
LIGHTSPEED
STUDIOS
GDC
MEET
LIGHTSPEED STUDIOS
AT GDC2023
March 20-24, 2023 | San Francisco, CA

Exploring the Technical Arts in the Development of UNDAWN

Joshua Yu

Lead Technical Artist of UNDAWN,

LIGHTSPEED STUDIOS

Leon Wei

Engine Team Lead,

LIGHTSPEED STUDIOS

March 20-24, 2023 | San Francisco, CA

Technical Arts in UNDAWN

Joshua Yu

Lead Technical Artist of UNDAWN,

LIGHTSPEED STUDIOS

March 20-24, 2023 | San Francisco, CA

Introduction

- UNDAWN is a next-generation MMORPG mobile game developed by LIGHTSPEED STUDIOS , powered by Unreal Engine 4 with SOC (Survival, Open World, Crafting) as its core experience.
- Players can use a variety of firearms to experience a multitude of gameplay options and an engaging storyline in the open world. This includes scavenging in abandoned cities, collecting and hunting in the wilderness, driving across the continent, searching for surviving towns, competing for strongholds and settlement battles, and finding partners to survive.

Technical Arts

1. Ocean Simulation
2. Techniques for the Buildings
3. Vertex Magics in VFX

I Ocean Simulation

Limitations on Mobile Devices

Tessellation doesn't work on all targeted devices. Even on high-end mobile devices such a feature is still a **performance killer**.

- Order-independent-transparency consumes **a huge amount of bandwidth**, causing the GPU to overheat.
- Tessendorf FFT is **relatively expensive** for mobile platforms and **heavy** for the CPU to simulate convincing buoyancy.

$$\tilde{h}(k,t) = \tilde{h_0} exp\{i\omega(k)t\} + \tilde{h_0}^{*} exp\{-i\omega(k)t\}$$

$$\vec{D}\left(\vec{x},t\right) = \sum_{\vec{k}} -i\frac{\vec{k}}{k}\tilde{h}\left(\vec{k},t\right) exp(i\vec{k}\cdot\vec{x})$$

$$\tilde{h_0}(k) = (\xi_r + i\xi_i)\sqrt{P\,(k)/2}$$

$$\omega^2(k) = gk(1 + k^2L^2)$$

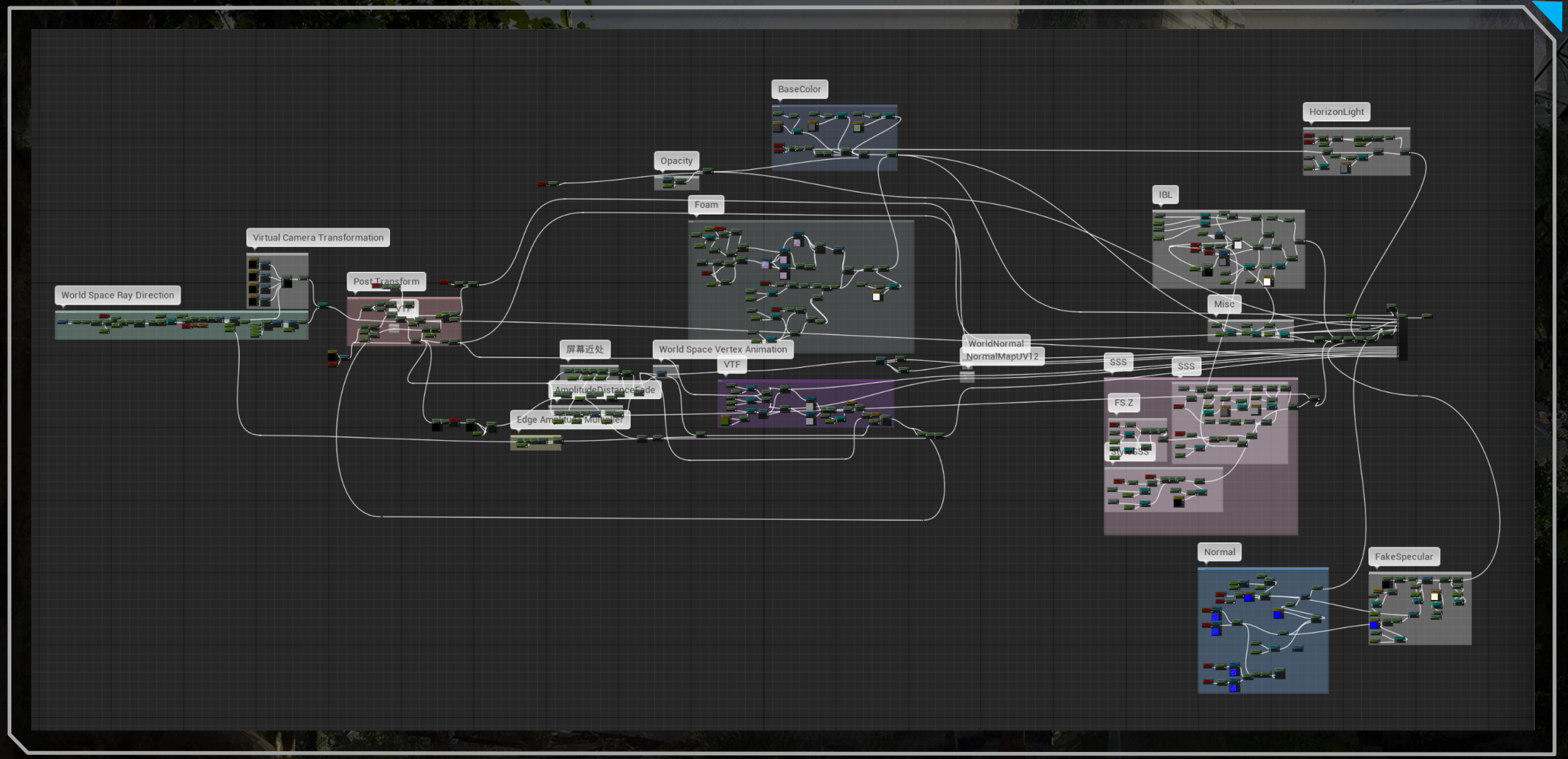
Limitations on Mobile Devices

- The ocean is expected to be compatible with all of our targeted devices, given that it is **directly linked to the gameplay**.
- We have allocated **very limited CPU and GPU performance** budgets for the ocean.
- Yet we still desire an ocean surface that is **transparent**.

Ocean Simulation in UNDAWN

- Utilize **Projected Grid** to avoid tessellation and CDLOD calculations.
- Use **4 Gerstner Waves** in the Vertex Shader for **significant wave displacements** and buoyancy simulation.
- Use a **4-channel tiled noise texture** for **additional displacement details** via VTF.

Ocean Simulation in UNDAWN



Tricks on Transparency

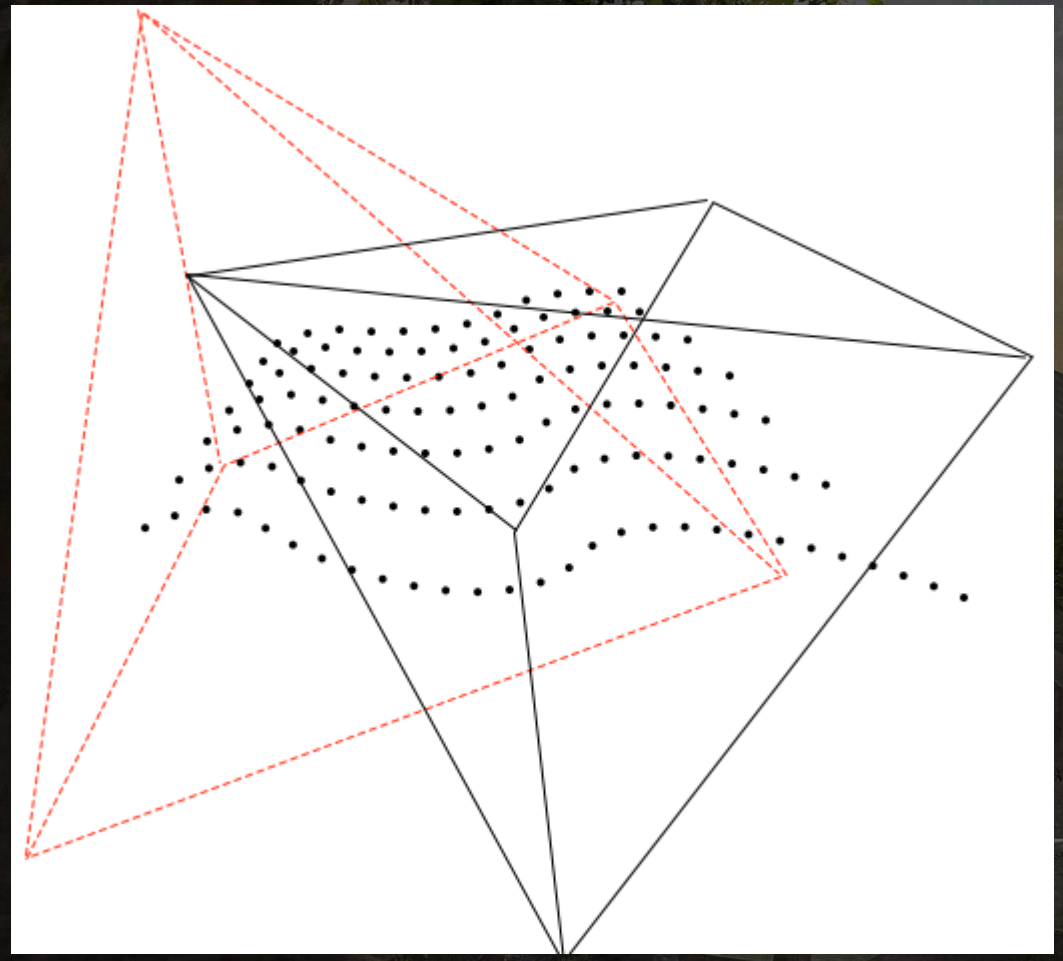
- OIT is totally beyond discussion.
- Double Pass works, but the water plane contains more than 30,000 vertices, taking up a large portion of the screen. We **cannot double the vertex shader nor overdraw**.
- Since the mesh is always projected from the view space, **the relative order** of mesh primitives on the screen will never change.
- If we ensure that the "lower" a primitive is on the screen, the smaller its primitive ID is and enable depth writing, we can make sure that water pixels behind it **will not pass the depth test**.
- Although this is not a true "Transparency Sorting," it still **gives satisfactory results**.


Everything went on smoothly
UNTIL ...

Why Bumpy?

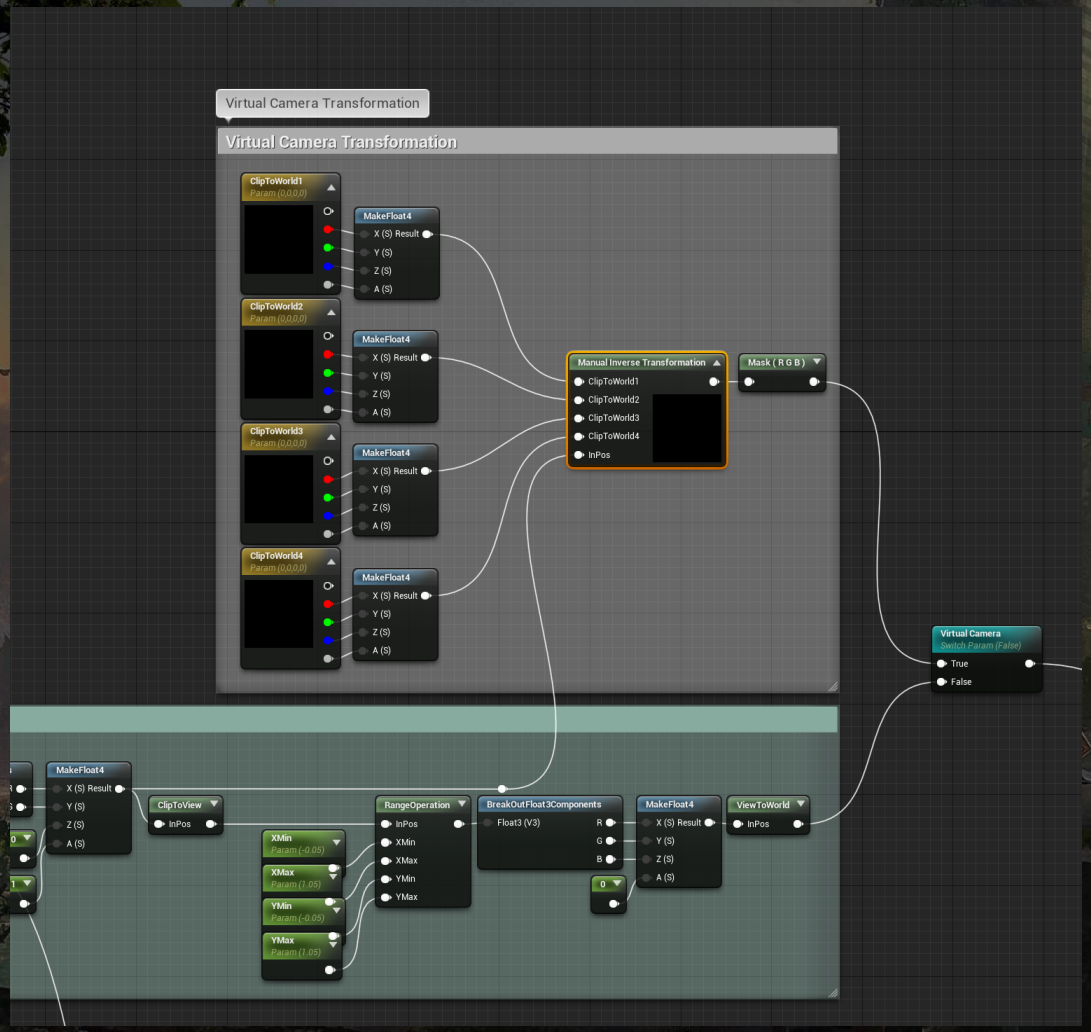
- The **vertex density** dramatically reduces as the distance increases.
- Based on parameters including FOV and etc. in UNDAWN, we have 50 vertices per meter around our character, yet that number dramatically drops to **0.06** on the water plane 200 meters away.

Solution to flickering



- Squeeze the vertices on Y direction to make sure **no vertex is clipped** – make each and every vertex into use.
- Introduce **Virtual Camera** to fix flickering under grazing angles – When the camera shoots horizontally it is unlikely to see the distant of ocean anyway.
- Re-arrange vertex density on the water plane.

Ocean Simulation in UNDAWN



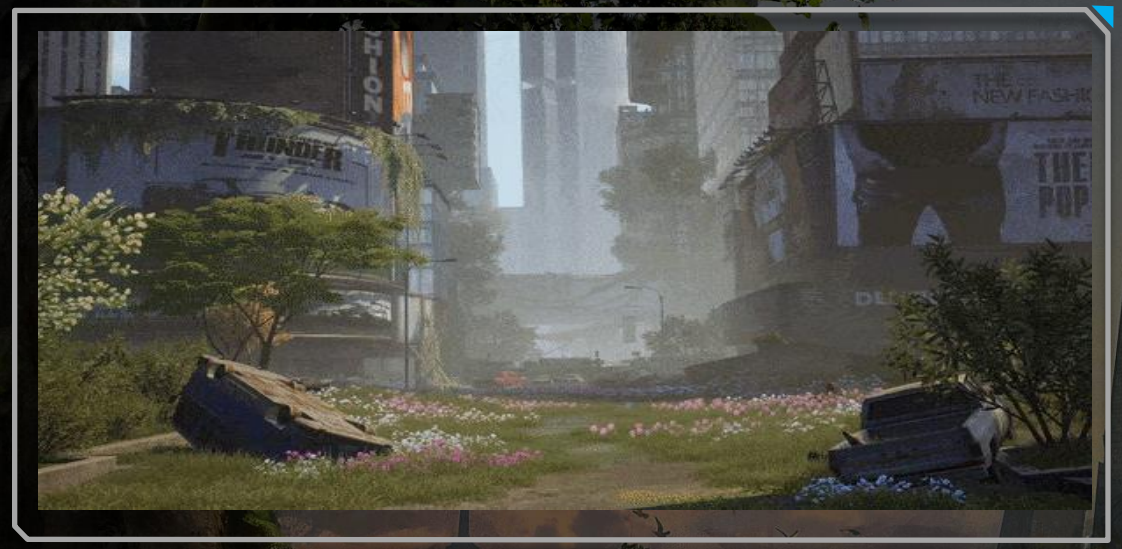

I thought that was a wrap
BUT …

Overlook Depth for Amplitude attenuation

- The ocean waves are supposed to be more **gentle** on the shoreline.
- We used an orthogonal camera 50 meters above the player and point it down to render rocks and landscapes to the **Overlook Depth Texture**.
- Determine the amplitude of waves by comparing overlooking depth and the height of water plane in Vertex Shader.
- Overlook Depth Texture is not updated per-frame to **save performance**. It is also used by **rain-roof collision** and **snow footprints feature**.

II Techniques for the Buildings

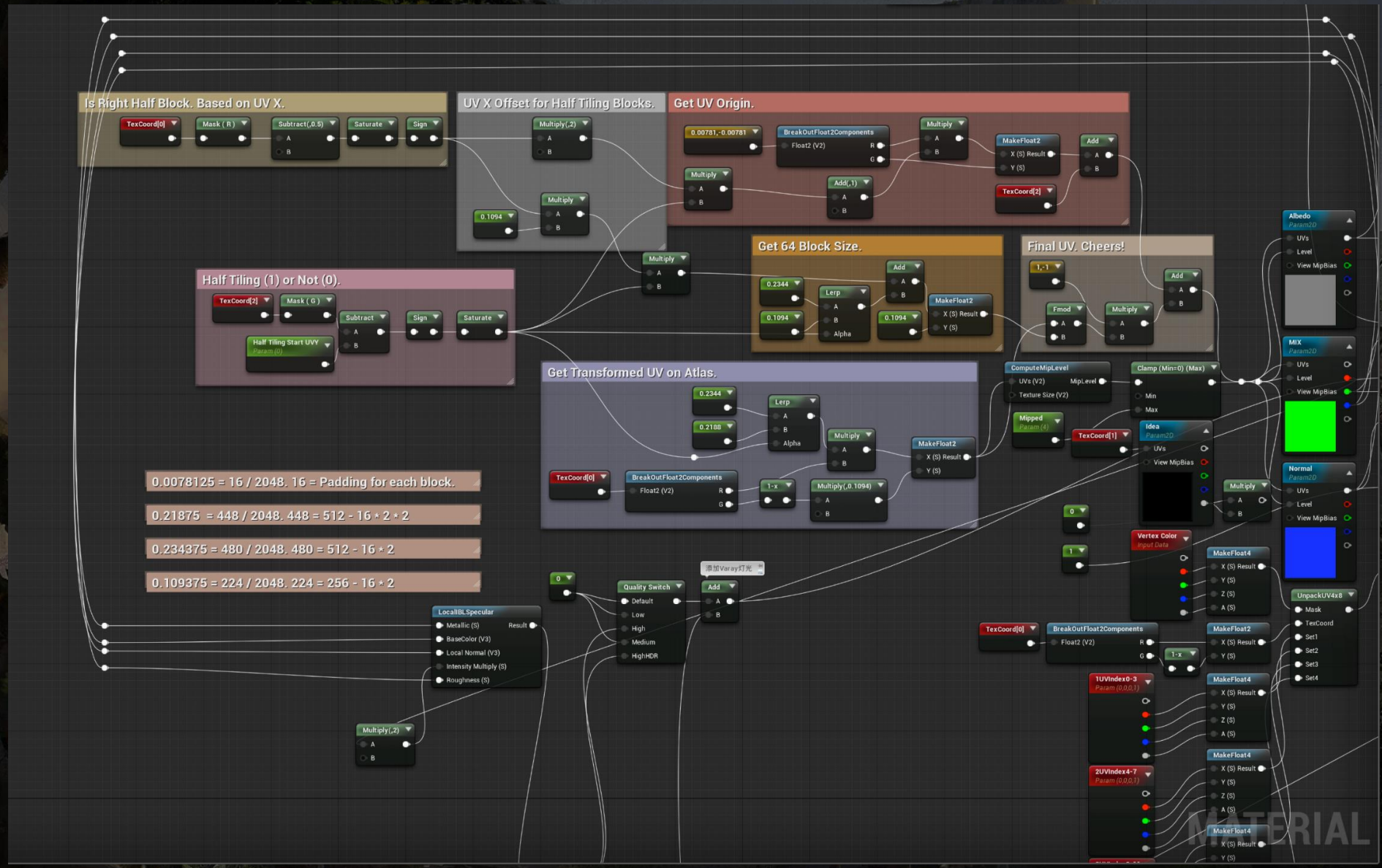
Challenges for Buildings



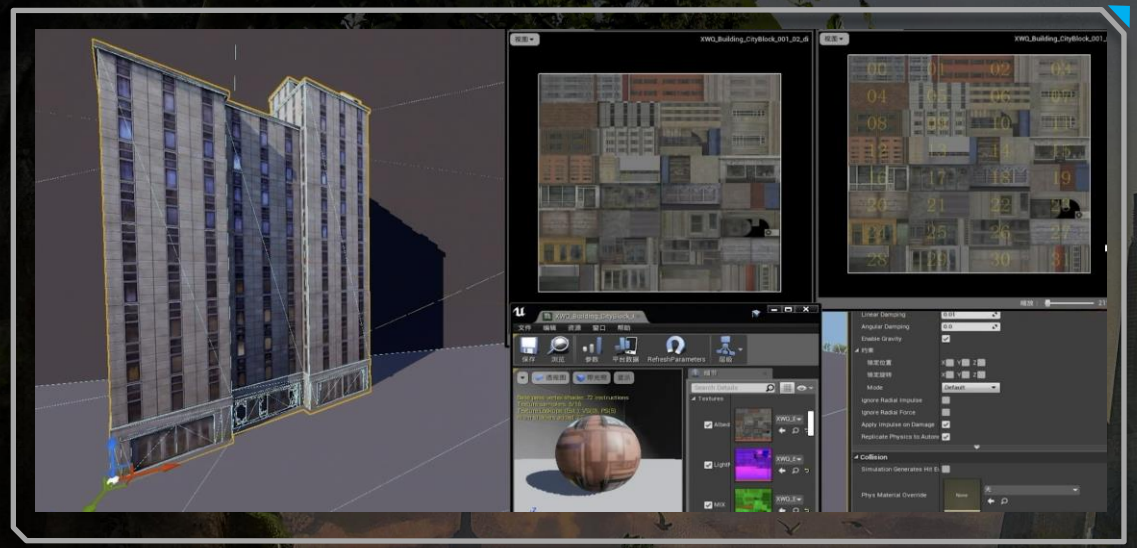
- There are **PLENTY** of buildings in the Lost City of UNDAWN.
- Other environment props have already occupied a large amount of GPU performance.
- We had an extremely tight budget for **texture memory, triangles and draw calls** for buildings.

Solution – Tiling based on Atlas and Dynamic UV

- Encode Vertex Color RGBA attributes of meshes in binary for **16 material IDs**.
- Map between Vertex Color and real UV coordinates in Vertex Shader.
- Perform **FMOD operation** in Pixel Shader for UV coordinates to complete tiling.


Is Right Half Block. Based on UV X.
UV X Offset for Half Tiling Blocks.
Get UV Origin.
Half Tiling (1) or Not (0).
Get 64 Block Size.
Final UV. Cheers!
Get Transformed UV on Atlas.
0.0078125 = 16 / 2048. 16 = Padding for each block.
0.21875 = 448 / 2048. 448 = 512 - 16 * 2 * 2
0.234375 = 480 / 2048. 480 = 512 - 16 * 2
0.109375 = 224 / 2048. 224 = 256 - 16 * 2
MATERIAL

Solution – Tiling based on Atlas and Dynamic UV



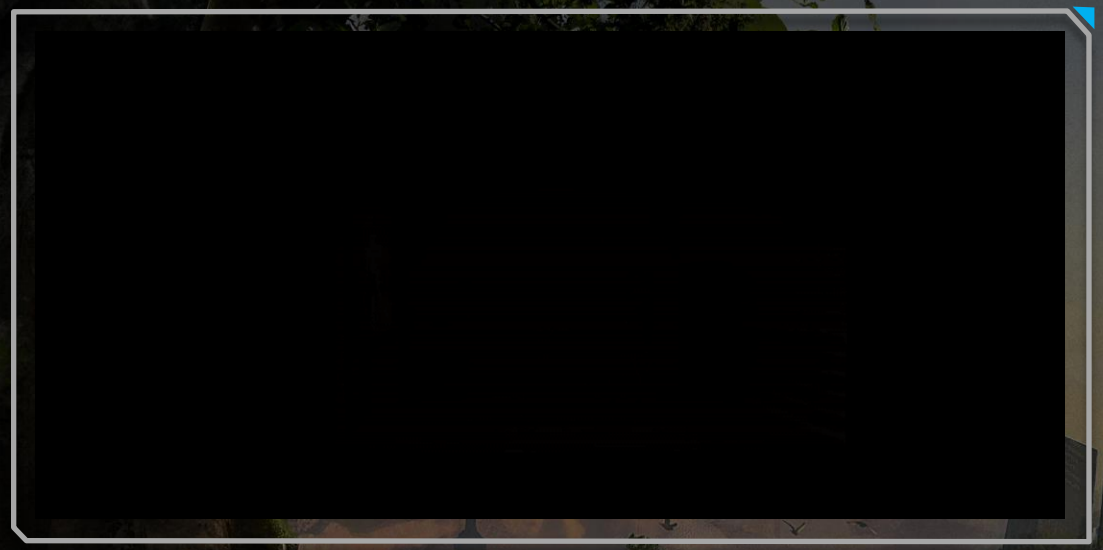
- Supports texture reuse to **reduce texture memory consumption**.
- The mapping relationship is stored in Instance Buffer rather than Uniform Buffer to support **instancing to reduce draw calls**. (Same mesh, same atlas, different block combinations)
- Supports tiling **to reduce the number of triangles**.


But no PAIN no gain …

Problems and Explanations

- The FMOD operation on UV coordinates should be accompanied by Mip levels derived from **DDX&DDY**.
- Use texture block **padding** to resolve Mip Seam problems.
- Calculating UV coordinates in Pixel Shader involves **Dependent Texture Fetch** which is bad for GPU cache and texture-prefetch. Approximately 1% - 5% overhead could be introduced.
- Hence our pixel shader has relatively more instructions. However since UV tiling saves more than 30% of primitives while a significant number of building pixels are occluded by foliage, **we consider it a gain**.

Perspective Interior Mapping



- We optimized Interior Mapping Shader instructions and our solution **only needs 1 Box – Ray Intersection** calculation.
- Perspective Projection Transformation is simulated in **Object Space**.
- We use one single Perspective 2D Texture for five walls rather than using a cubemap, which further **improves memory and texture sampling performance** on mobile platforms.

Extended Perspective Interior Mapping

- We did an extra plane-ray intersection to support the **middle layer**, enhancing the feeling of perspective and parallax.
- Finally, we implemented a front layer based solely on Vertex UV, yielding effects such as **glass, decals, and curtains**.


That’s enough for exteriors.
But what about
Interiors?

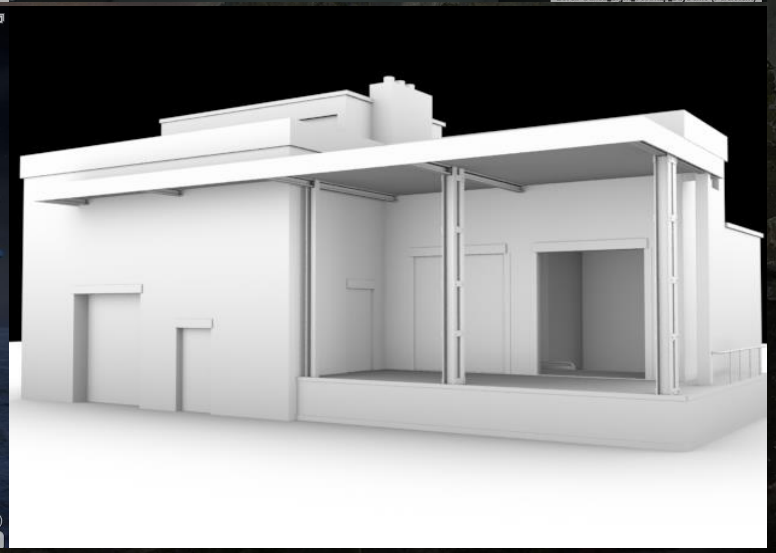
Challenges for Interior Baking

1. Rendering indoor environments presented even greater challenges.

2. UNDAWN features a dynamic **TOD** system, which means that the lighting environment changes constantly. As a result, the baking solution must be able to adapt to these changes.

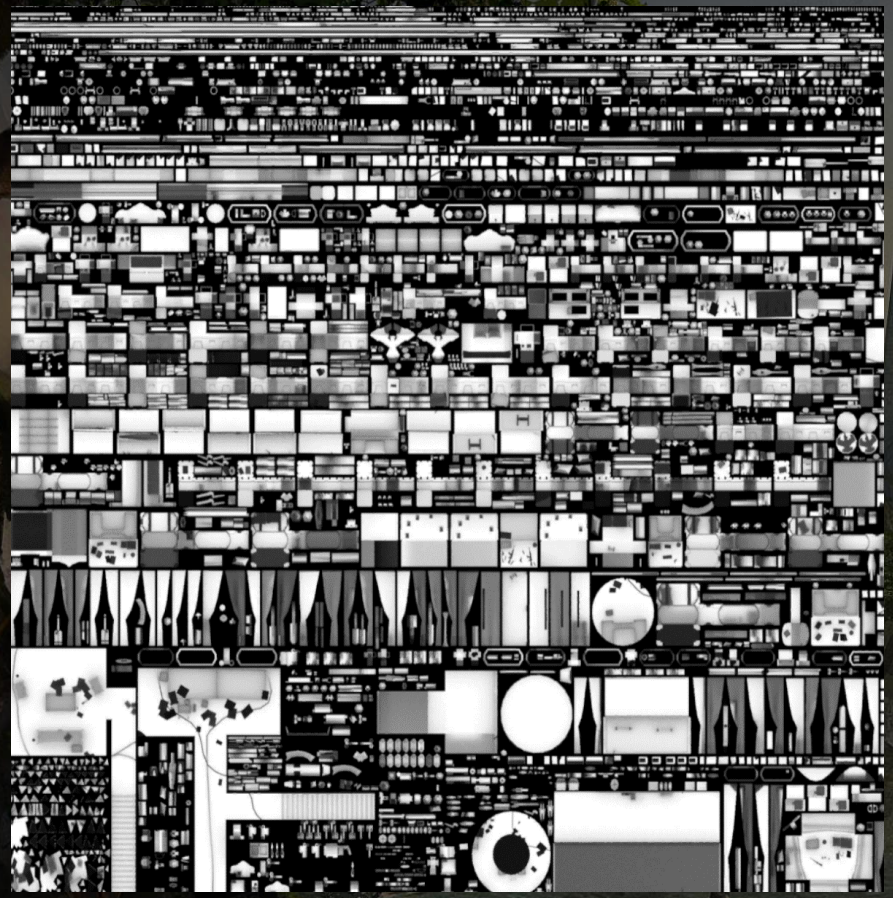
3. Using lightmaps could potentially consume a significant amount of texture memory. Therefore, we must carefully **balance visual quality and performance** to ensure optimal results.

Baking for Dynamic lighting – Dentro



- A **per-building** baking solution for optimized streaming.
- Bake **Ambient**, **Indirect**, **AO** and **artificial lighting** separately.
- Combine the greyscale of those lightmaps into **1 RGBA texture**.
- The color is then calculated in real-time based on **dynamic lighting** in the scene.

Baking for Dynamic lighting – Dentro



AO

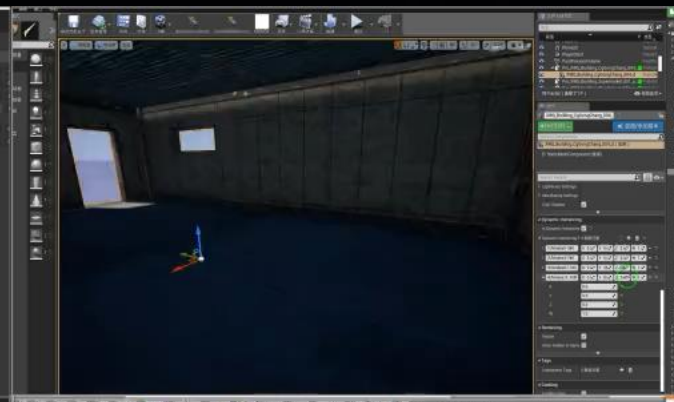
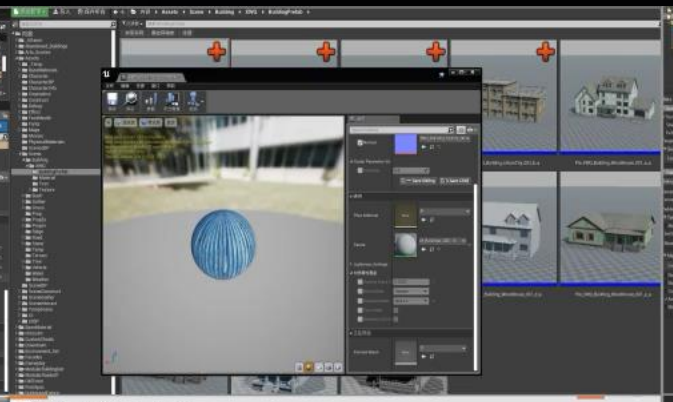
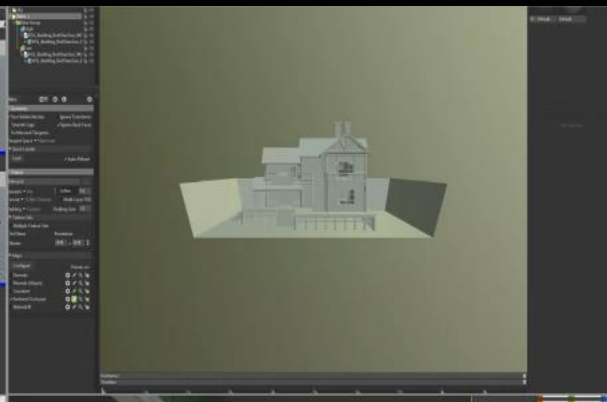
Artificial Lighting for night

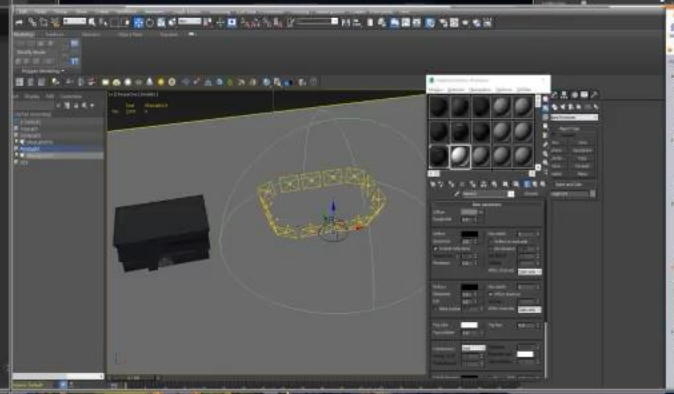
Baking for Dynamic lighting – Dentro

Indirect Lighting for Averaged Directional Light

Ambient Lighting for Skylight

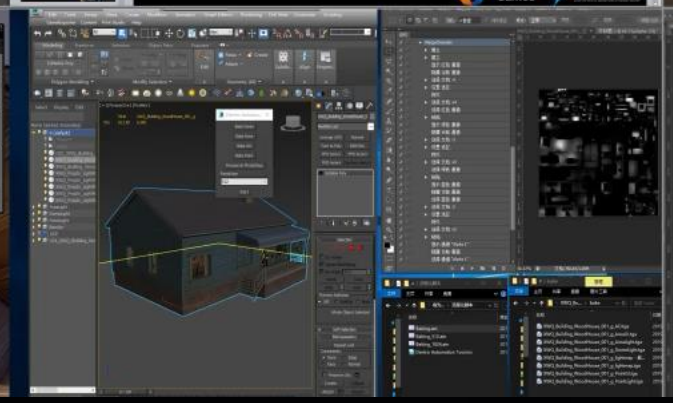
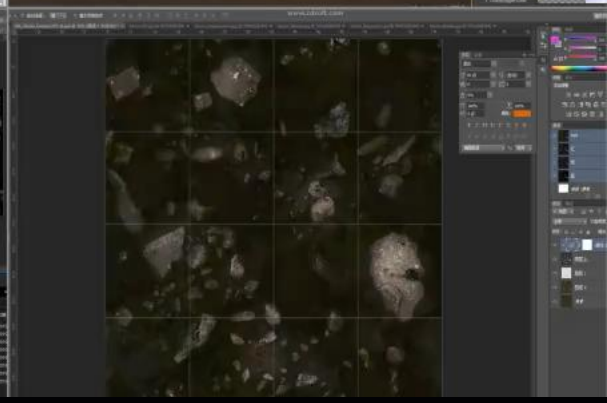

This concludes our Building
Technique, a BATTLE with
performance…

TechSmith
MADE WITH CAMTASIA FREE TRIAL


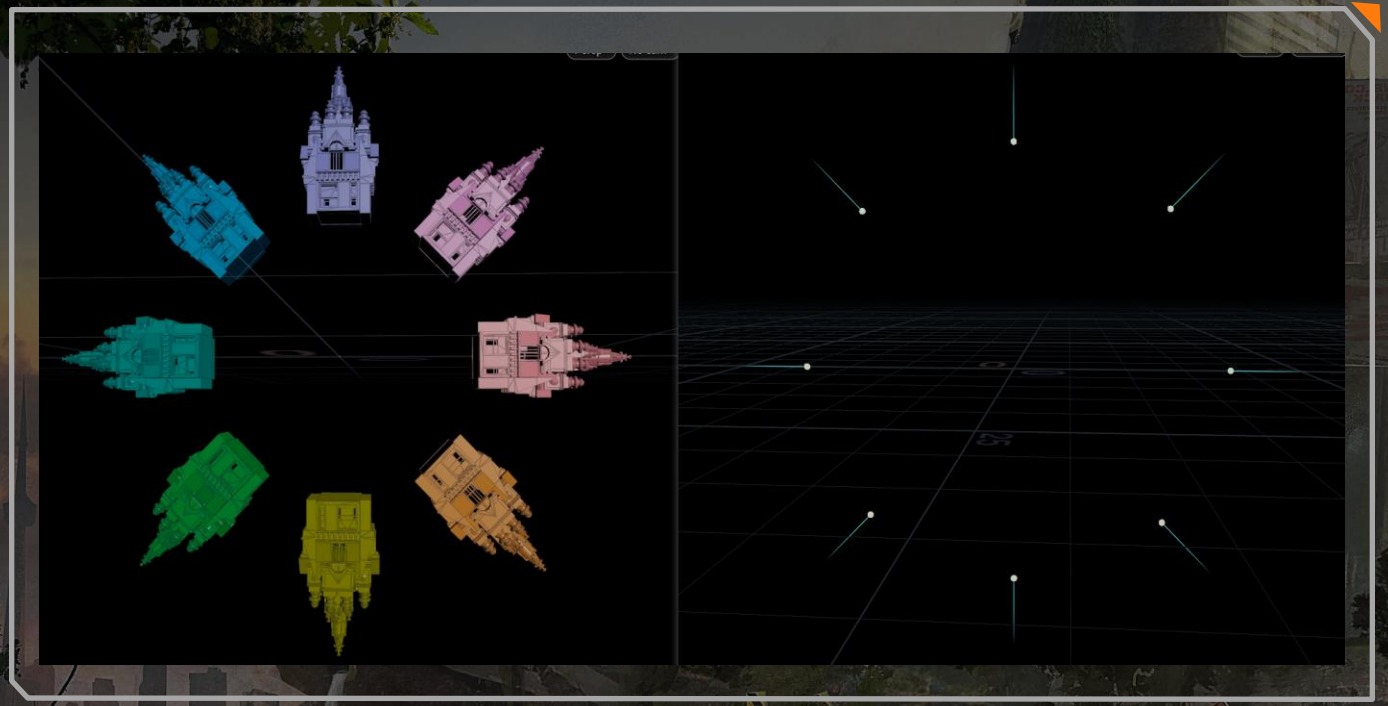
V-Ray LightMap通道分配
R: DomeLight贴图（天空光的GI光照效果）
G: AreaLight 贴图（白天室内Direct+GI光照效果）
B: AO（模型AO效果）
A: PointLight 贴图（夜晚有灯室内Direct+GI光照效果）
LightMap
Tencent Games



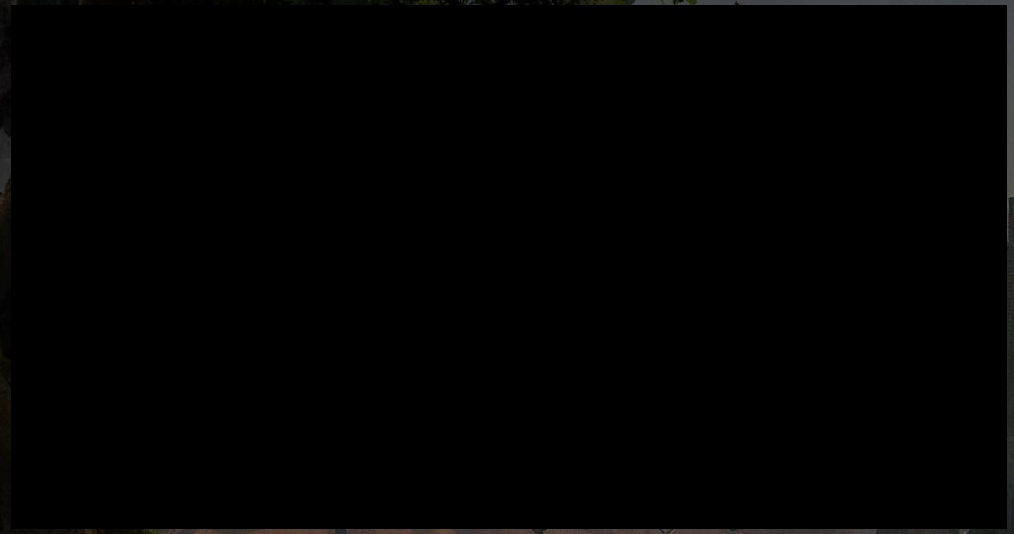
III Vertex Magics in VFX

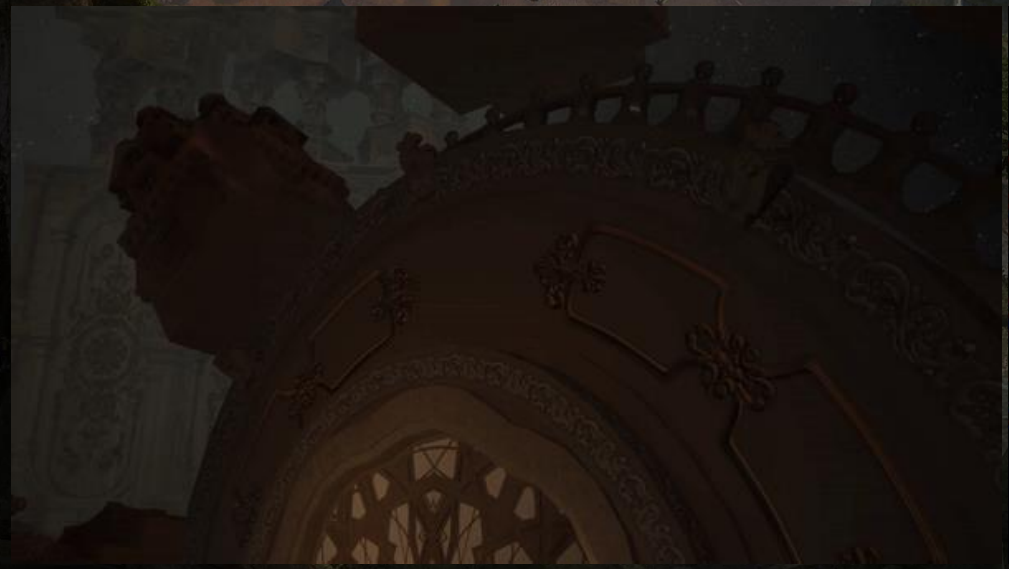
Pivot Painter for Transform



- Transforming multiple objects per frame is **heavy** on CPU for the mobile platform.
- Pre-bake all the transform data into **mesh attributes** and access that data in Vertex Shader.
- With the help of Pivot Painter, we can achieve many **unique** VFX for mobile games.

Pivot Painter for Transform

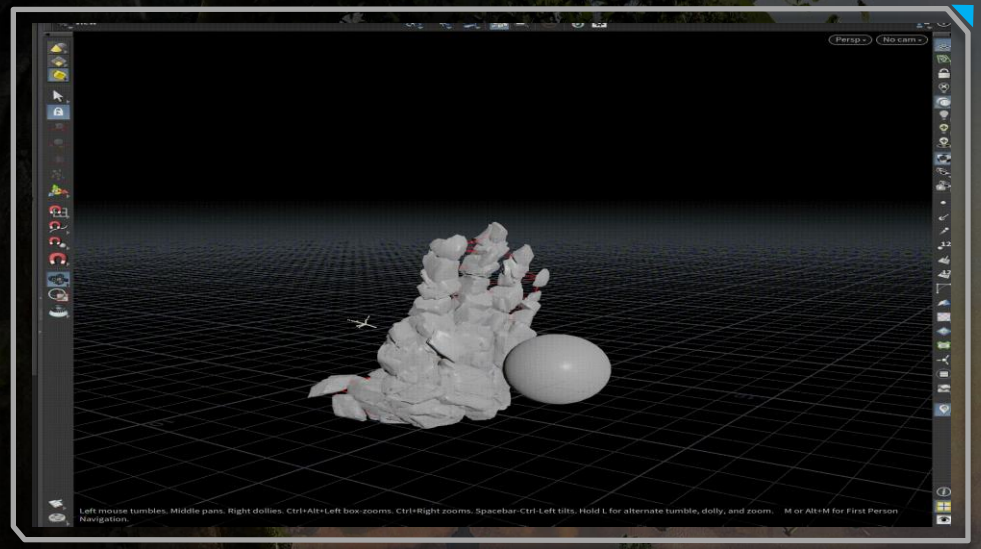


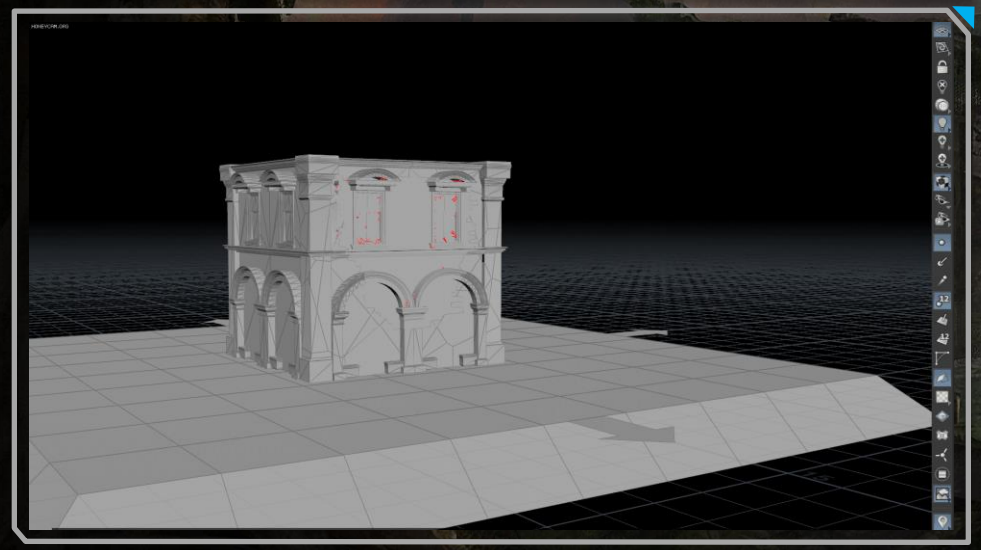


Rigid body Simulation based on VAT

- Real-time destruction physics is too heavy for mobile games.
- Simulate RBD Fracture and Rigid body physics in **Houdini**.
- Pre-bake the transform information of each fracture into **Texture**.
- Access the transform data of each frame in Vertex Shader.

Rigid body Simulation based on VAT

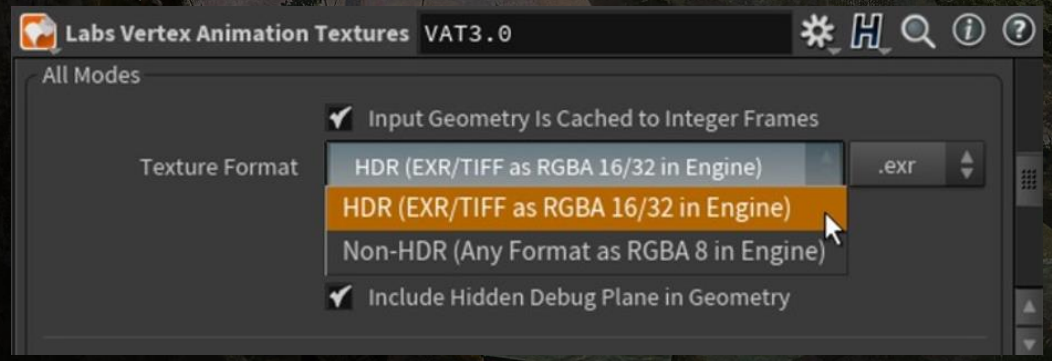


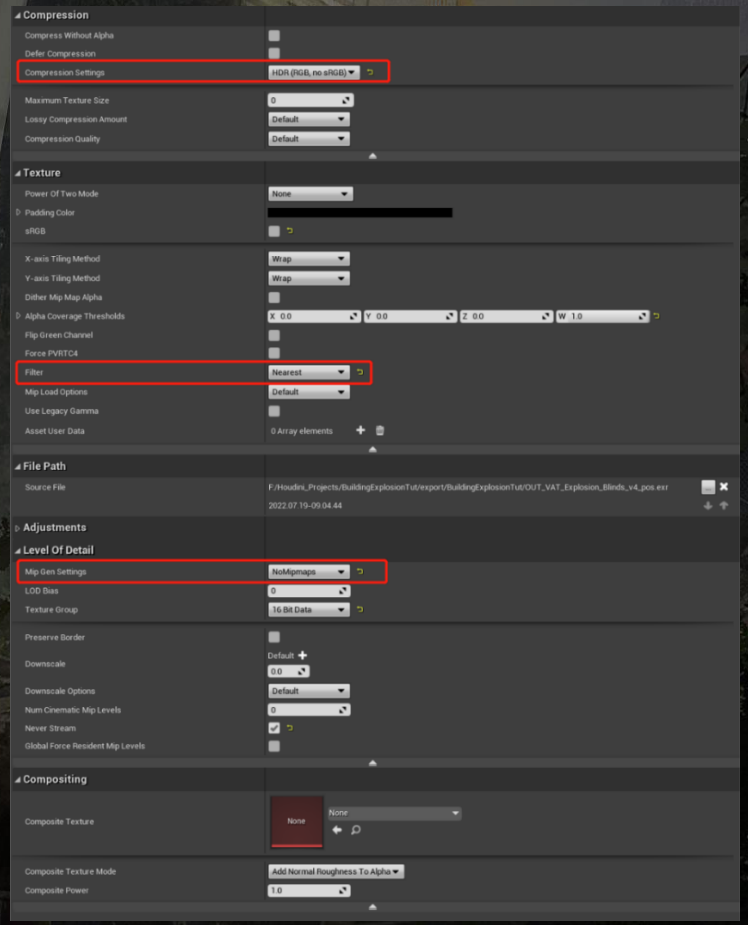


- We used **Houdini** for Rigid body simulation.
- Houdini provides multiple fracture types and dynamic solvers to meet our need.
- Consider each fracture as an atomic unit, store the pivot of fracture into vertices attributes.
- Write each fracture's transform information into the texture frame by frame.

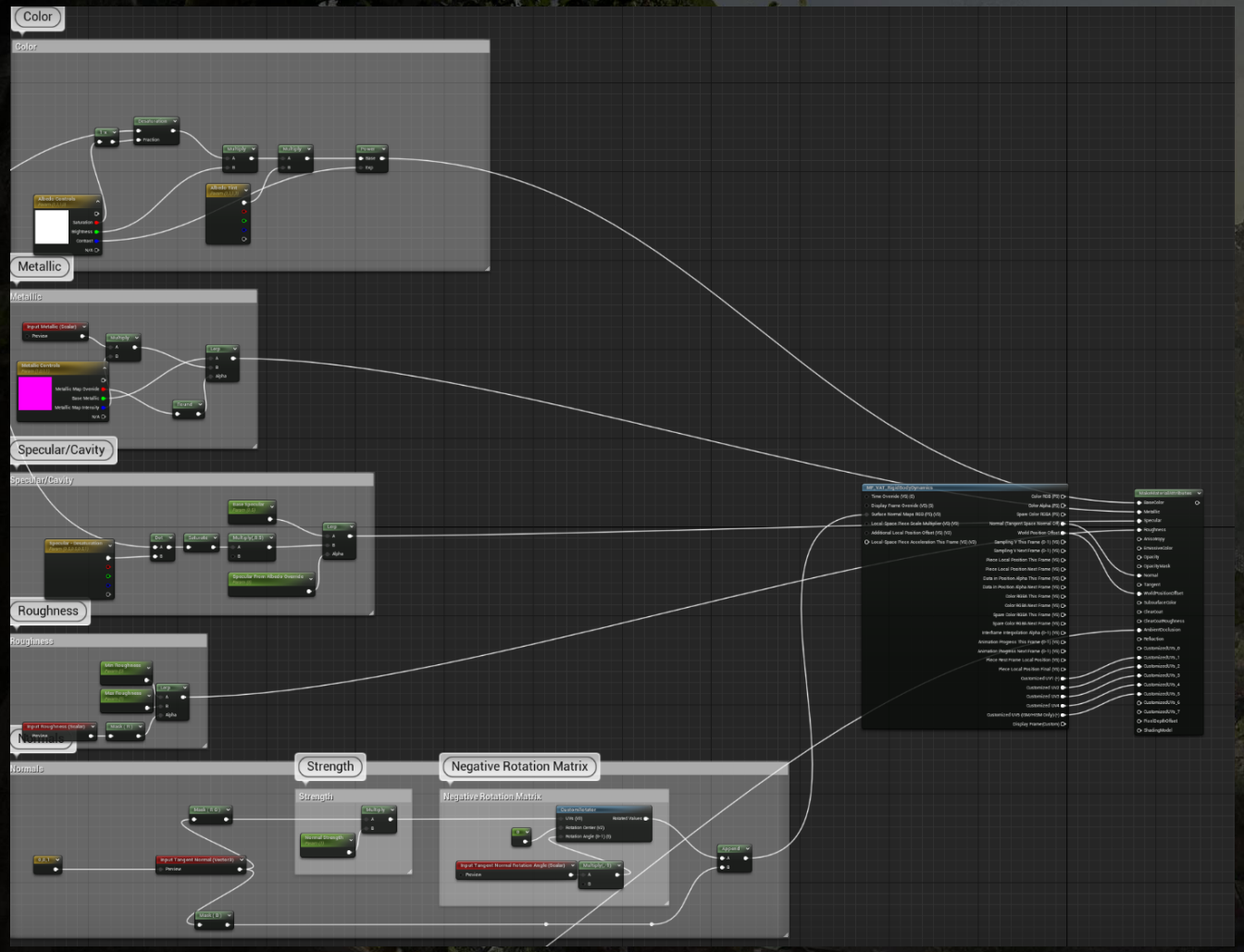
What it looks like in Unreal Engine

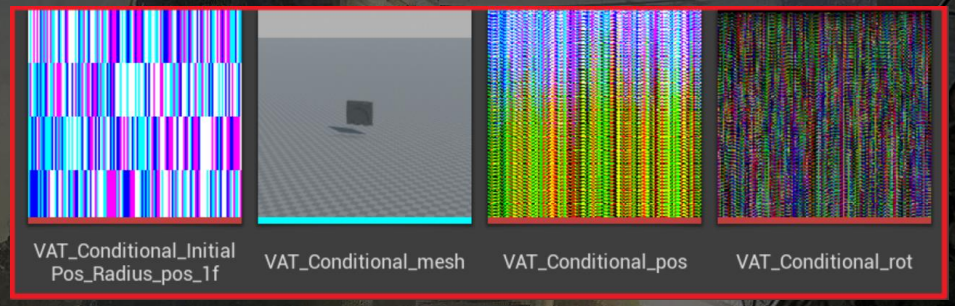
Based on

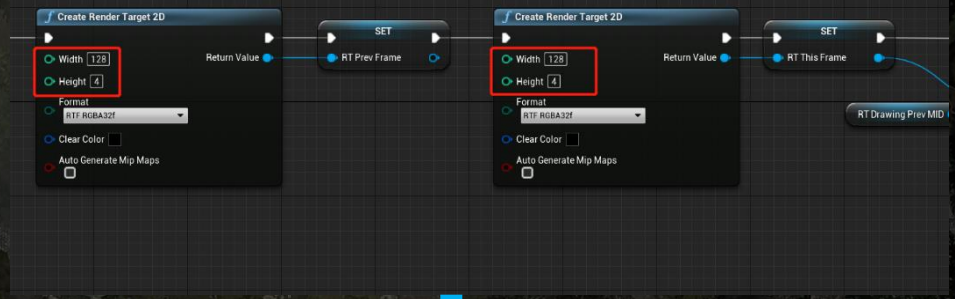




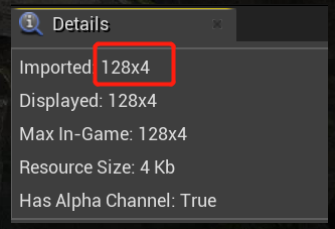
What it looks like in Unreal Engine



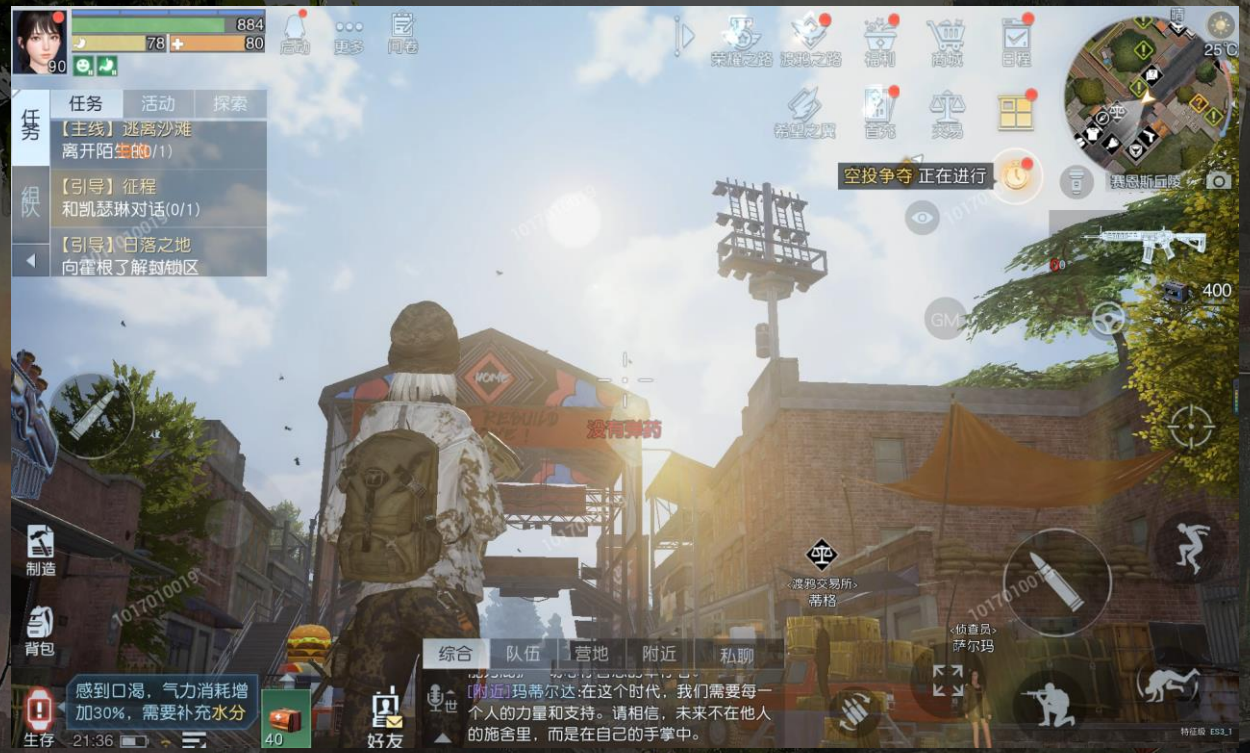




Based on

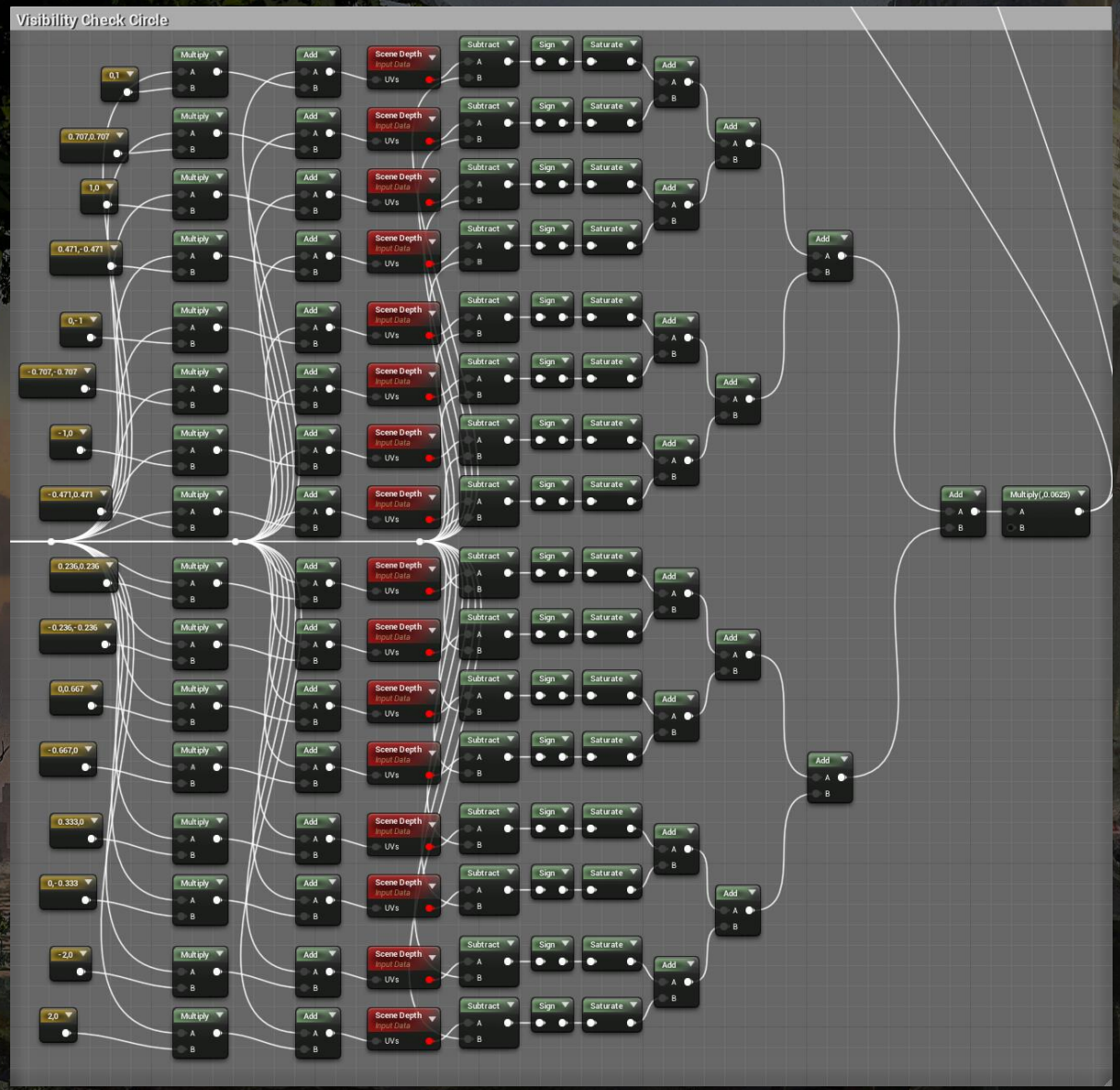


Lens Flare



- Real-time ray-object detection is expensive for effects which do not involve gameplay.
- Ray-object detection on CPU doesn’t work on **alpha test foliage**.
- We used a depth buffer for visibility checks and moved the transformations of the flares into the Vertex Shader, while the parameters were stored in the Instance Buffer.
- This enabled us to achieve sophisticated Lens Flares with only a few draw calls.


Clip Space Billboard Vertex Position - Still suffering from Mysterious Billboard Scale. But works ...
Absolute World Position
Input Data
Subtract
Object Position
Input Data
TransformVector
World Space to Local Space
Multiply
-1,1,1
Object Position
Input Data
WorldToClip
In
缩放
Multiply
Mask (R G)
Mask (B)
Mask (A)
Mask (R G)
GetScreenUV
InClip
Custom
In
Add
Multiply
Additional Movement
Add
XYZ
Append
Multiply
xyzw
Append
ClipToWorld
InPos
Object Position
Input Data
Add
Subtract
Absolute World Position
Input Data


Visibility Check Circle
Scene Depth
Input Data
UVs
Multiply
Add
Subtract
Sign
Saturate
0,1
0.707,0.707
1,0
0.471,-0.471
0,-1
-0.707,-0.707
-1,0
-0.471,0.471
0.236,0.236
-0.236,-0.236
0,0.667
-0.667,0
0.333,0
0,-0.333
-2,0
2,0
Multiply(,0.0625)

Our Team

Lhil Yang

- Current Lead TA of UNDAWN.
- Houdini Simulation, Pipeline & VFX.

Xurong Ge

- Character & VFX TA.

Zhenhao Zhang

- Animation & Rigging.
- Lead TD.

Kenny Liu

- Environment & VFX TA.

Our Team

LIGHTSPEED STUDIOS | GDC

THANKS

March 20-24, 2023 | San Francisco, CA

Website: https://www.lightspeed-studios.com/

Facebook: LightSpeedStudiosGames

Twitter: LIGHTSPEED STUDIOS

Youtube: LIGHTSPEED STUDIOS

Open Position:

- Technical Art Director
- Senior Technical Animator
- Senior Rigging TA
- Procedural Technical Artist
- Senior Technical Artist

Location:

United States, Canada, France, Japan, South Korea, New Zealand, United Kingdom, Singapore and United Arab Emirates.

If you are interested, please contact: rubylei@lightspeed-studios.com

Real Time Irradiance Probes

An Indirect Lighting Approach in UNDAWN

Leon Wei

Engine Team Lead, LIGHTSPEED STUDIOS

March 20-24, 2023 | San Francisco, CA

Big World UNDAWN Dynamic Crafting

Indirect Lighting
(for mobile platforms)

Survival - Open World
- Crafting

UE4.21

- Android - Mali T860 / Adreno 505 / RAM 3G +
- IOS - IPhone 6S+
Windows

Quality

Disk Storage

Preprocessing Time

Artist Work Flow

Dynamic World

Dynamic Weather

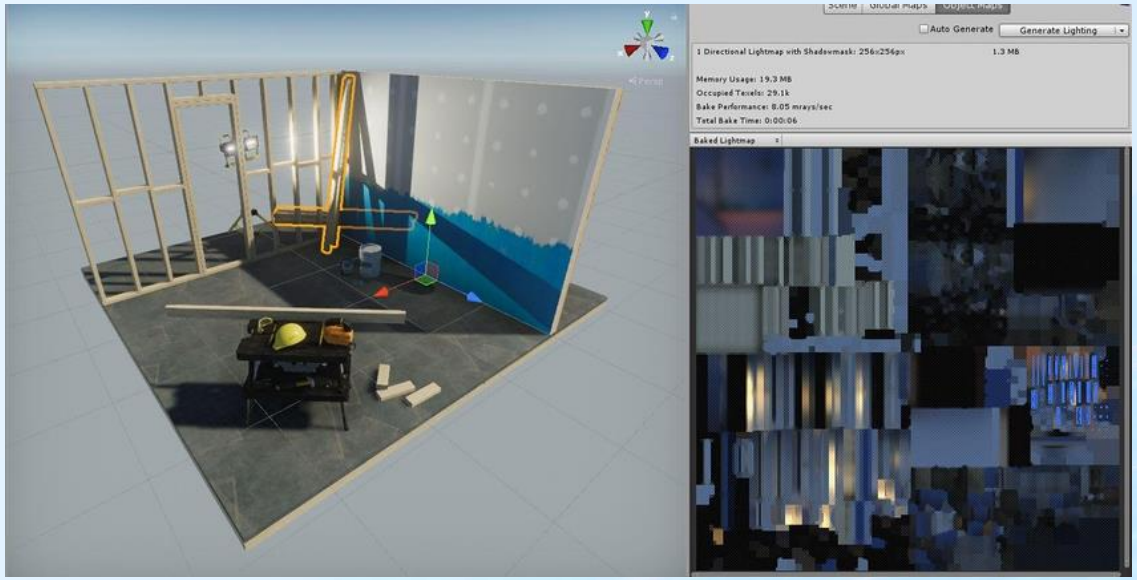
Current Solutions

Baking Light-map/PRT SH

- Store the Irradiance or PRT SH in textures off-line

Good quality (full details, specular, AO)

Huge storage (Big world)	Only support static objects (Dynamic crafting)



limitedly used in the game(inner buildings…)

Current Solutions

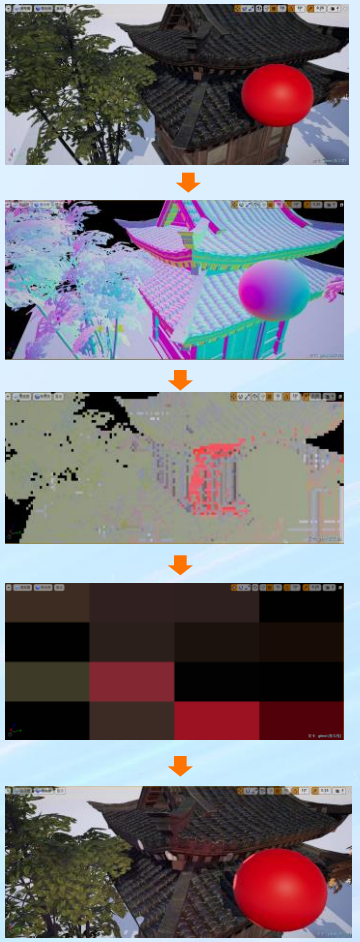
Screen Space

- A pass of image post-processing
- Our team have implemented the “VPL based SSGI” (Presented in UOD ShangHai 2021)

Real-time

Not accurate (only few VPLs)

Not efficient on low-end



Current Solutions

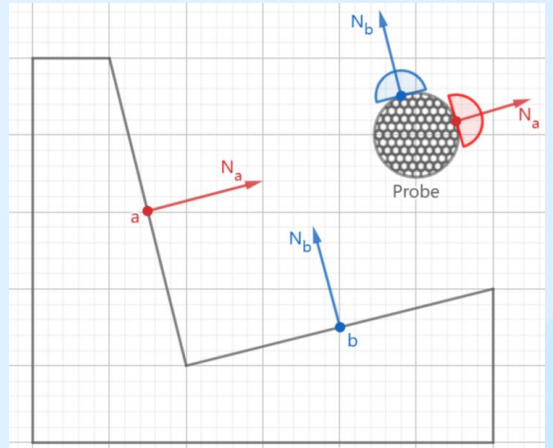
Probe

- Store the irradiance (always SH) in discrete space proxy points

Fewer storage

Limited probe quantity

Storage and calculation work is limited for big world



Details lost, no specular or AO

Assisted with SSAO, IBL…

- Always pre-calculated in today's solutions for runtime efficiency.

We tried the realtime probe approach !

Realtime Probe Generation

Smooth Frame

- Limit the number
  - Key position
  - Around the player
- Generate Progressively
  - Near to Far

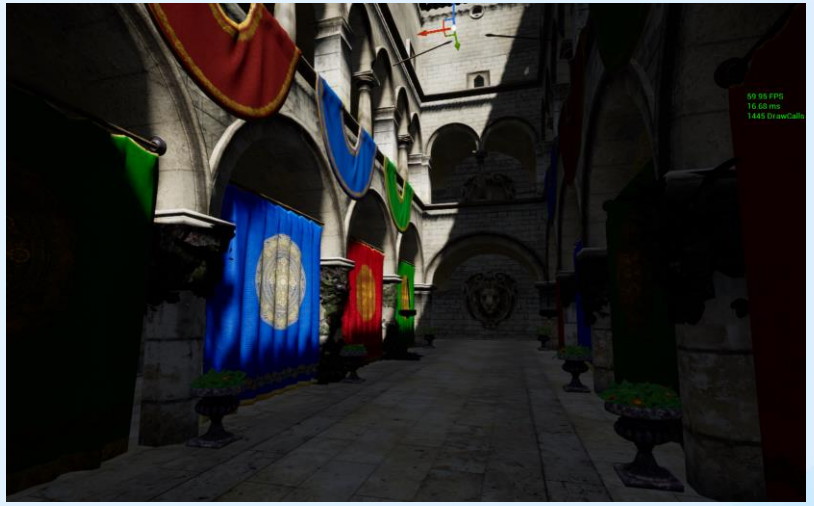
Realtime Probe Generation

Generation Efficiency

- We capture the scene at each probe position in each direction for indirect sampling.

Huge draw calls at each probe view direction !

- Replace the scene with simple boxes
  - Full GPU pipeline can be utilized, 1dc for 1 probe direction!
  - Mesh shape is not important



Original Scene(1445 draw calls)

Scene With Simple Boxes (1 instancing draw call)

Realtime Probe Generation

Lighting Quality

Probe Lighting Quality is Always Poor !

- 3rd level SH just “average” the indirect color
- Probe lighting seems just “brighten the scene”

“Large flat color boards” are used

- Only Prominent mesh color is kept
- Not physically right, but artists like the tool.

Others

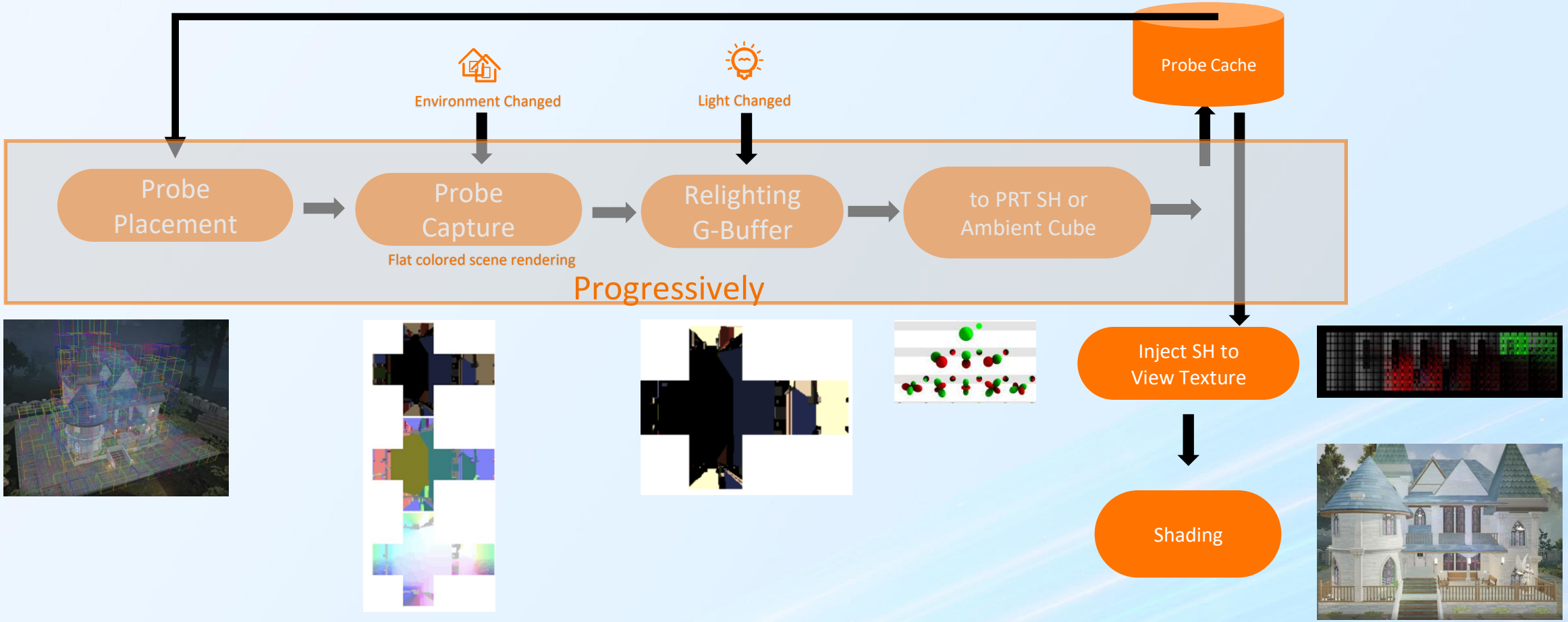
- Anti light leak
- AO
- Multiple bounce …

Our Indirect Lighting Solution

Progressively Irradiance Probe from Flat Colored Boxes

- Probe Placement and Reuse (Automatic Position, Cache)
- Flat Colored Boxes Scene Rendering (Mesh Preprocessing, GPU Pipeline)
- Probe SH Calculation (with PRT)
- Probe Lighting (Ambient Cube, Anti Light Leak, Sky AO, Multi Bounces)

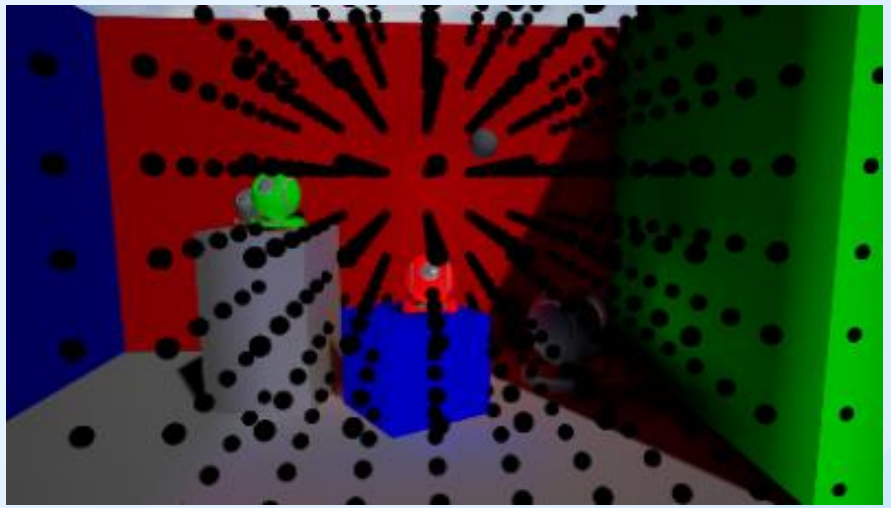
Framework Overview



Probe Placement

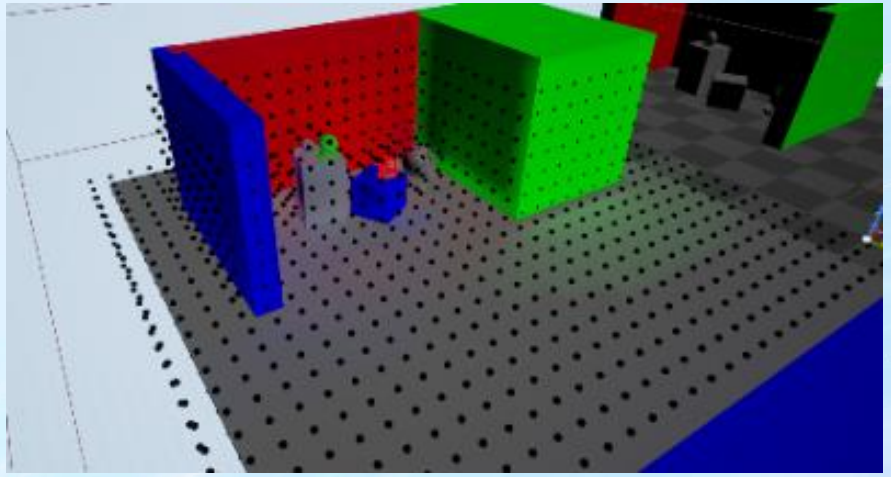
- Place around the player automatically from near to far

Uniform Distribution



- Probes in sparsely populated areas waste memory.
- Invalid positions: inside the wall.

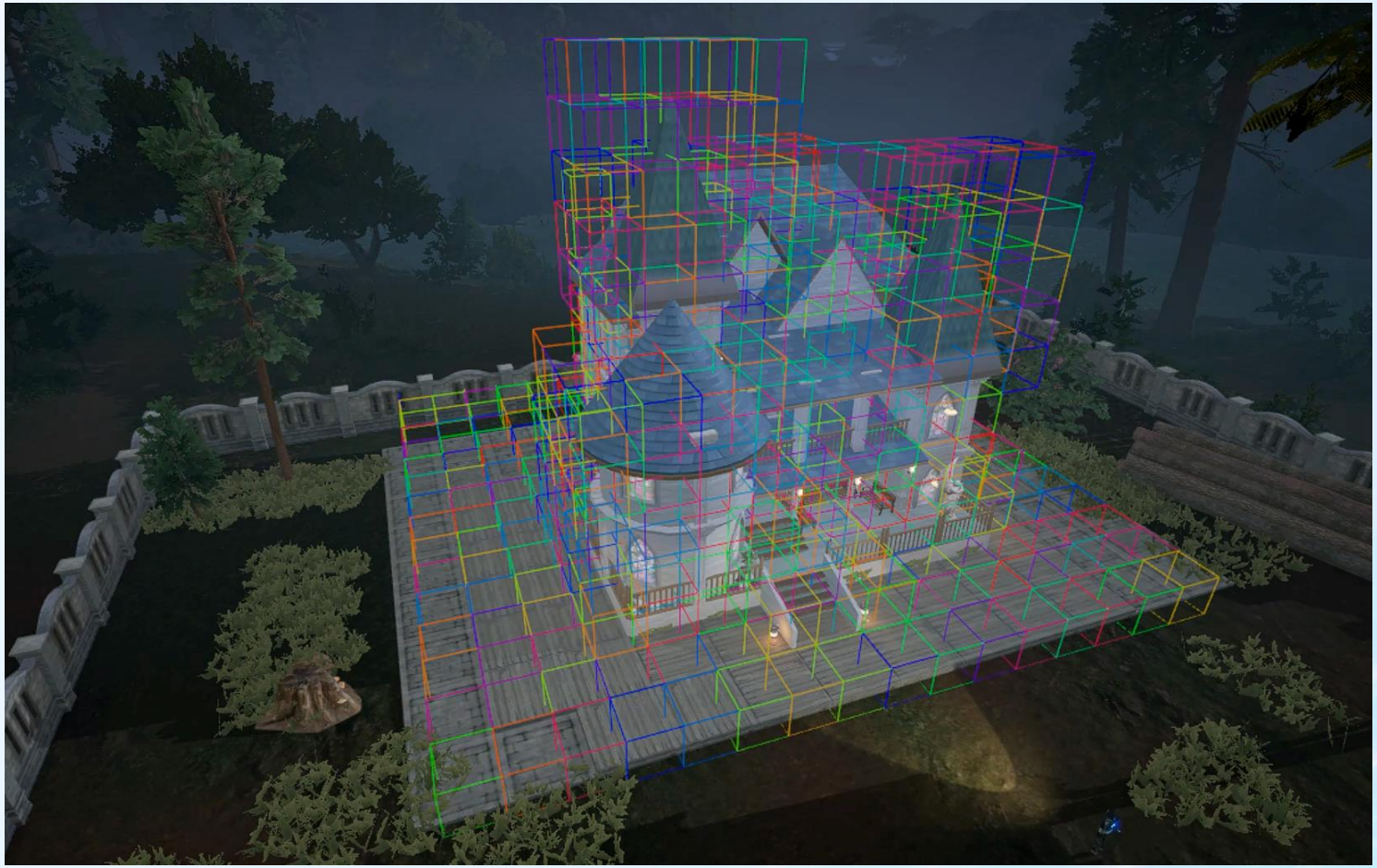
Sparse Distribution



- Oct-tree is used to culling the sparse area
- Test ray intersect to find the valid positions

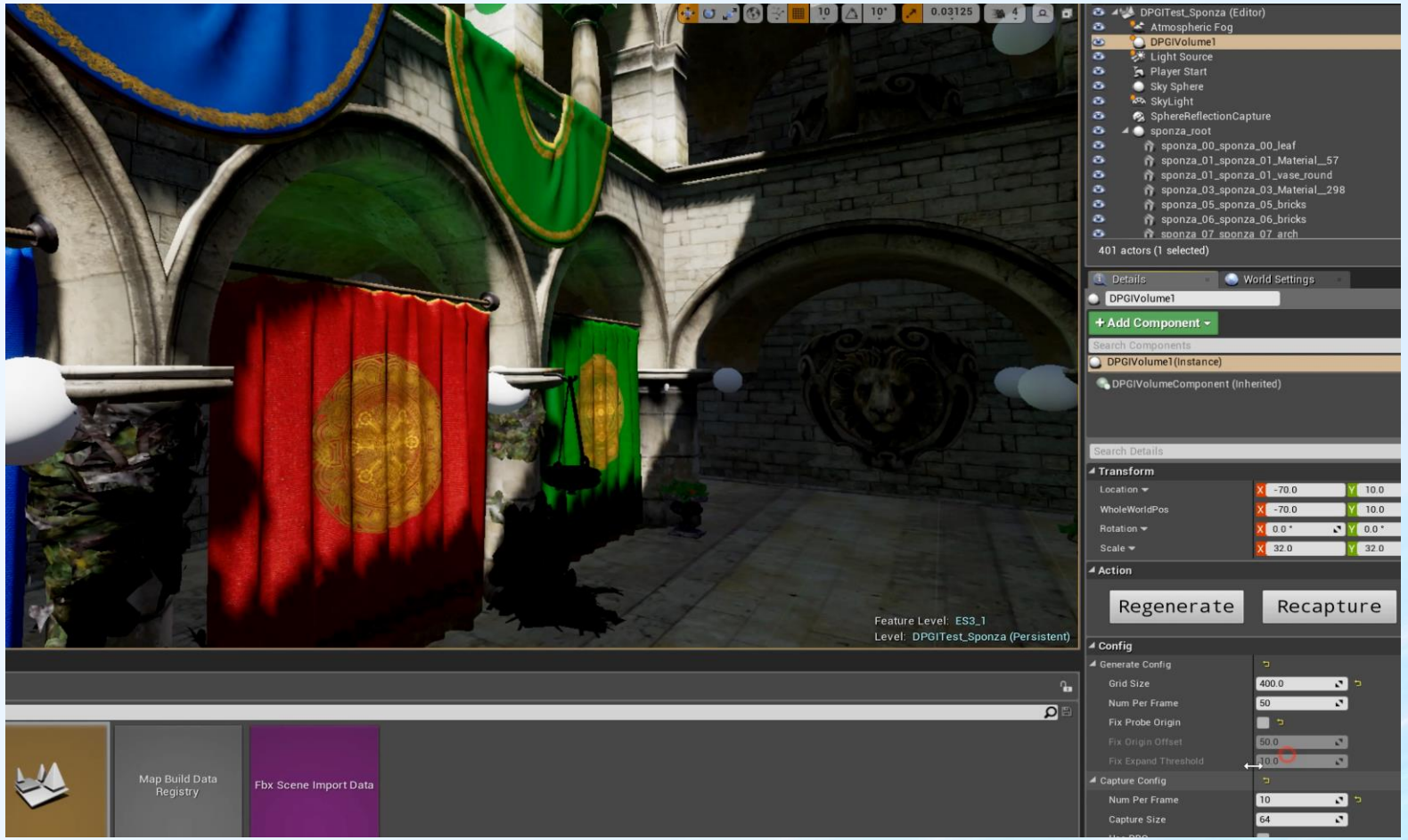
Probe Placement

Oct-tree is used to culling the sparse area

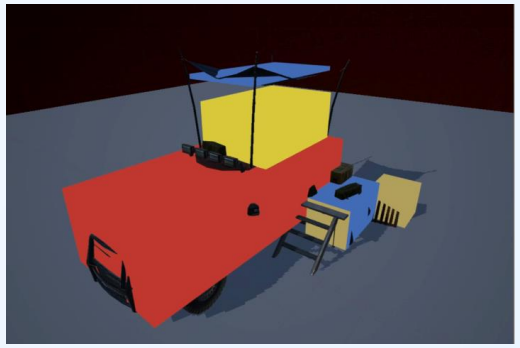


Probe Placement

Ray intersect test is used to find the valid positions

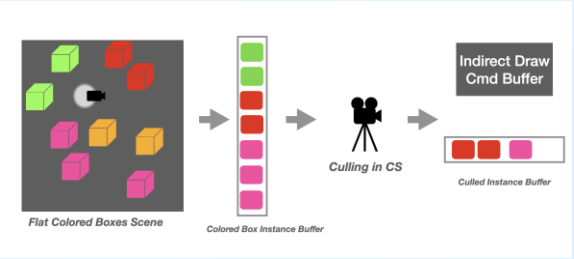


Probe Capture (Flat Colored Boxes Scene Rendering)

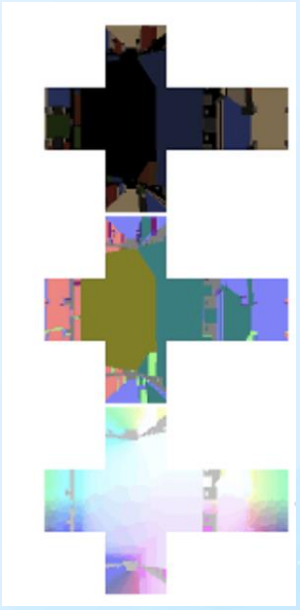


Preprocess mesh to a flat-colored box.

Replace the scene with a flat-colored box.



Start a GPU-pipeline to render flat-colored boxes.

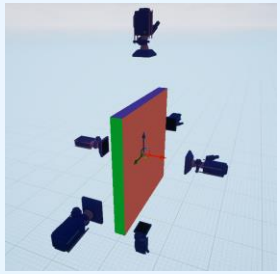


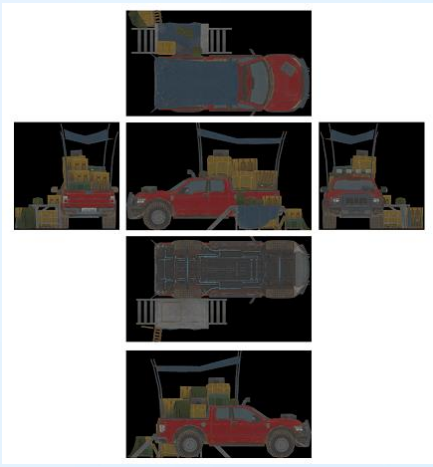
Rendered results

Offline Mesh Preprocessing

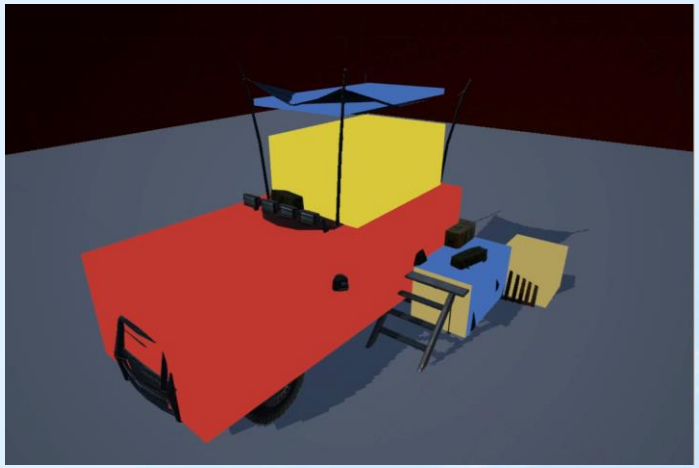
- Automatically processed when a new resource is imported to the workspace.
- Artists can also assign the face color manually

Original Mesh





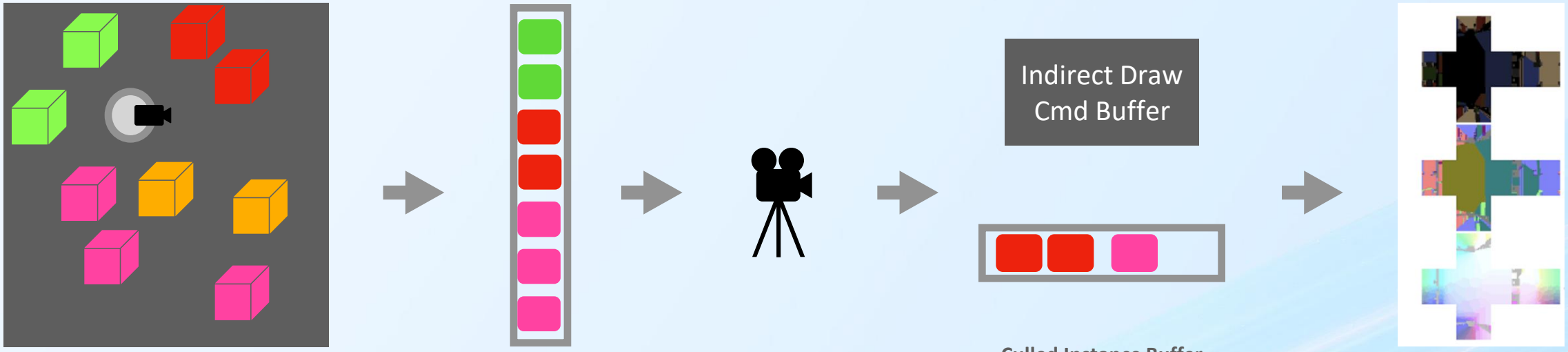
OBB calculation
Capture each face



Calculate the “main color”

Gpu Pipeline Rending

1 draw call for 1 probe cube map face



Probe Capture Rendering Results

- G-Buffer is captured for future relighting purposes.

Progressively

- 64-128 cube map face (dc) / frame

Optimizations

- Very low resolution (64-128)
- Very limited instance (significant objects)

Albedo | Position | Normal

Sampling G-Buffer to Generate SH

Why SH (Spherical Harmonious Function Basis)?

(Peter-Pike Sloan, Jan Kautz, and John Snyder. 2002. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments.)

Surface normal p

SH is rotation invariant

view directionW0

Ignored as for Lambertion

Each incoming Indirect direction Wi

$$L(p,\omega_0) = \int_S L(p,\omega_i)\cdot T(p,\omega_i,\omega_0)\cdot \mathrm{d}\omega_i$$

From Captured G-Buffer

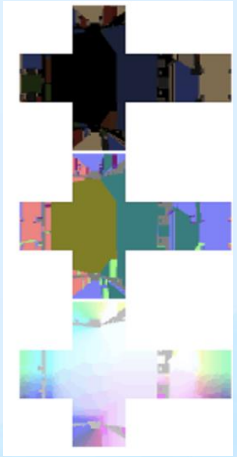
Indirect Lighting Samples

BRDF

Integration of function products
= Product of SH basis

$$= \sum_{k=0}^{k<n} l_k \cdot t_k$$

Lk & Tk are SH basis weights of L() & T()

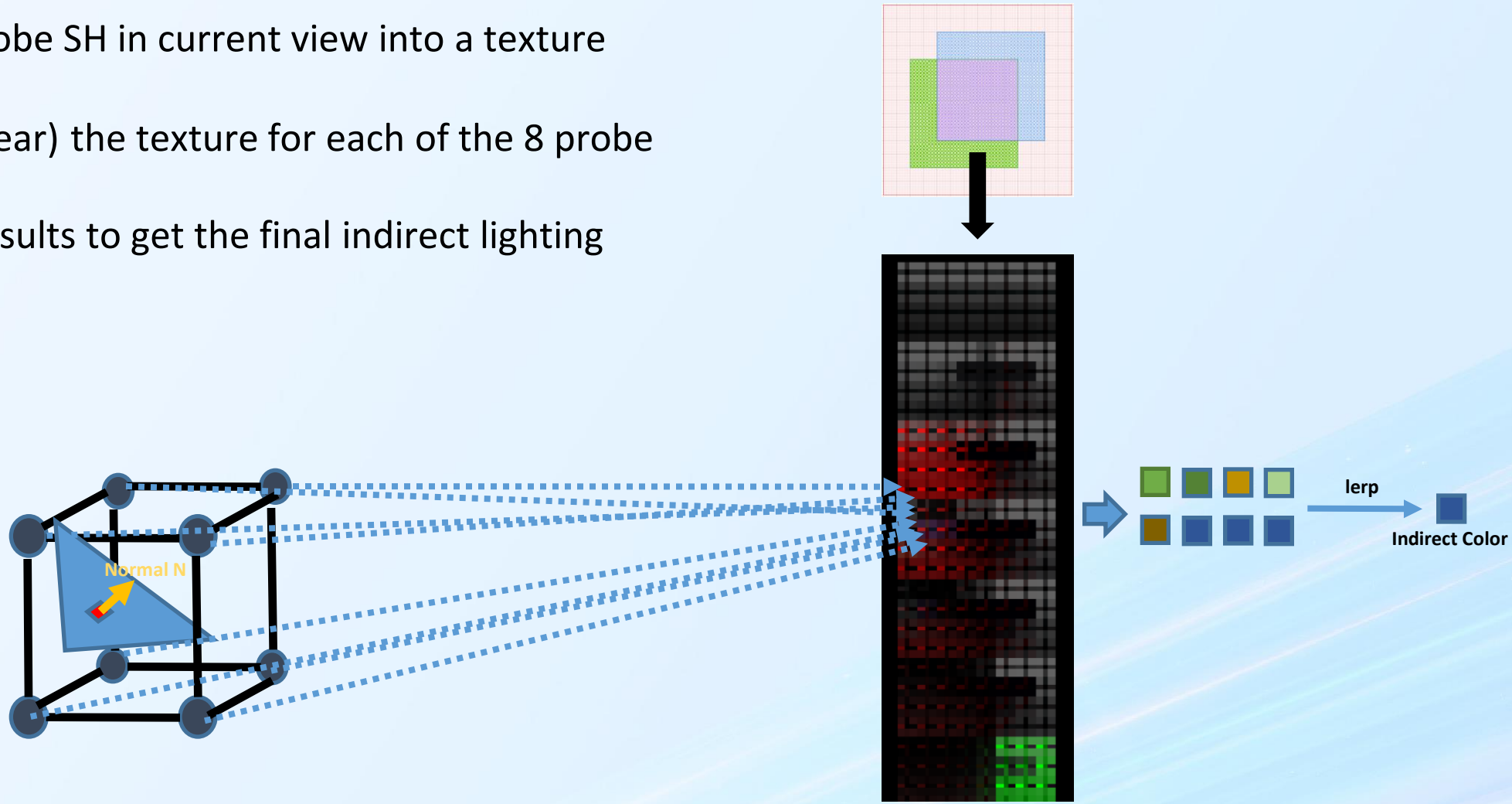


Monte-Carlo Integration

- Tk is a fixed value for Lambertion
- The only problem is to calculate Lk
  - Monte-Carlo Integration From G-buffer samples
  - One direction

Shading with Probes

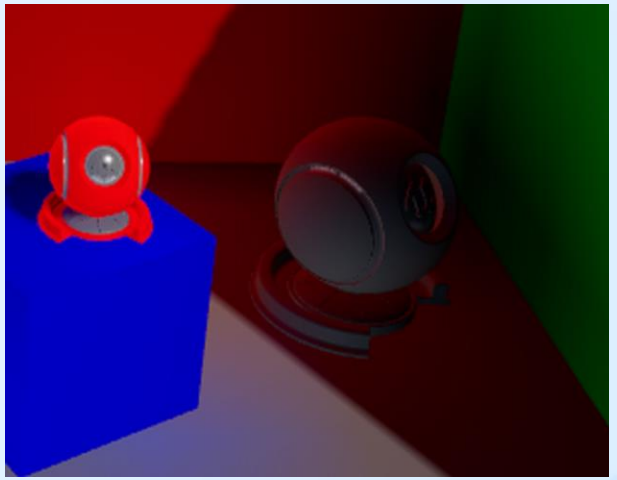
- Inject the probe SH in current view into a texture
- Sample(bilinear) the texture for each of the 8 probe
- Lerp the 8 results to get the final indirect lighting



Shading with Probes

PRT SH

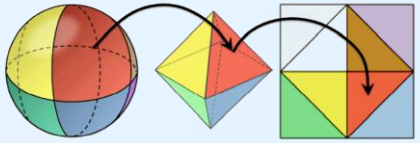
- When shading: 54 samples (3 channels × 9 bases × 8 probes)
- Volume-texture filter could not be used to prevent light leaks.
- Used on high-end devices



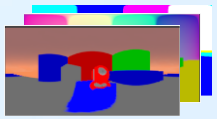
PRT SH

Ambient Cube

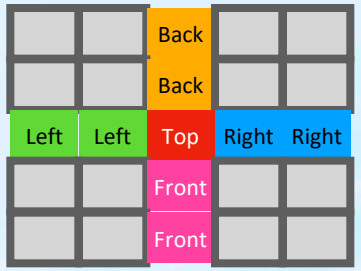
- Calculate the six-faced irradiance of a probe and store it in a 5x5 2D-texture.
- Transfer normal to oct representation map UV and bilinear sample
- 1 sample/probe
- Quality Loss
- Used on Low-end devices



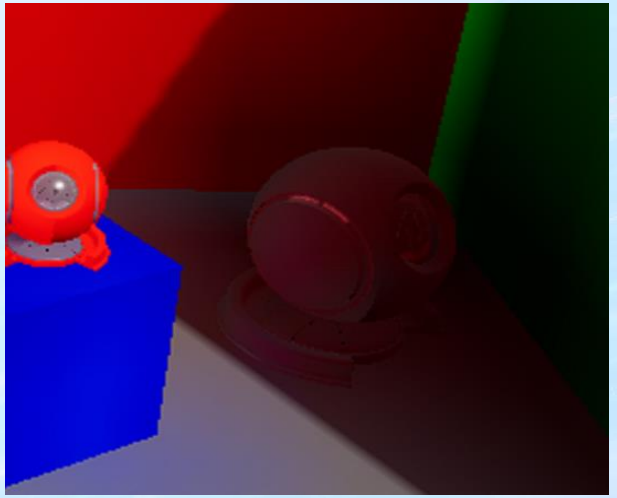
Transfer normal to 2d UV



G-Buffer Integration



5*5 texture



Ambient Cube

Probe Cache

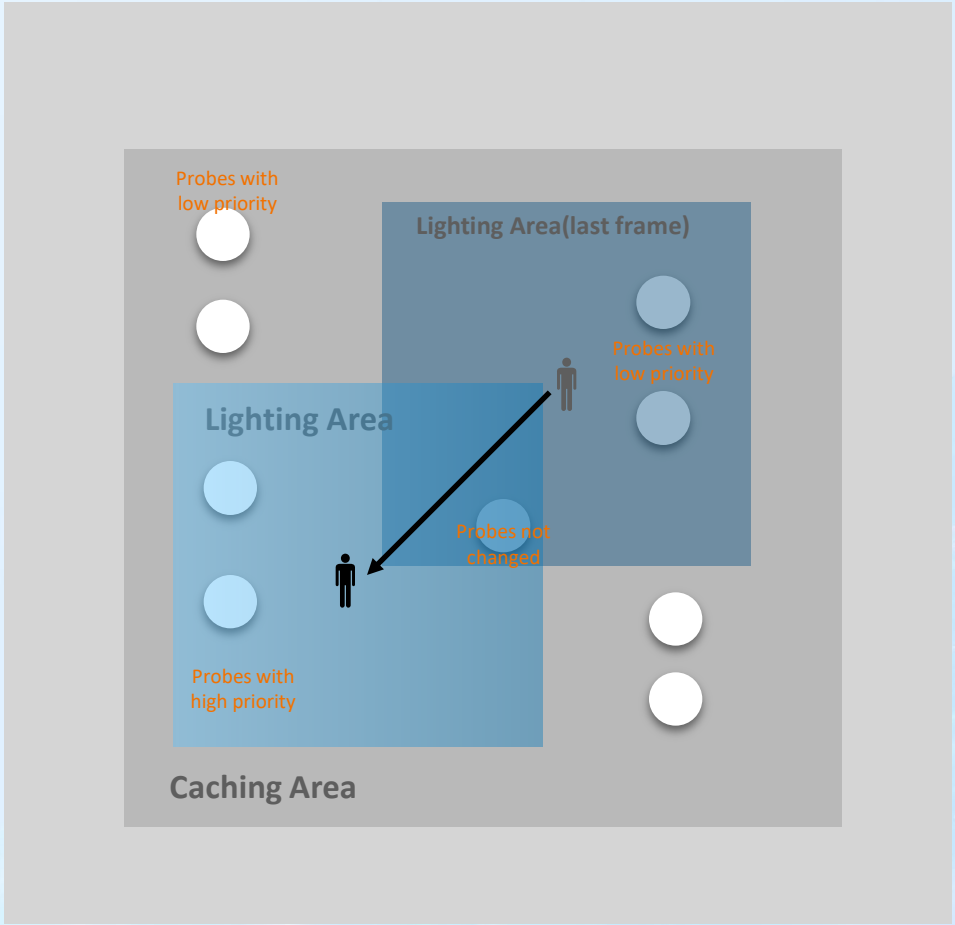
- Dynamic probe is an expensive resource, try to reuse !
  - Player wanders locally
  - Outdated probe is OK
  - Generate more probes when idle

Caching Area

- Always kept (may be outdated)
- Processed in low priority

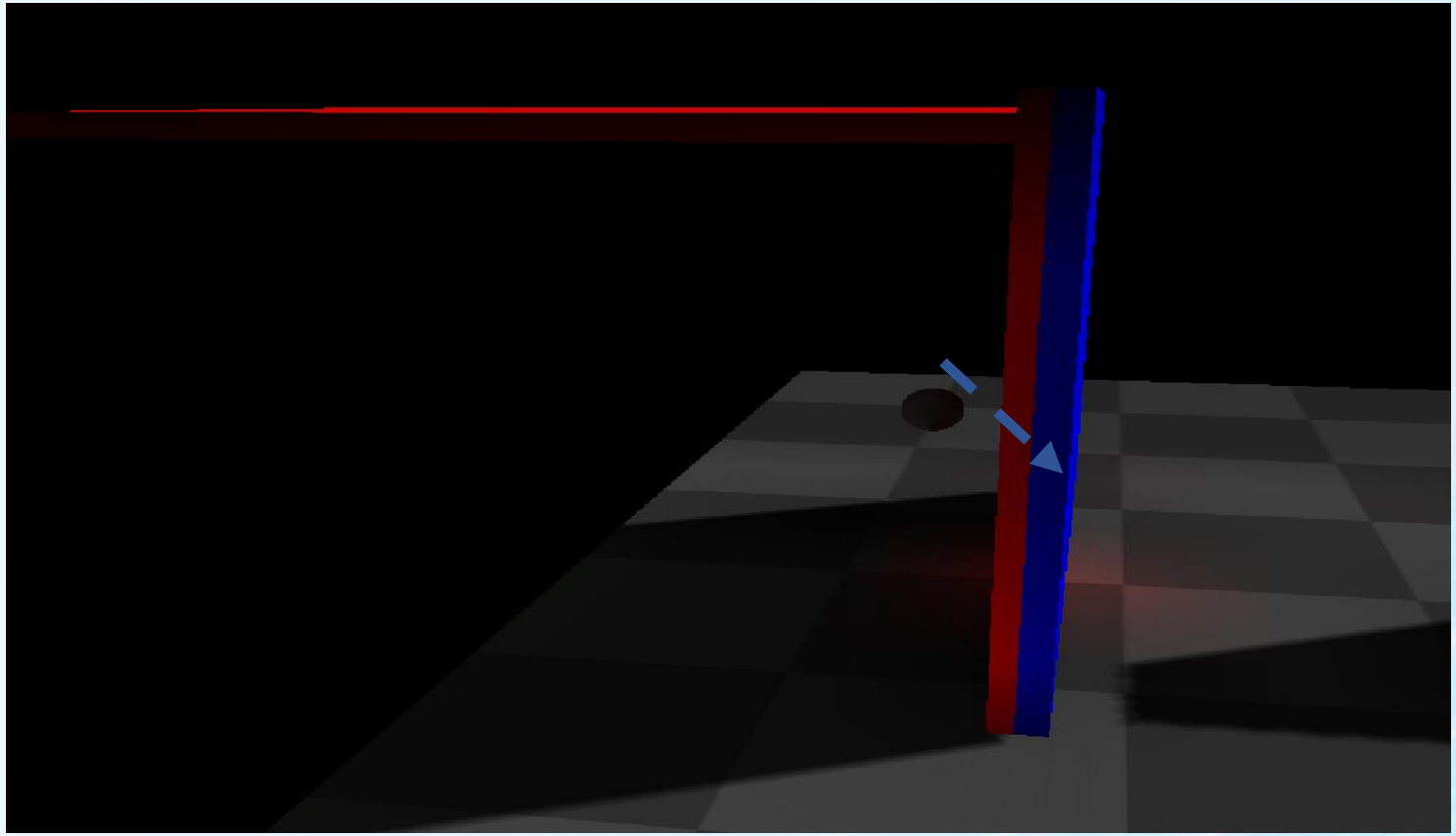
Lighting Area

- Injected to current viewport shading texture
- Processed in high priority



Anti Light Leaks

Probe lighting penetrates the space between the inside and outside.



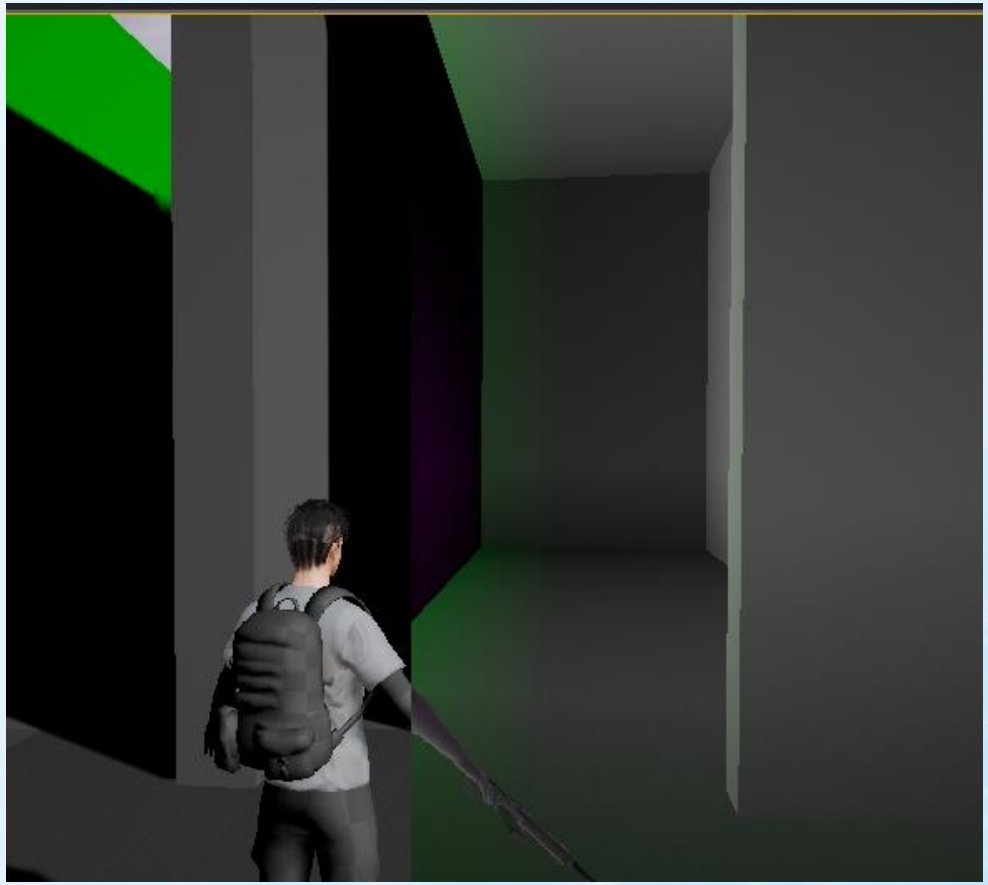
Anti Light Leaks—Chebyshev's Inequity

$$P(x > t) \leq \frac{\sigma^2}{\sigma^2 + (t - \mu)^2}$$

μ : mean

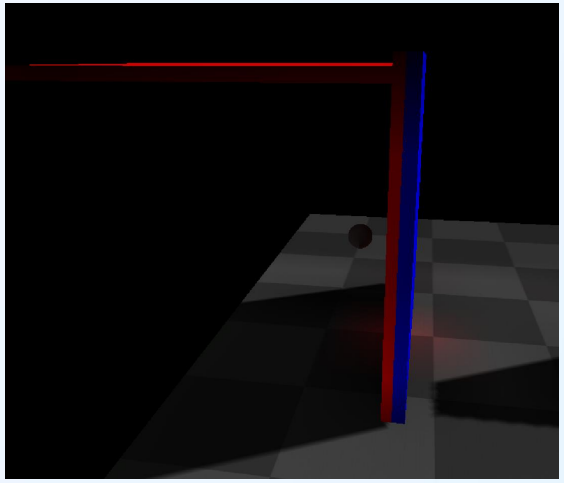
σ^2 : variance

- The far surface has more probability to be occluded
- P is the occluded probability： Capture the depth map (x) from each probe and compare with the probe-surface distance(t)
- Utilized by DDGI in UE4.27
- Still leaks somewhere

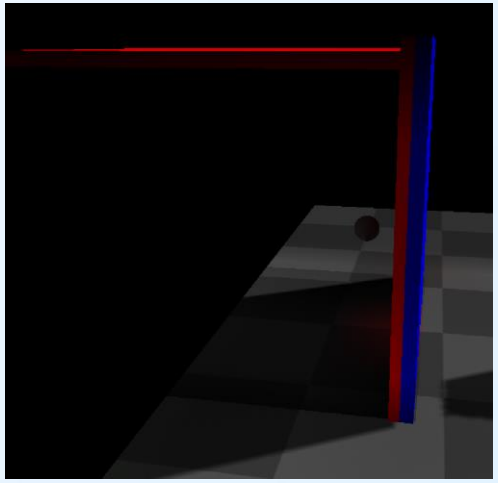


Anti Light Leaks—Probe Tag

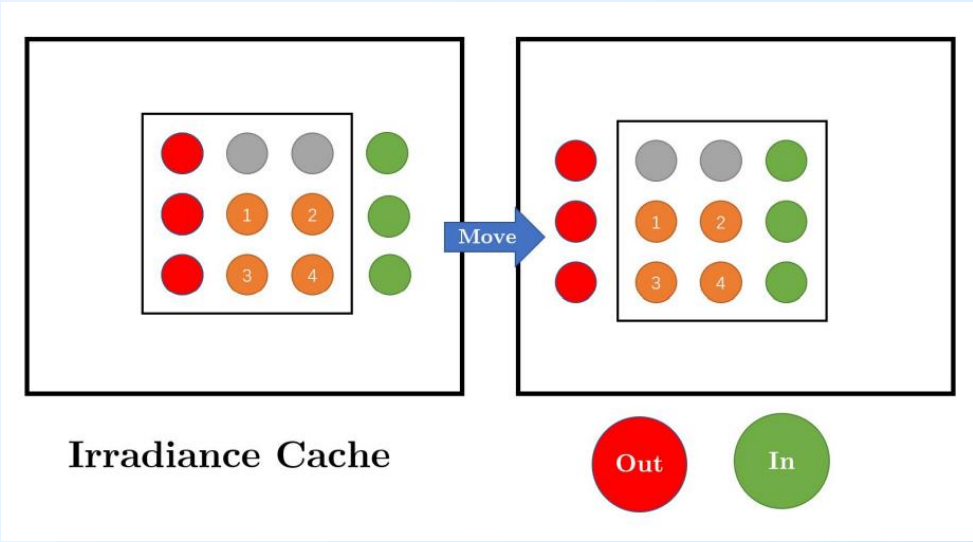
- Tag the probe with “IN” & “OUT”
- Surface inside only affected by inside probes
- Using a top view depth texture to decide “IN” & “OUT”
- Limitation: semi-open architecture



In-room probe leaks outside

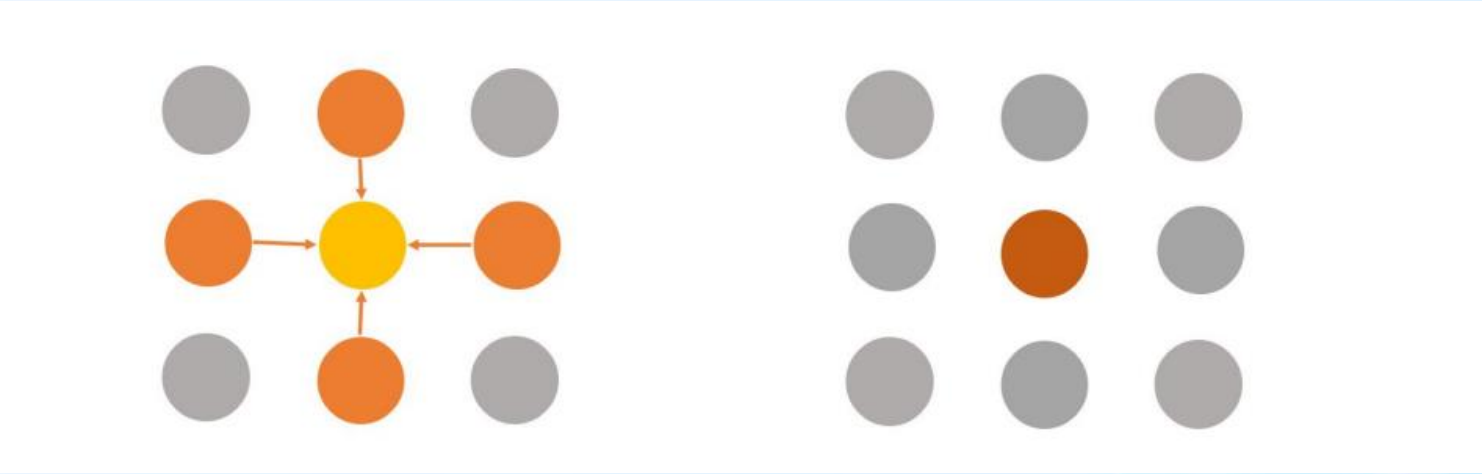


In-room probe doesn't affect outside



Multi Bounces

- Until now… only bounce for once
- Using the probes to light the probes will generate multi bounces
- Monte-Carlo integrates the surrounding 1st generation probes



1st-gen probes

2nd-gen probes

Sky AO

Record the sky visibility when capturing the G-Buffer.

$$A(\vec{n}) = \frac{1}{\pi}\int_{\Omega} v(\omega_i)\cdot(\vec{n}\cdot\omega_i)\mathrm{d}\omega_i$$

Original

+ Irradiance Probe
(lighting up the inside)

+ Irradiance Probe + Sky AO
(darken the occluded corners)

Result

Game Scenes

Probe lighting gives the "color bleeding" effect, lightening the shadowed areas.

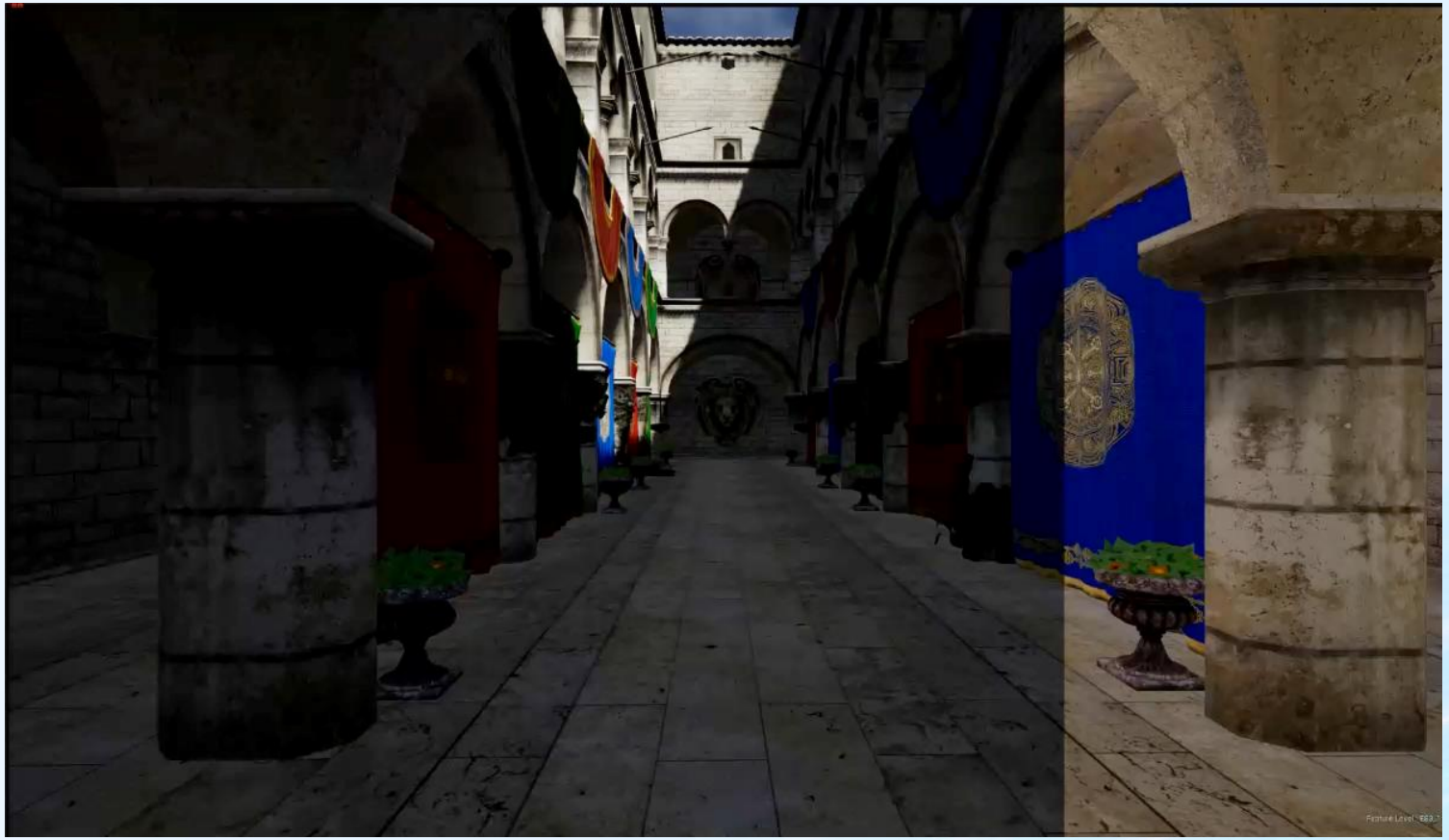
A part of a static scene

A player-created house

Result

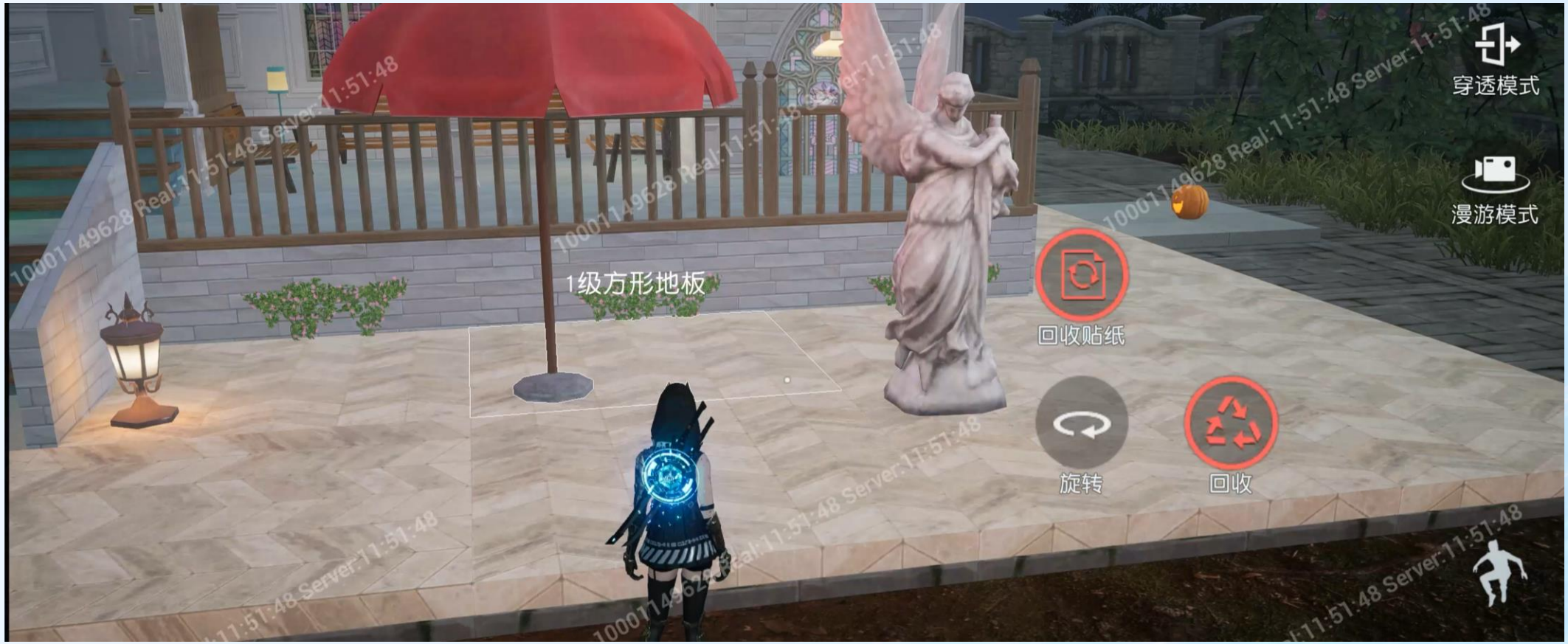
Player Moving

- On the SD855 device, 300 probes per second are processed in an area of 15m x 15m x 8m, which is the main surrounding of the player.
- With appropriate probe caching, preloading, and idle time probe processing, real-time probe has no lag.



Result

Dynamic Crafting



Device Scalability

Applicable on the majority of mobile devices

- Compute shader required
- Indirect Draw required

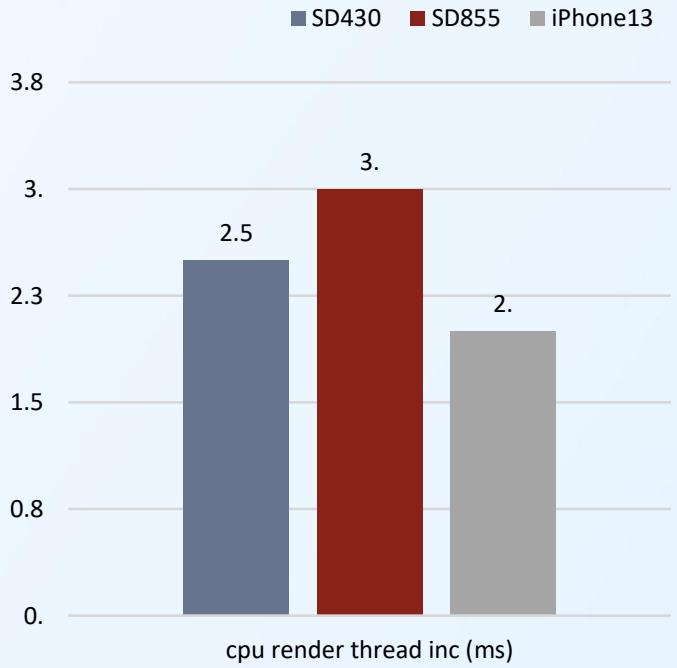
Configurable Parameters

- PRT SH or Ambient cube
- Probe generation speed
- Caching size
- Probe density

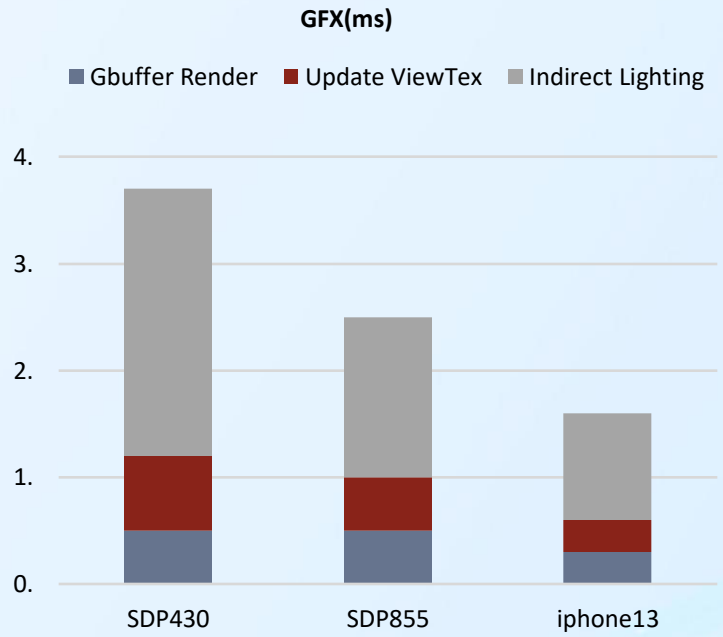
	probe storage	probe speed	caching probe size	probe density
Low-end Android	Ambient Cube	150/second	256*256	4meters
Middle-end Android	Ambient Cube	300/second	512*512	2meters
High-end iPhone	PRT SH	600/second	256*256	1meter

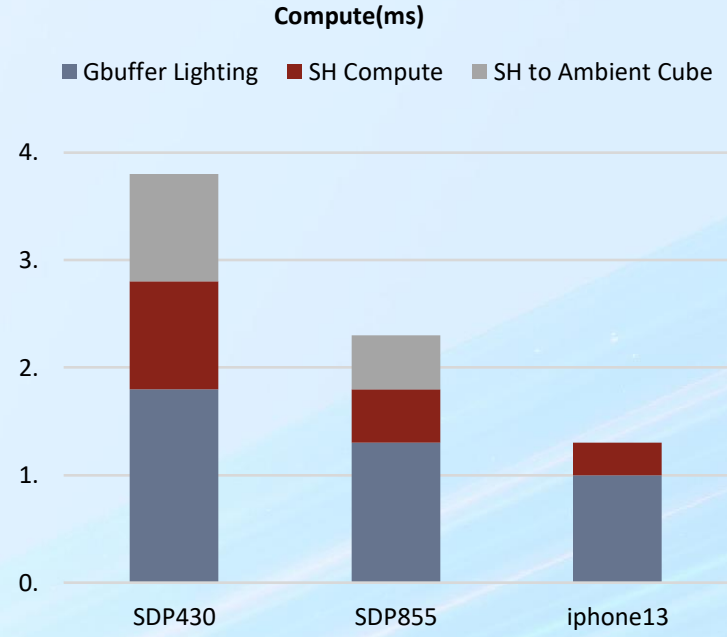
Performance

CPU Increase



GPU Increase






LIGHTSPEED
STUDIOS
GDC
MEET
LIGHTSPEED STUDIOS
AT GDC2023
March 20-24, 2023 | San Francisco, CA