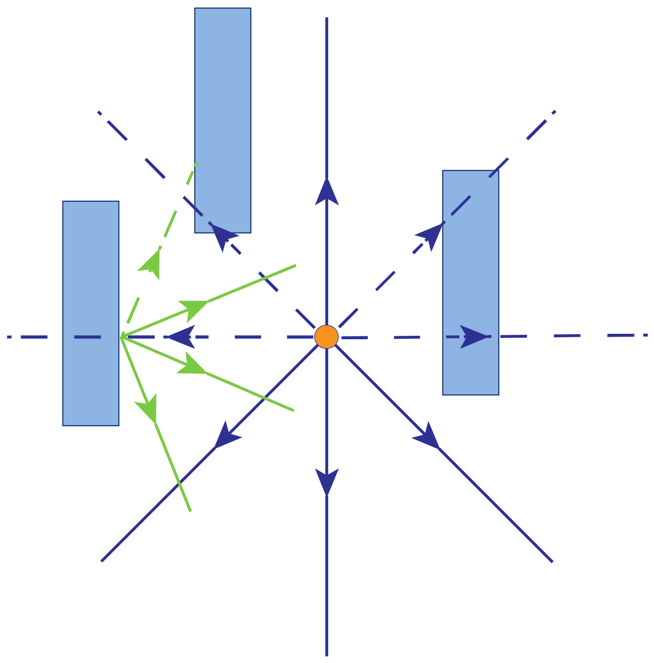
HWRT Baker

- Lights
- **Sky Visibility**/AO

  second rebound



$$AO(\mathbf{p},\mathbf{n}) = \frac{1}{\pi}\int_{\Omega} V(\mathbf{p},\omega)\,\mathbf{n}\cdot\omega\,d\omega,$$



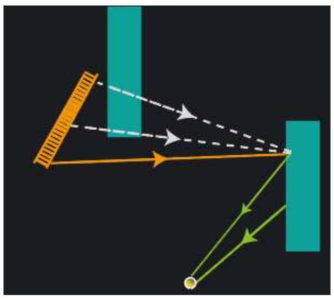
The method for computing Ambient Occlusion (AO) generally involves recording the proportion of rays that hit obstacles in the surrounding area. In addition to recording the collision ratio of rays emitted from probes, we also take into account the hit ratio of rays after they rebound off objects. The second rebound allows for a more natural transition in sky visibility.

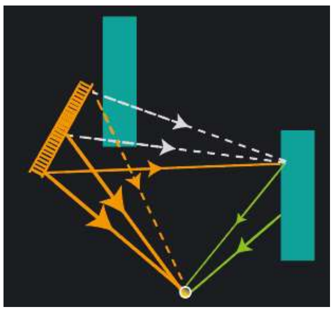
HWRT Baker

- Lights

  Mixed/Baked

  Directional/Point/Spot/**Rectangle/Disc**


Indirect


Direct + Indirect

Like most GI bakers, our system supports a variety of lighting types including directional, point, spot, and area lights. Additionally, we offer modes dedicated to baking either indirect lighting alone or integrating direct lighting into the final GI result.

HWRT Baker

- Bake time: 1 min/sector → **3 seconds/sector**
- **Customization**

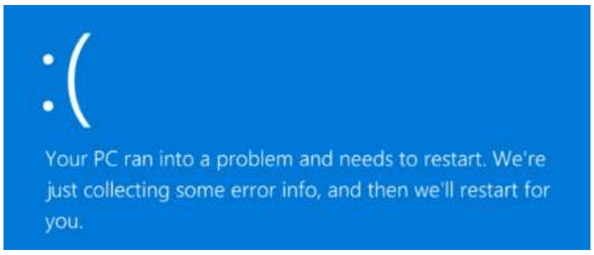
  GBuffer Depth baker supported

  Different Transmission Rate…

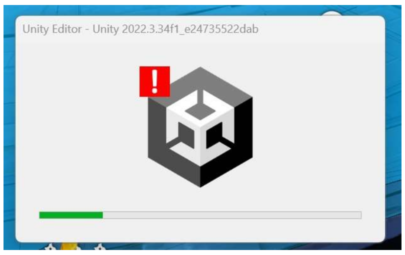
With our hardware ray tracing baker, baking time per sector has dropped from one minute to under three seconds. Our custom baker allows us to tailor the process, handle special data like Gbuffer Depth around probes, and adjust for varying transparency of vegetation and other materials. This enhances SkyVisibility baking, resulting in a smoother gradient.

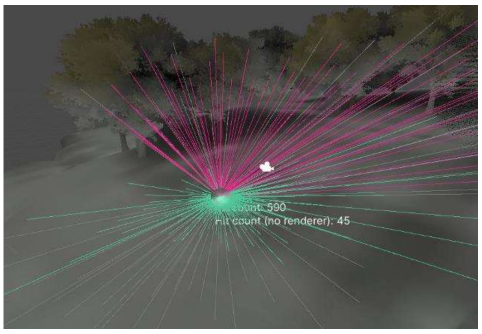
Debug Tools

- NSight/Aftermath/DRED...
- Custom Visualization Tools


:(

Your PC ran into a problem and needs to restart. We're just collecting some error info, and then we'll restart for you.

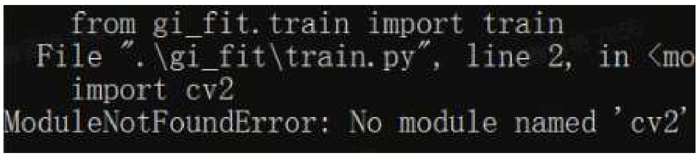

Unity Editor - Unity 2022.3.34f1_e24735522dab



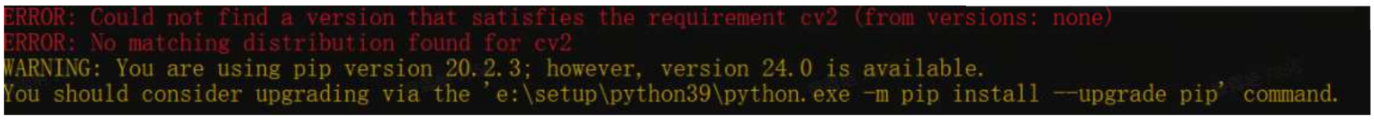
Developing with ray tracing APIs often leads to various bugs, such as engine crashes and DX12 device remove issues. To address these, we use debugging tools like NSight and have developed our own tools. For example, our tools can display ray distribution near each probe to help identify problematic materials or objects causing baking errors.

Why Docker?

- **Docker** → Import predefined images
- Free from set up training environment



```
    from gi_fit.train import train
  File ".\gi_fit\train.py", line 2, in <mo
    import cv2
ModuleNotFoundError: No module named 'cv2'
```

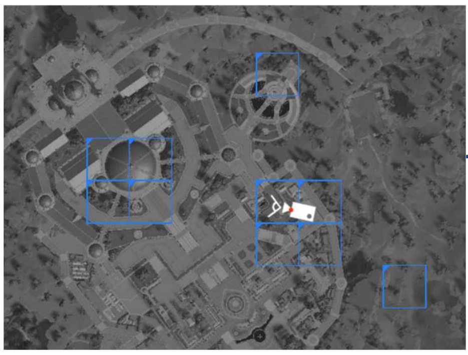


```
ERROR: Could not find a version that satisfies the requirement cv2 (from versions: none)
ERROR: No matching distribution found for cv2
WARNING: You are using pip version 20.2.3; however, version 24.0 is available.
You should consider upgrading via the 'e:\setup\python39\python.exe -m pip install --upgrade pip' command.
```

After finishing the baking process, the next step involves using training for compression. We completed the training process using PyTorch, but deploying it on various machines presented numerous challenges. For someone new to neural networks like myself, setting up the environment can be quite troublesome. After configuring my own machine, we also need to help the artists resolve configuration issues on dozens of other machines, like baking farms, which is very time-consuming. Therefore, we've opted to use Docker to import predefined images to simplify the setup process.

Trainer in Editor

- On Server/On Local PC
- Compressing and Baking in parallel

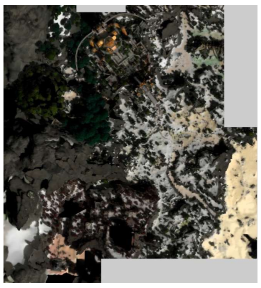


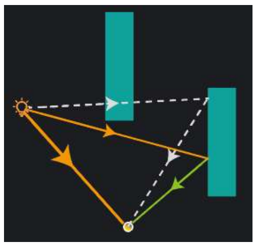
Select and Go

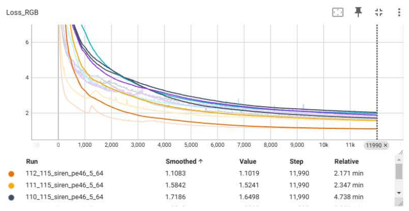
<1min/sector

< 5 hours/1kmx1km

< 2 hours with fewer Points of Interest





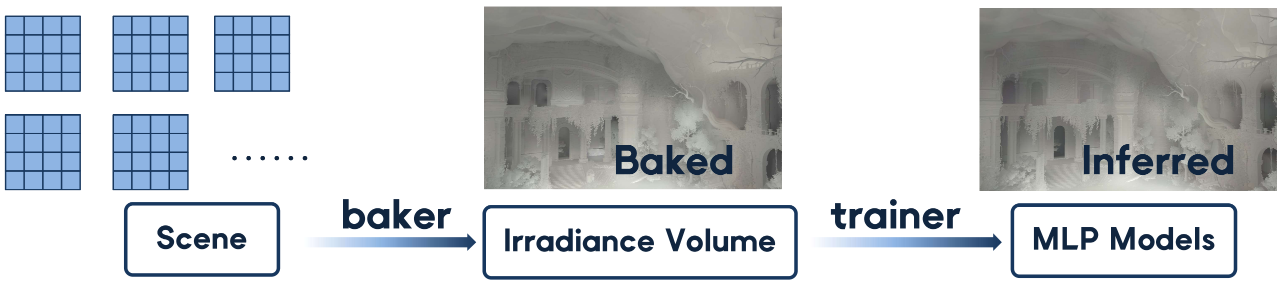


	Run	Smoothed ↑	Value	Step	Relative
●	112_115_siren_pe46_5_64	1.1083	1.1019	11,990	2.171 min
●	111_115_siren_pe46_5_64	1.5842	1.5241	11,990	2.347 min
●	110_115_siren_pe46_5_64	1.7186	1.6498	11,990	4.738 min

The entire process is seamlessly integrated into a user-friendly scene editor. Artists need only select the modified sectors and click a button, at which point both the baking and training processes run simultaneously. The training can be deployed either locally or on a server. Time efficiency is optimized, with the process for a 64m x 64m sector taking less than one minute. Consequently, a scene measuring one kilometer by one kilometer can be completely processed in under five hours using a single RTX 3090. For sectors with fewer points of interest (POIs), such as simple seascapes and terrains, the processing time can be further reduced.

Trainer in Editor

- Distribute Tasks on Different Machines based On Sectors
- CI tasks for baking and training when needed
- Intermediate Results can be Displayed for Debug



By distributing tasks across different machines based on sectors, we can further reduce the time required by several multiples. Additionally, we provide CI tasks that enable convenient remote operations or scheduled baking and training when scenes are modified. Operations such as the selection of MLP layers, post-processing of GlobalGIMaps, and regular saving are made transparent to users. If any issues arise with the results, the baking and inference results can be displayed separately to identify where the problem occurred.

Conclusion

- **Neural Compression & Video Compression**
  Consistent and Scalable GI Effects

- **User-Friendly Workflow is More Important**
  A tool unused is a tool useless

In summary, by utilizing neural compression, we have managed to achieve consistent and scalable GI effects across various platforms with minimal storage requirements. Employing video compression for distant scenes has also proven to be highly effective.

Moreover, to ensure better implementation of these solutions, we have invested more time in refining our workflows. We believe that the true merit of these innovations lies in their application to projects, not just their theoretical capabilities.

Future works

LIGHTSPEED STUDIOS | GDC

- Accelerate Workflow
- Refine GI compression rates
- NPU-accelerated inference across more devices
- Apply Neural Acceleration to other content

In our future work, we will accelerate our workflow by optimizing model structures and improving workload distribution strategies. We will continue to refine GI compression rates and enhance visual quality. More significantly, we aim to extend the use of NPU-accelerated inference frameworks across more devices, especially mobiles, promoting collaboration between GPUs and NPUs. Furthermore, we plan to extend neural acceleration to more rendering techniques and various aspects of game content development and compression.

Reference

- https://www.sci.utah.edu/~bigler/images/msthesis/The%20irradiance%20volume.pdf
- https://research.nvidia.com/publication/2021-06_real-time-neural-radiance-caching-path-tracing
- https://www.activision.com/cdn/research/Neural_Light_Grid.pdf
- https://developer.download.nvidia.cn/video/gputechconf/gtc/2019/presentation/s9900-irradiance-fields-rtx-diffuse-global-illumination-for-local-and-cloud-graphics.pdf
- https://www.gdcvault.com/play/1015326/Deferred-Radiance-Transfer-Volumes-Global
- https://dev.epicgames.com/documentation/en-us/unreal-engine/lumen-global-illumination-and-reflections-in-unreal-engine
- https://www.iryoku.com/smaa/downloads/SMAA-Enhanced-Subpixel-Morphological-Antialiasing.pdf
- https://facebook.github.io/zstd
- https://research.activision.com/publications/2021/09/moving-basis-decomposition-for-precomputed-light-transport
- https://github.com/davidAlgis/InteropUnityCUDA

These are the references related to my work, which have been of great help to me.

Acknowledgements

- **Programmers**

  Jiyang Li, Zhao Zhang, Senhang Chen, Minjie Zhang,

  Donghai Huang, Yanfei Liu, Bin Yang,

  Cheng Yang, Junjie Zhu, Yixuan Qin

- **Artists**

  Hu Xuan, Yao Guan, Shuaishuai Ding, Xiaowei Qi……

I would also like to express my gratitude to my directors, teammates and collaborators. My teammates have a deeper understanding of neural networks than I do, and they advised me to explore video-based compression. And the creativity of the artists has had a greater impact on the visuals than my work.

THANKS

luyancao@lightspeed-studios.com

LIGHTSPEED STUDIOS | GDC