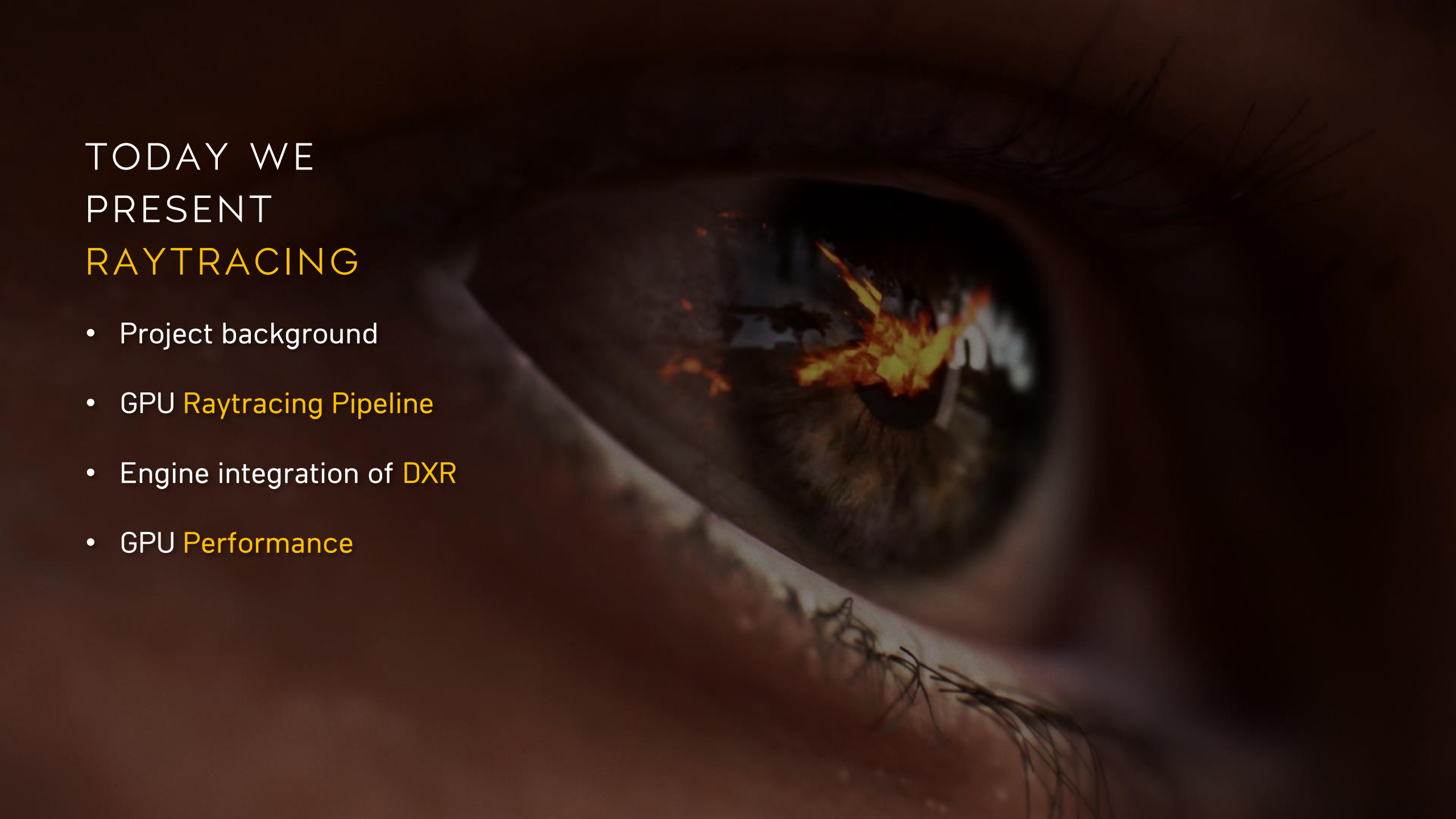
"IT JUST WORKS": RAY-TRACED REFLECTIONS IN 'BATTLEFIELD V'

Johannes Deligiannis
Jan Schmid
EA DICE

GAME DEVELOPERS CONFERENCE
MARCH 18–22, 2019 | #GDC19



TODAY WE PRESENT RAYTRACING

- Project background
- GPU Raytracing Pipeline
- Engine integration of DXR
- GPU Performance

- FPS set in WWII
- Released Nov 2018
- Raytracing work began Dec 2017
- First DXR game released!

PROJECT BACKGROUND

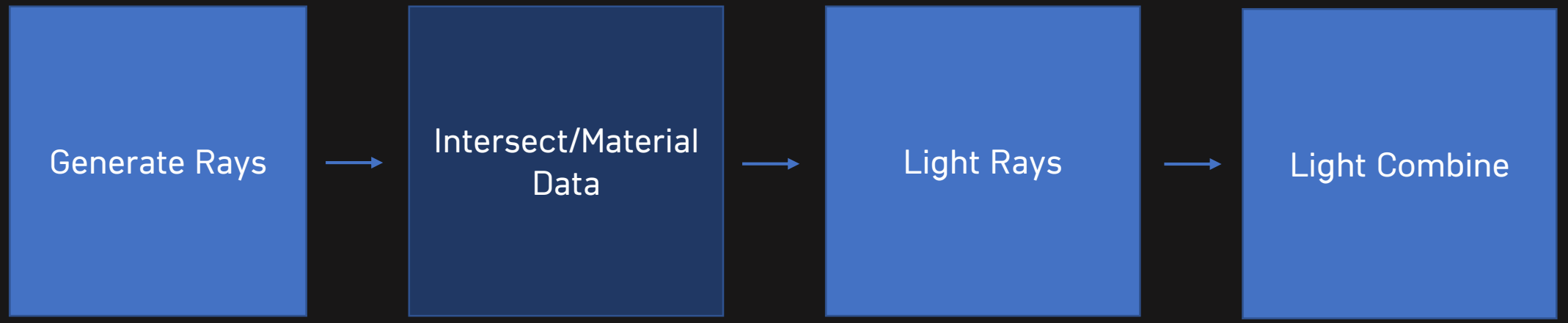
- ~10 months dev time
- Use DXR in Battlefield V
  - AO
  - GI
  - Shadows
  - Reflections

- Engineering
  - Yasin Uludag (EA DICE)
  - Johannes Deligiannis (EA DICE)
  - Jiho Choi (NVIDIA)
  - Pawel Kozlowski (NVIDIA)
- And a bunch of other people! ☺

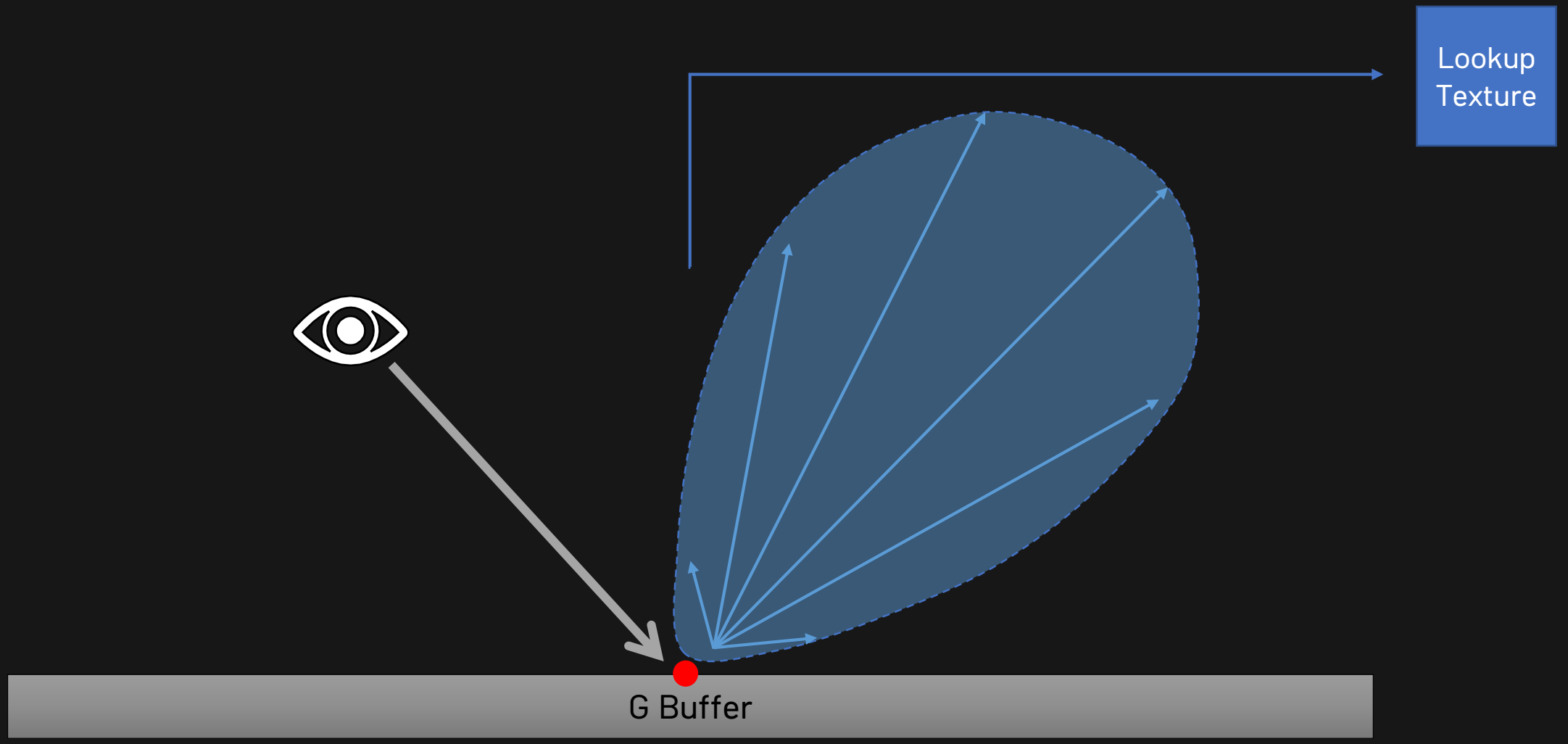
MAIN CHALLANGES

- Not a Tech Demo
  - Content is set
- Game in full production
- Scope of Engine changes
- Performance
  - Denoising vs Ray Count
  - No RTX cards
- Early adopter tax
  - API not final
  - Driver hang/bugs
  - BSoD
  - No capture tool (Nsight, Pix)
- But we shipped it☺

(SIMPLE) RAYTRACING PIPELINE



GENERATE RAYS



*Tomasz Stachowiak and Yasin Uludag, Siggraph 2015. “**Stochastic Screen-Space Reflections**”

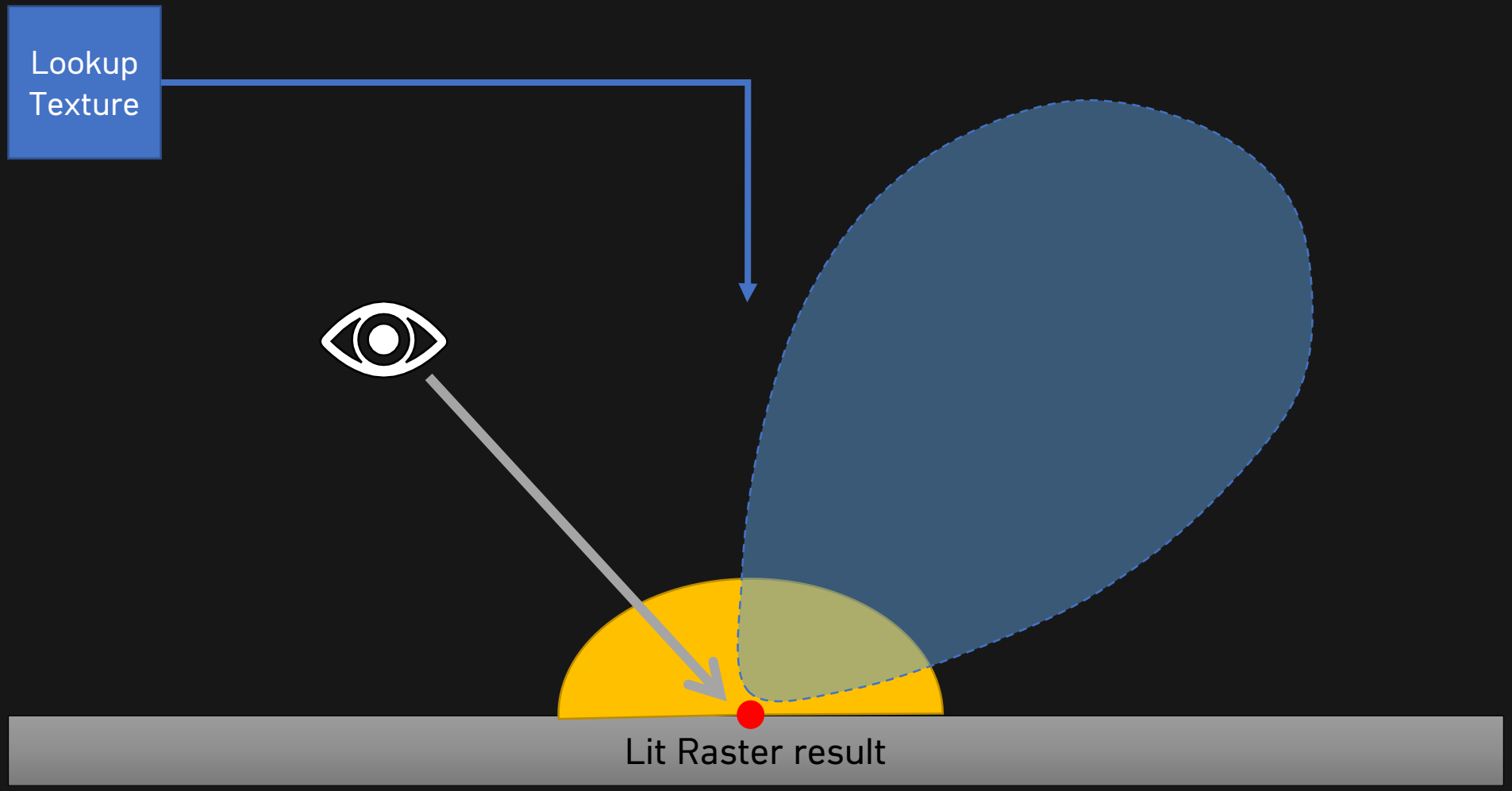
RAYTRACING

MAGIC

LIGHT RAYS

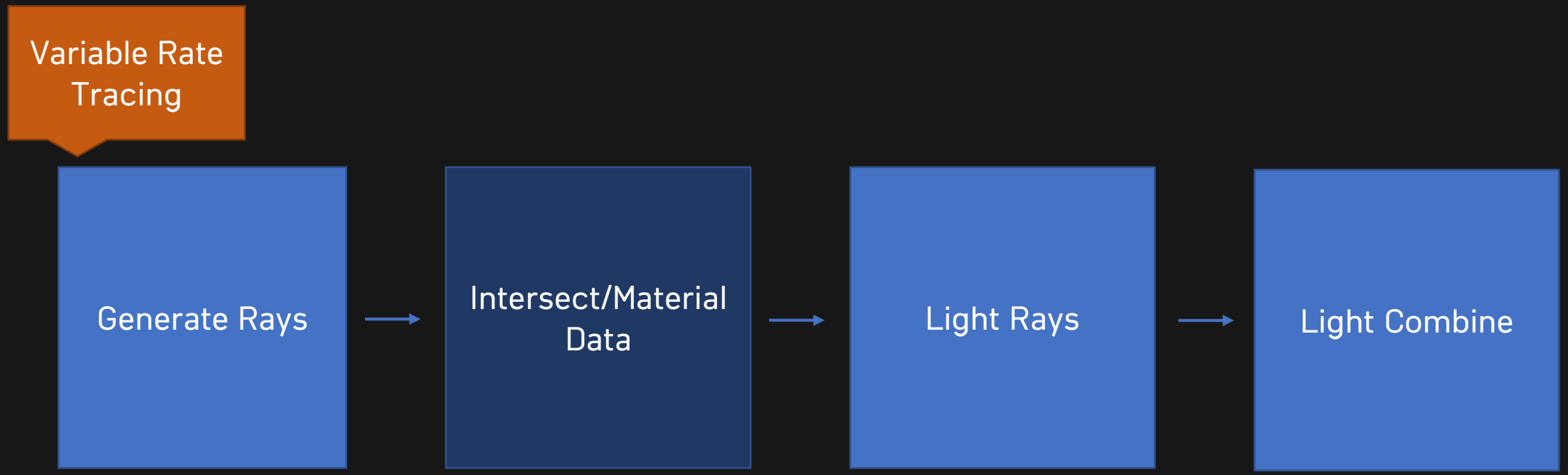
```
float4 light(MaterialData surfaceInfo, float3 rayDir) {

    foreach (light : pointLights)
        radiance += calcPoint(surfaceInfo, rayDir, light);

    foreach (light : spotLights)
        radiance += calcSpot(surfaceInfo, rayDir, light);

    foreach (light : reflectionVolumes)
        radiance += calcReflVol(surfaceInfo, rayDir, light);


…
}
```

LIGHT COMBINE

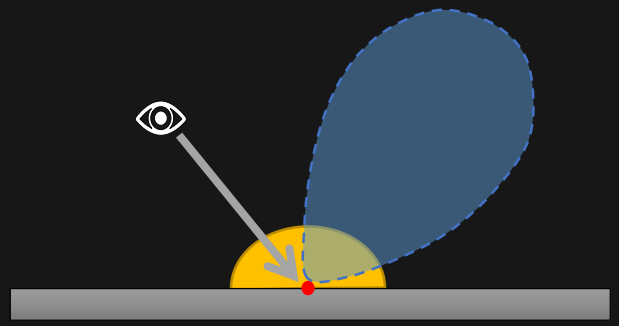



Rays Contribute Less
Slooooow
18,458ms
Very Noisy

IMPROVING RAYTRACING PIPELINE



VARIABLE RATE TRACING



Max Ratio

0	.5	.5	0
.5	1	1	.5
0	.5	.5	0

Normalize

0	.1	.1	0
.1	.2	.2	.1
0	.1	.1	0

Classify

0	128	128	0
128	256	256	128
0	128	128	0

VARIABLE RATE TRACING

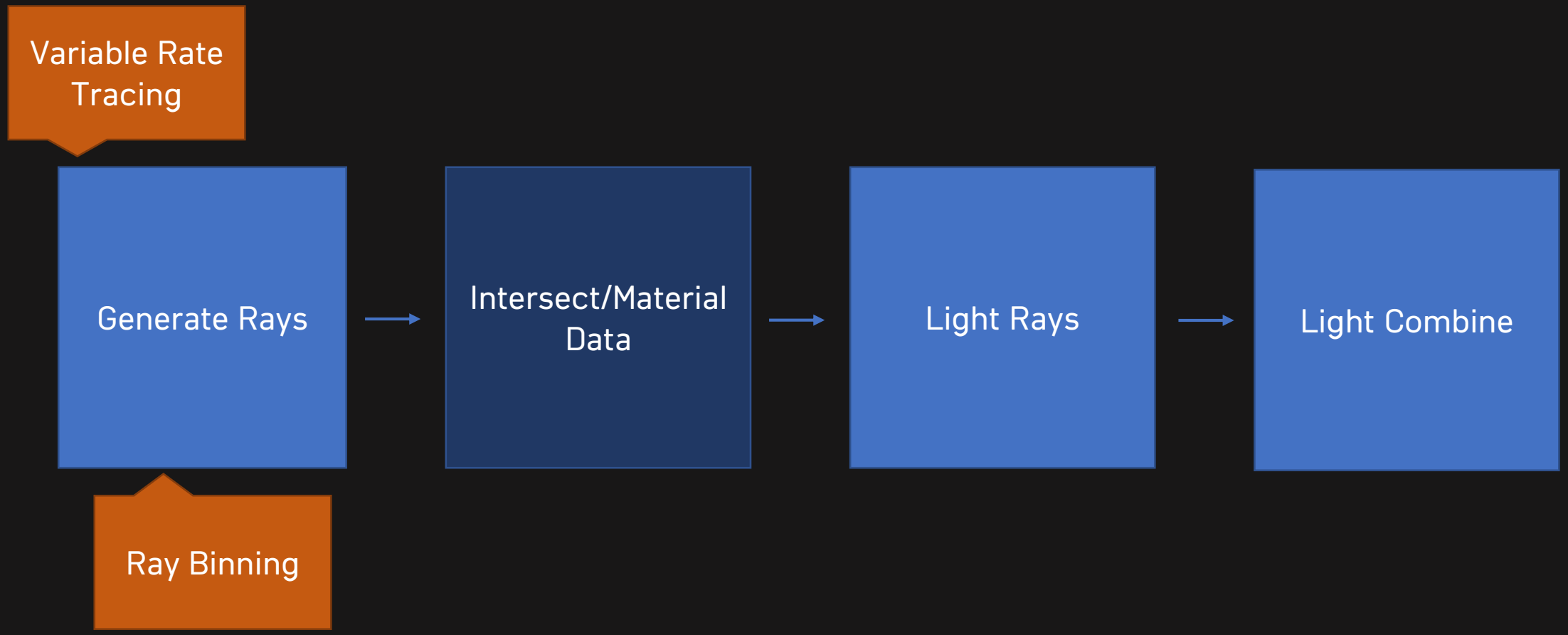
VARIABLE RATE TRACING

Success!

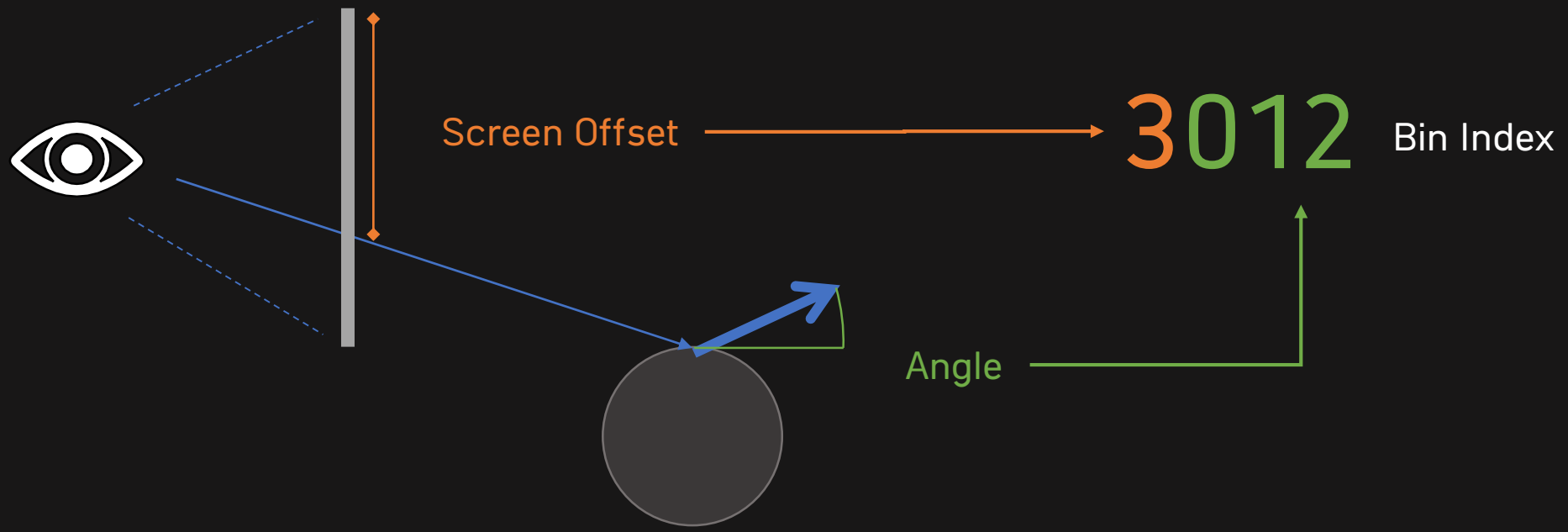
- More Rays on Water
- More Rays on grazing angles

PROBLEM

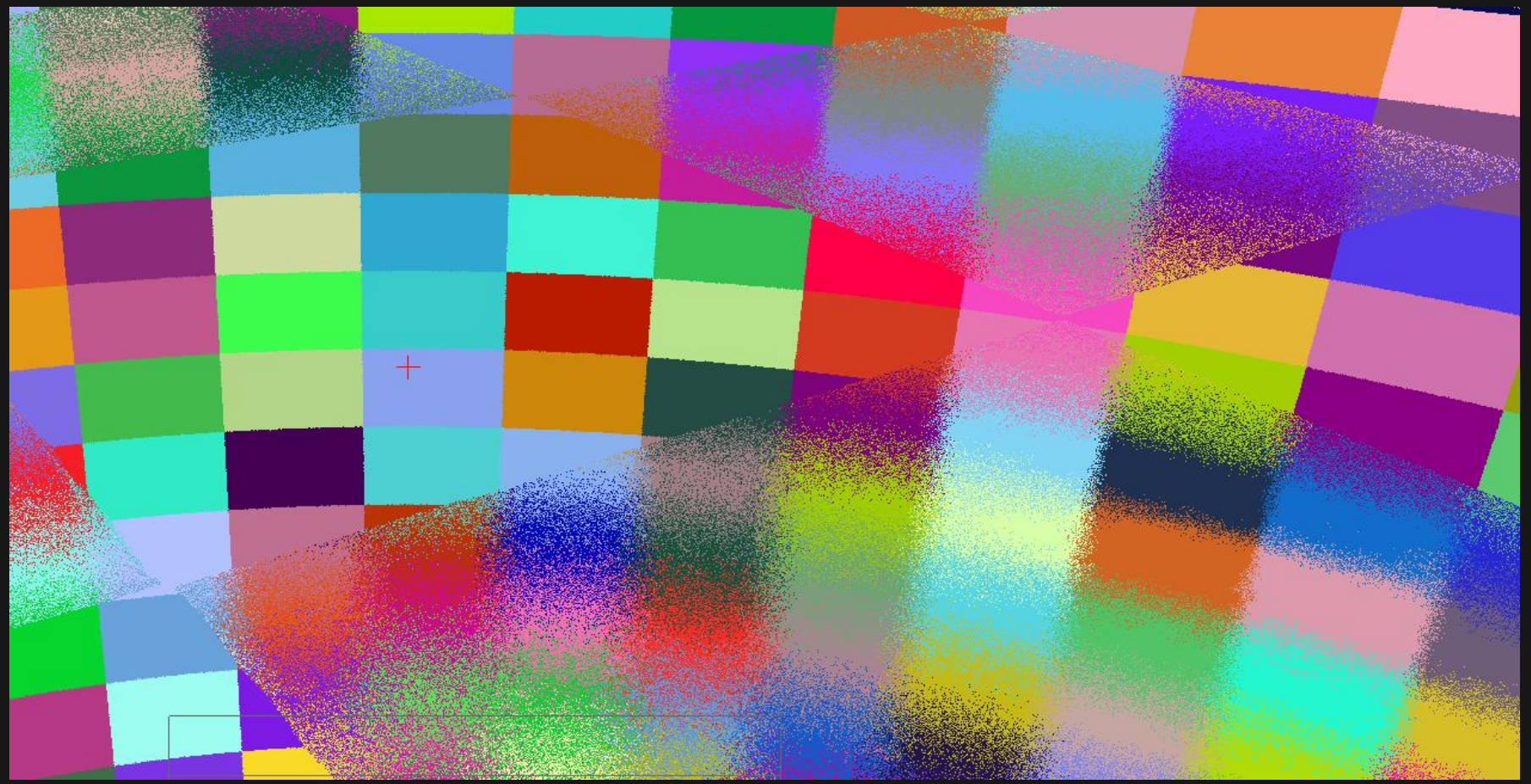
IMPROVING RAYTRACING PIPELINE



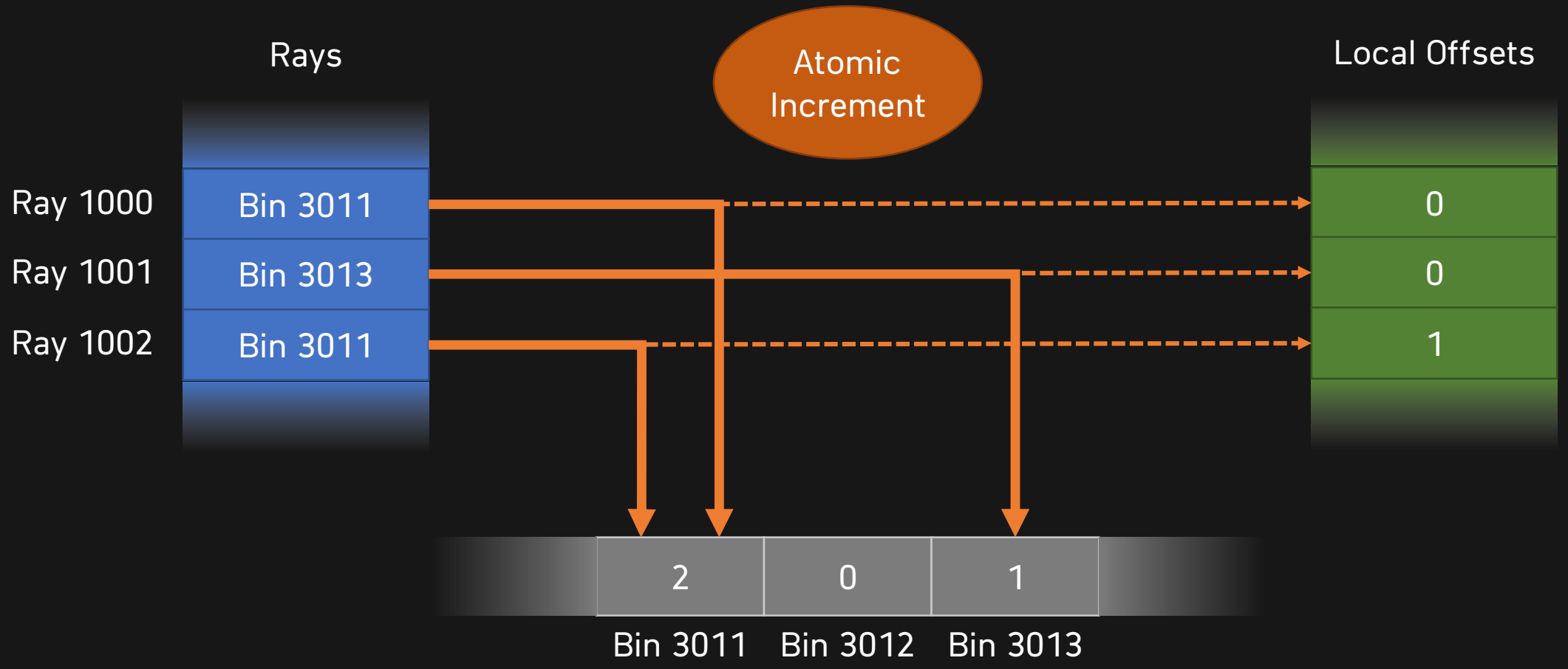
RAY BINNING



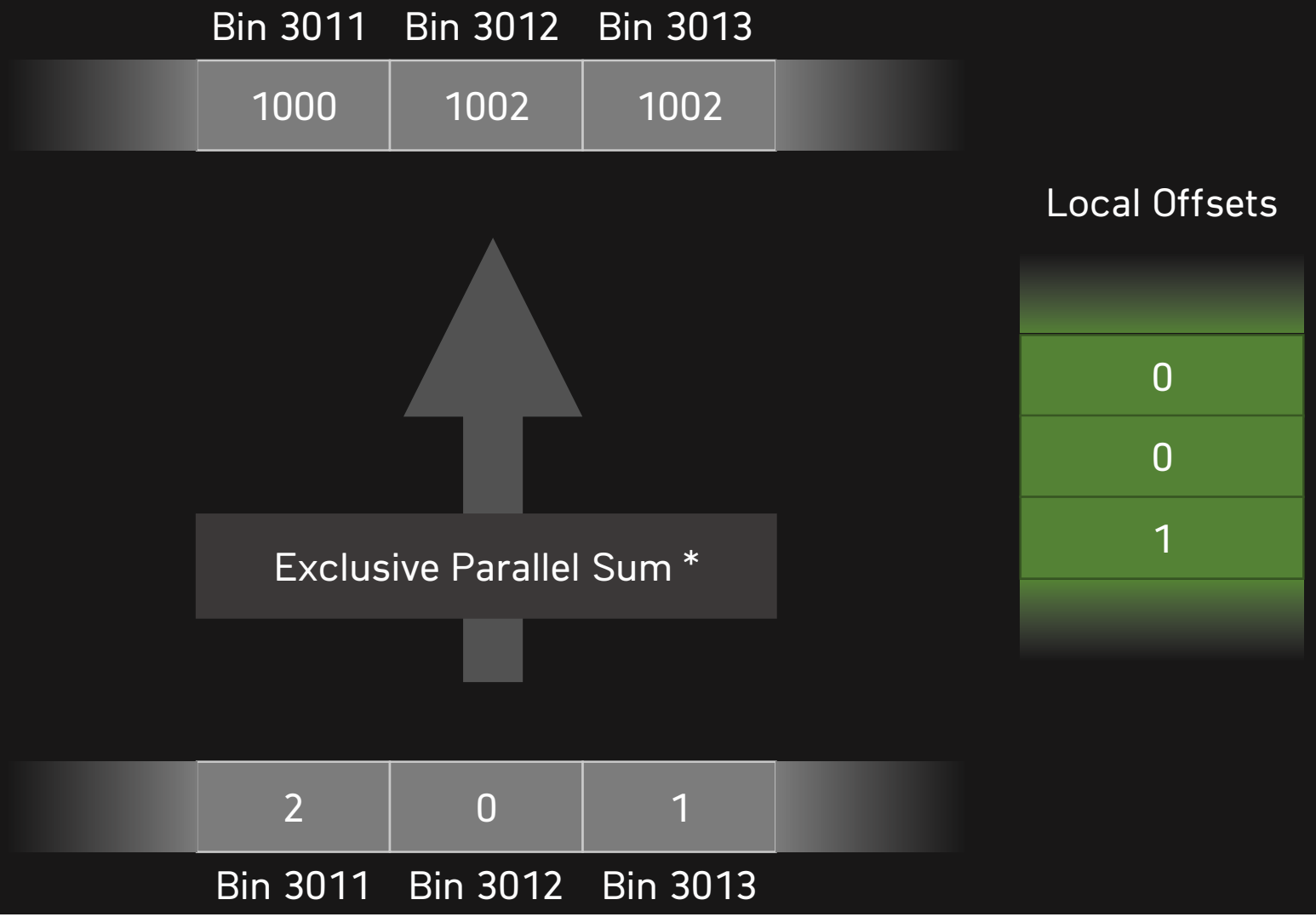
RAY BINNING



RAY BINNING

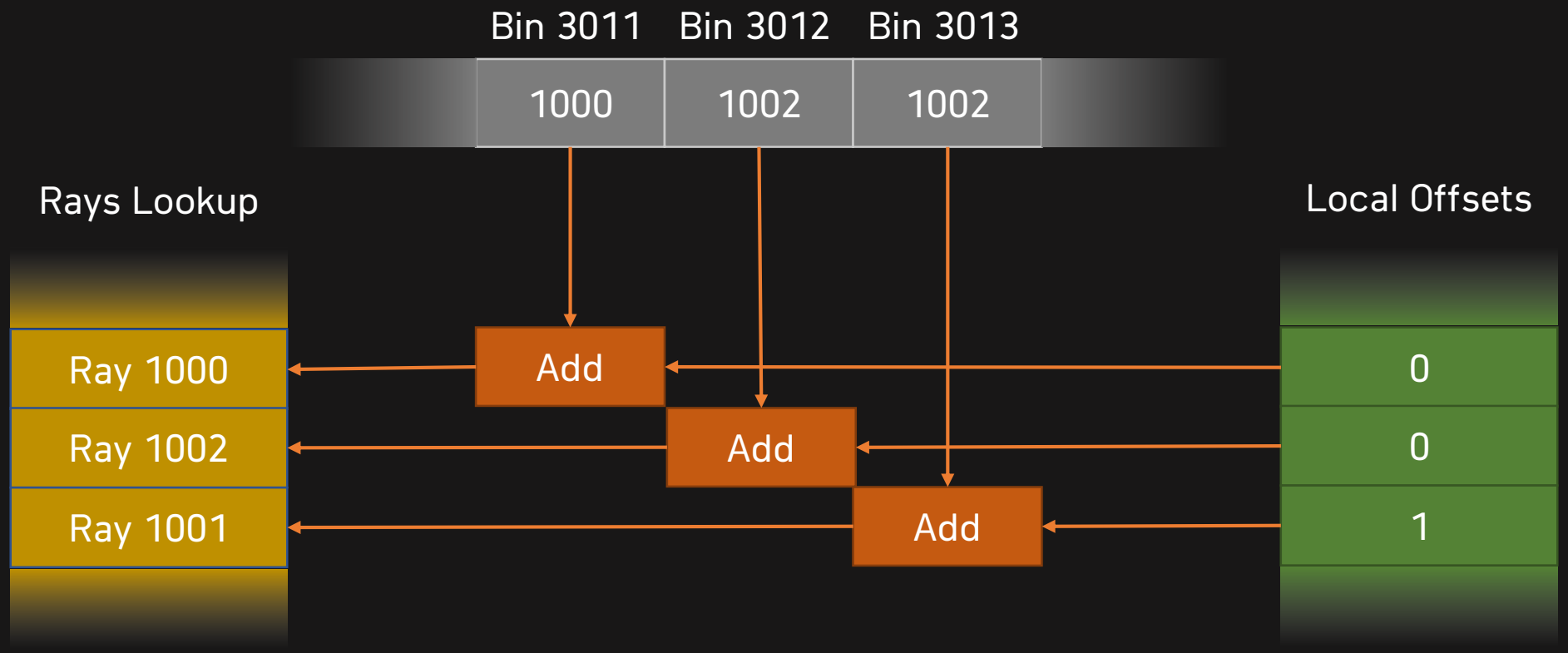

Rays
Atomic Increment
Local Offsets
Ray 1000
Ray 1001
Ray 1002
Bin 3011
Bin 3013
Bin 3011
0
0
1
2
0
1
Bin 3011
Bin 3012
Bin 3013

RAY BINNING



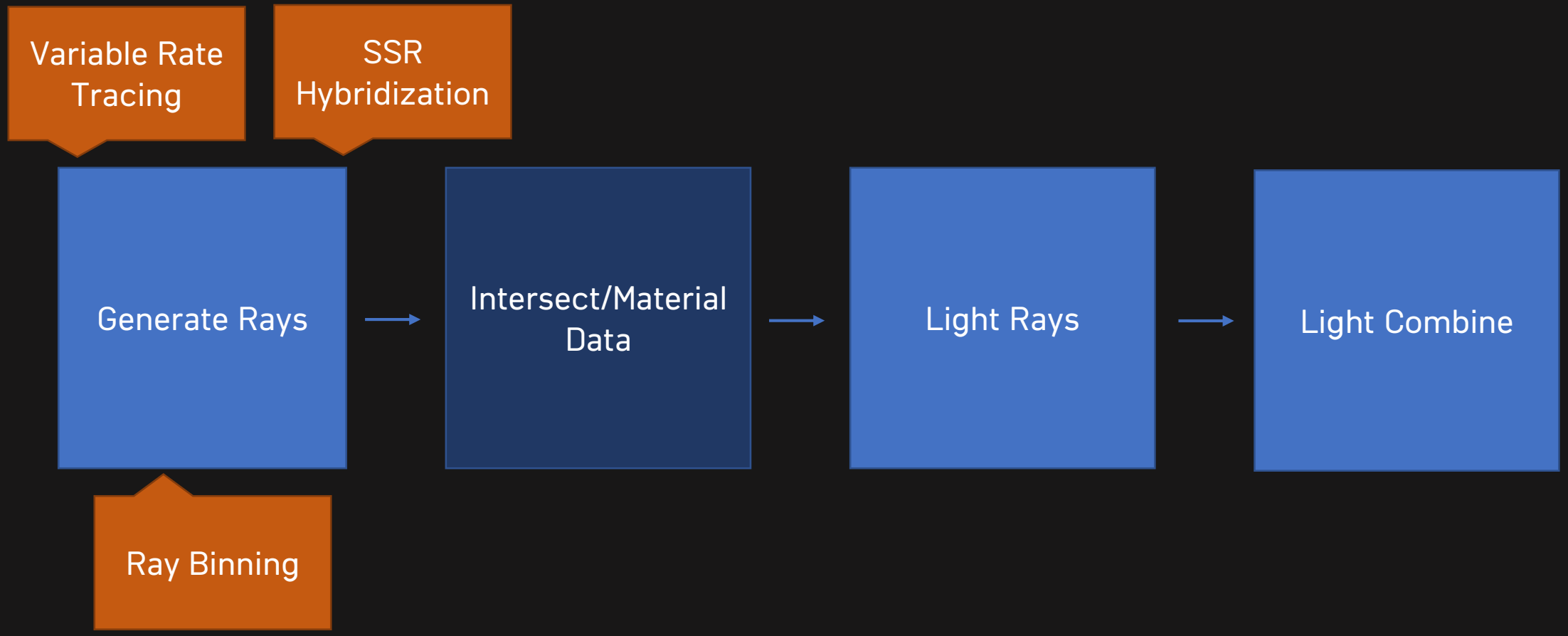
*Mark Harris, Shubhabrata Sengupta, and John Owens. “Parallel Prefix Sum (Scan) with CUDA”

RAY BINNING

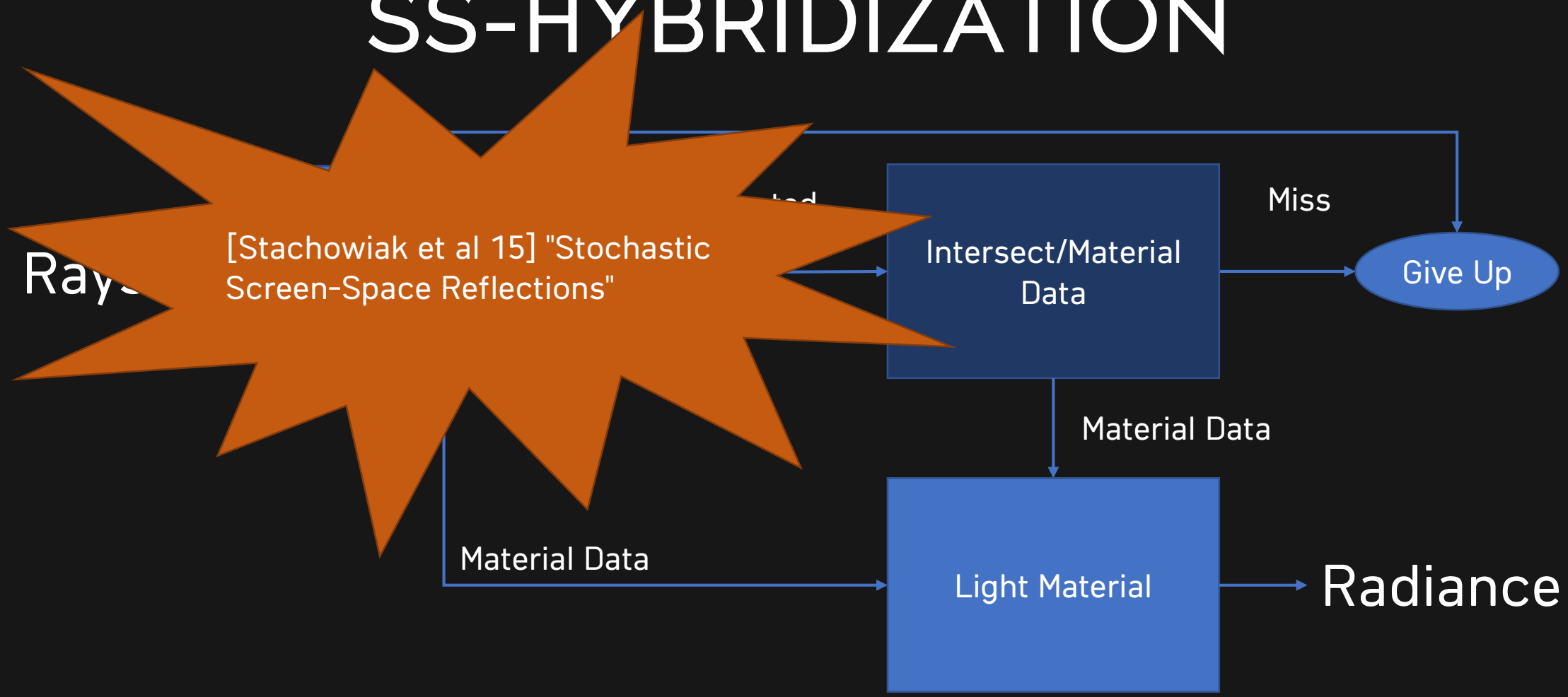

Bin 3011
Bin 3012
Bin 3013
1000
1002
1002
Rays Lookup
Local Offsets
Ray 1000
Ray 1002
Ray 1001
Add
Add
Add
0
0
1

PROBLEM

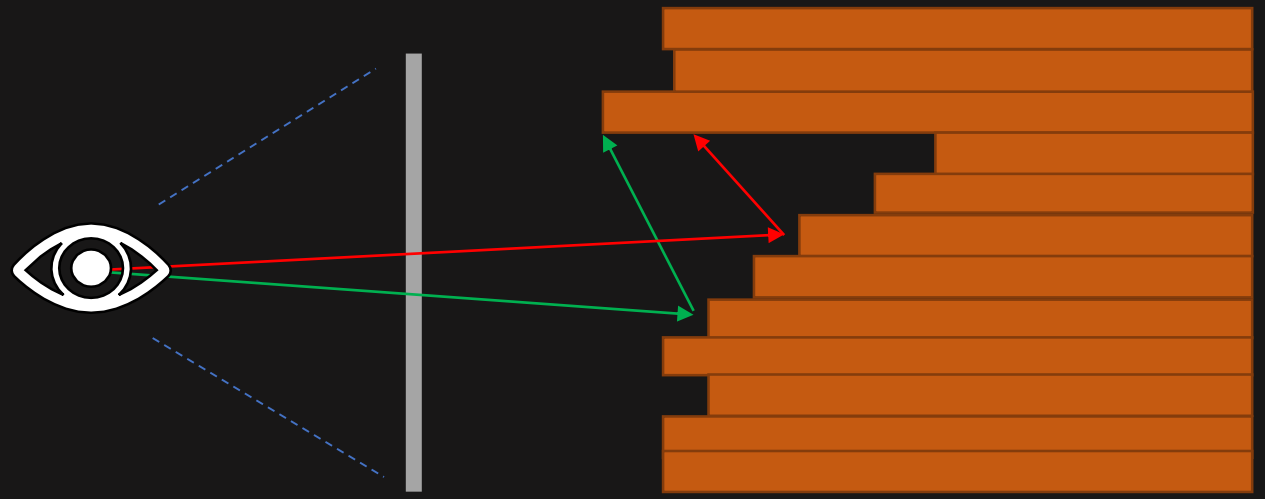
IMPROVING RAYTRACING PIPELINE



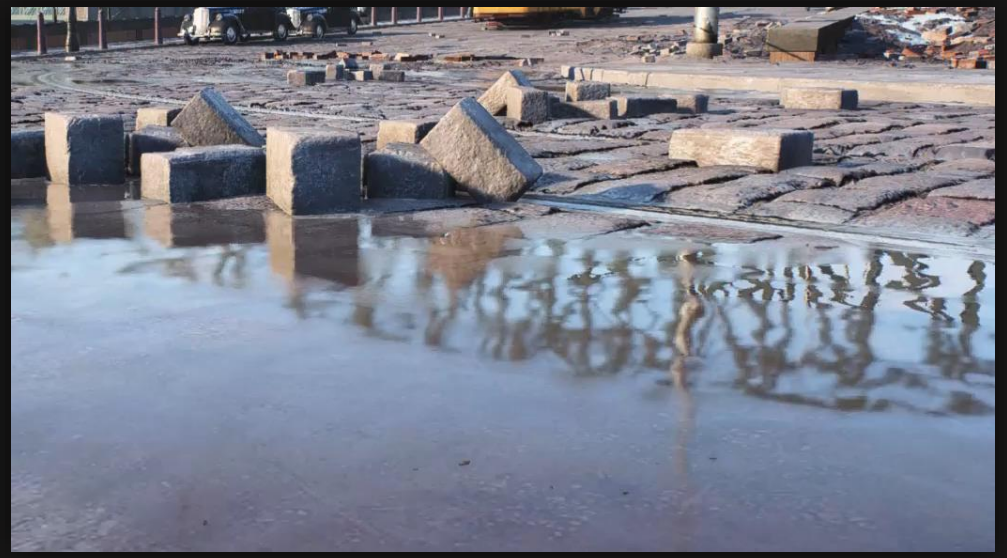
SS-HYBRIDIZATION



SS-HYBRIDIZATION

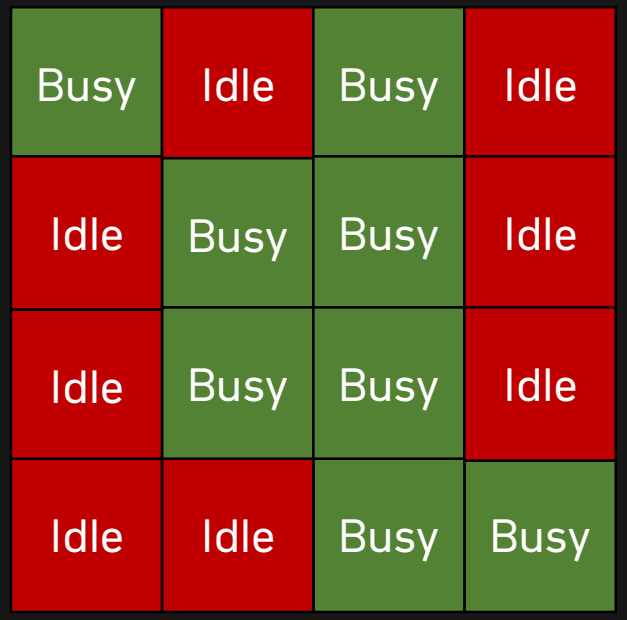


SS-HYBRIDIZATION



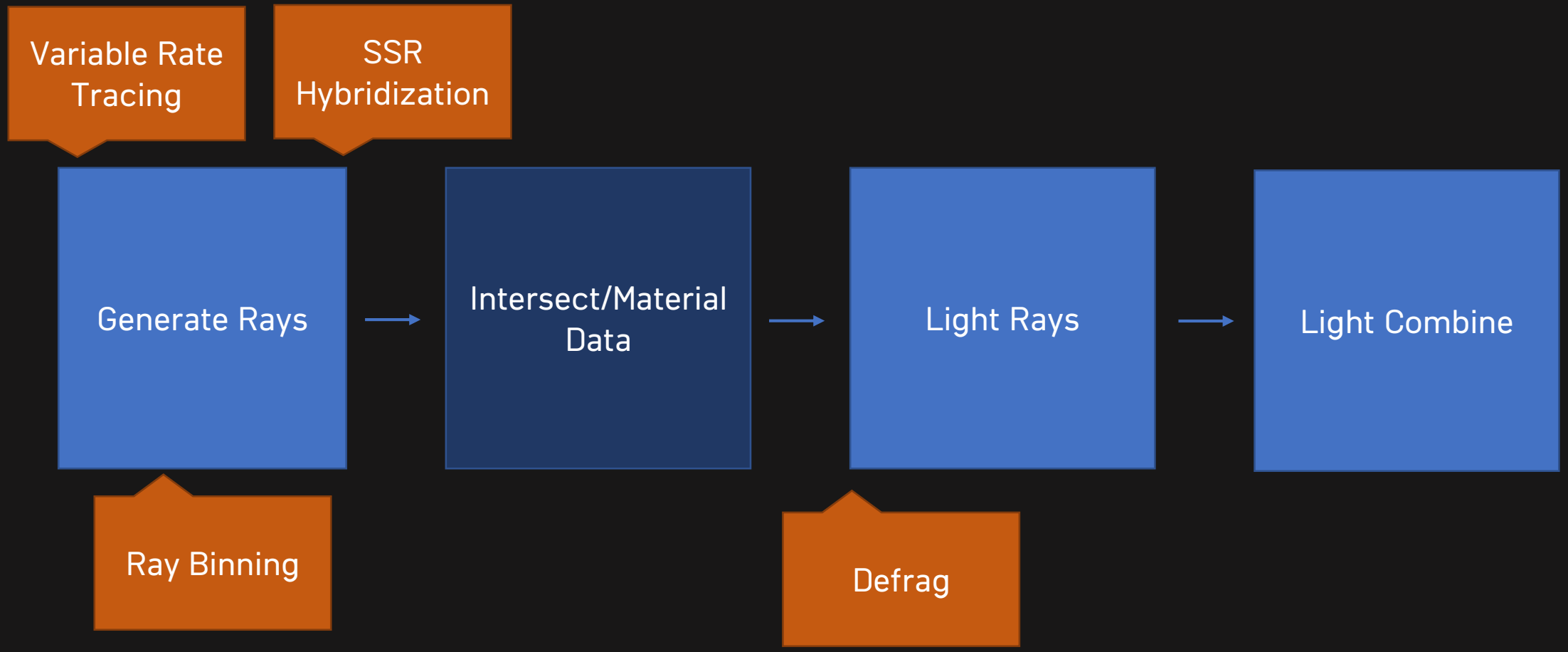
PROBLEM

Raytrace

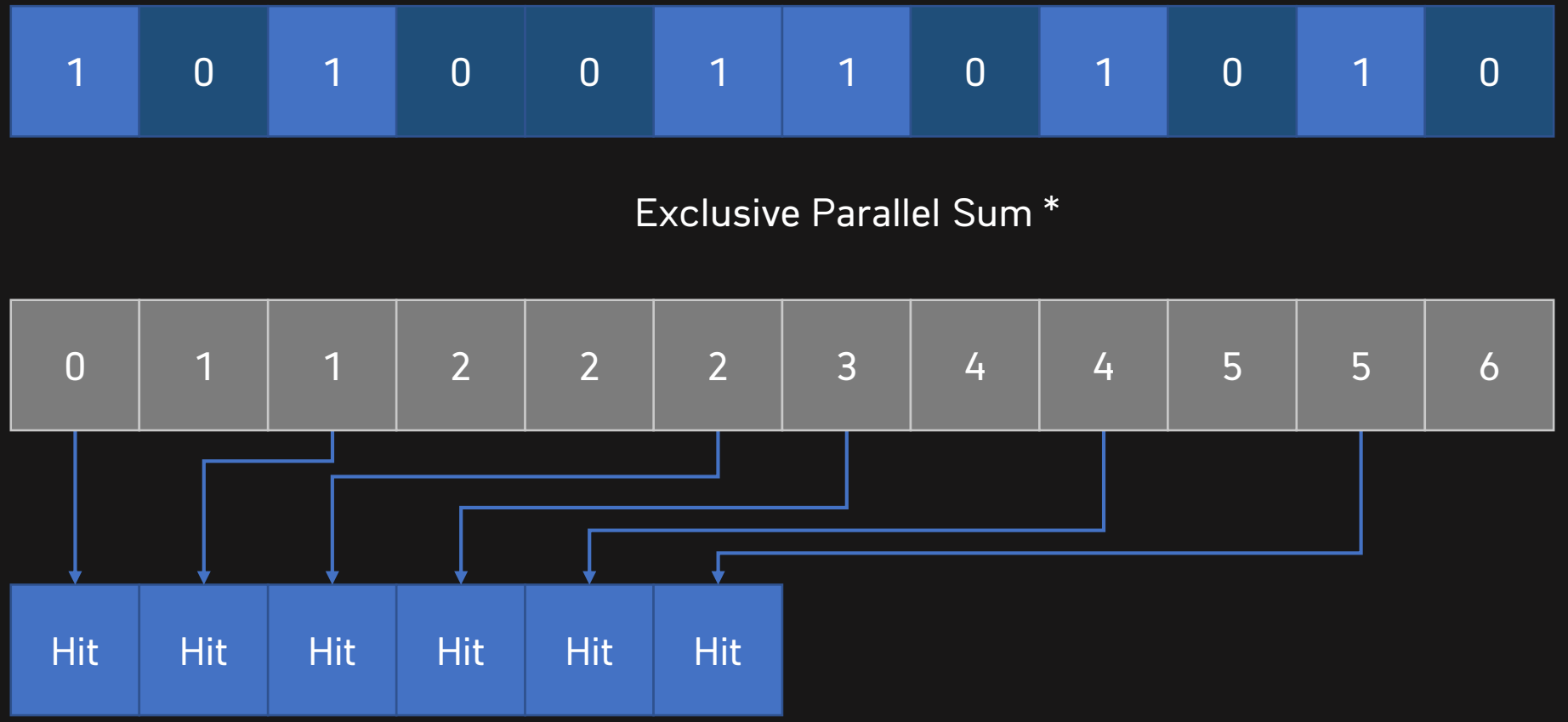


Light Shader Wavefront

IMPROVING RAYTRACING PIPELINE



DEFRAG



*Mark Harris, Shubhabrata Sengupta, and John Owens. “Parallel Prefix Sum (Scan) with CUDA”

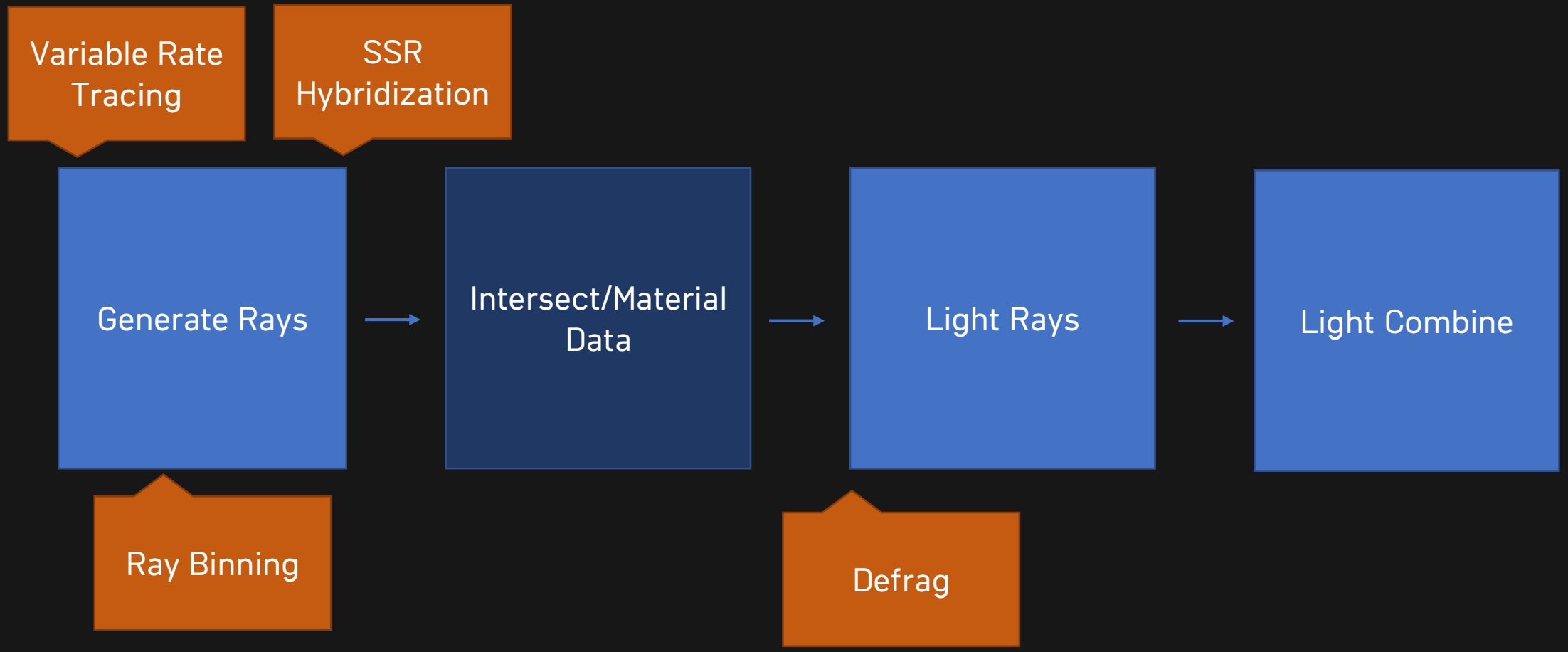
PROBLEM

Busy	Busy	Busy	Busy
Busy	Busy	Busy	Busy
Busy	Busy	Busy	Busy
Busy	Busy	Busy	Busy

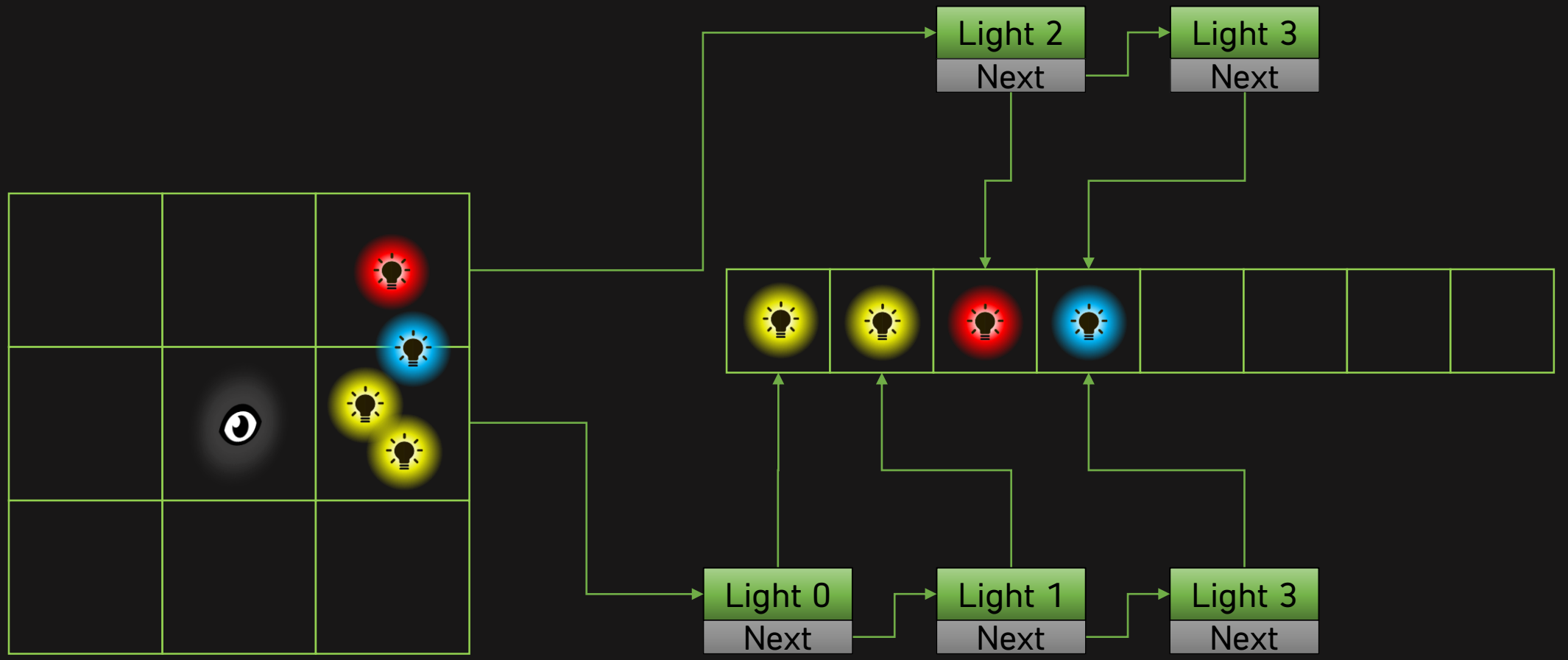
Light Shader

2.0ms

IMPROVING RAYTRACING PIPELINE

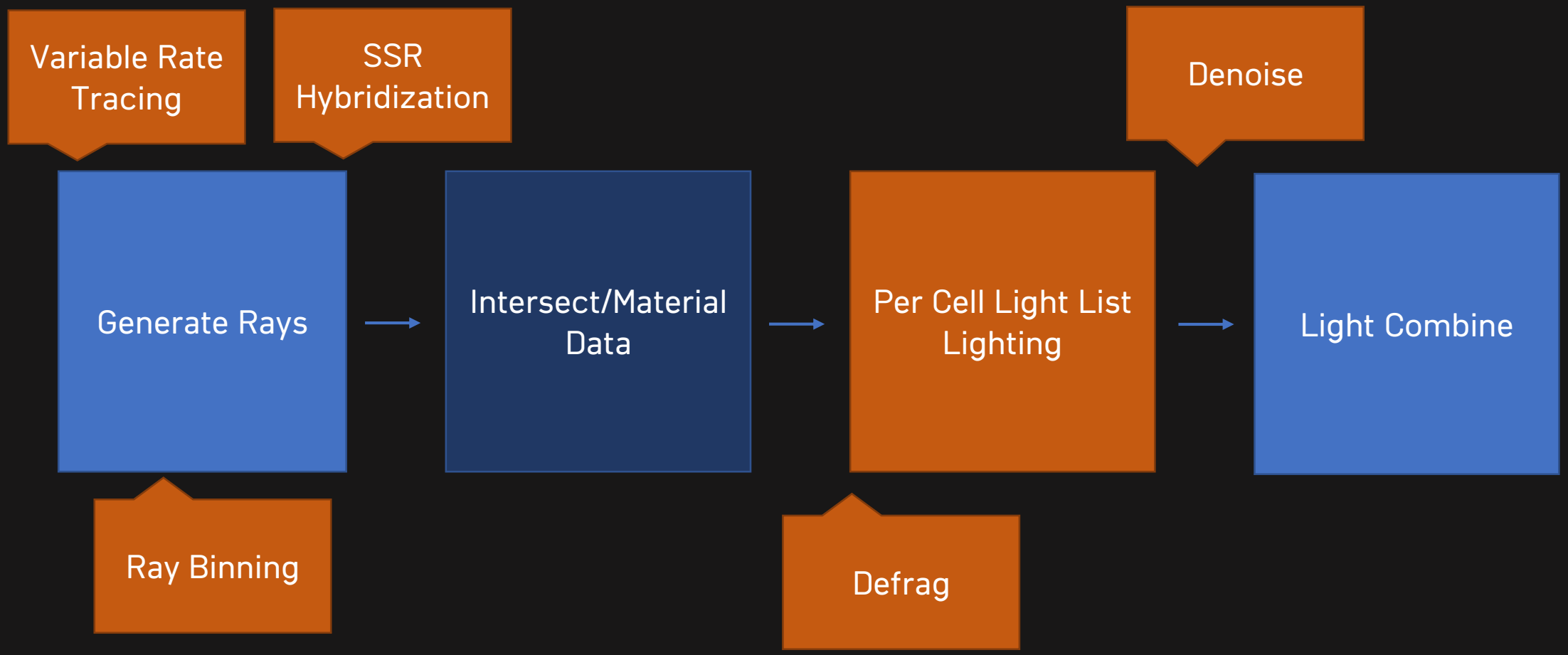


PER CELL LIGHT LISTS

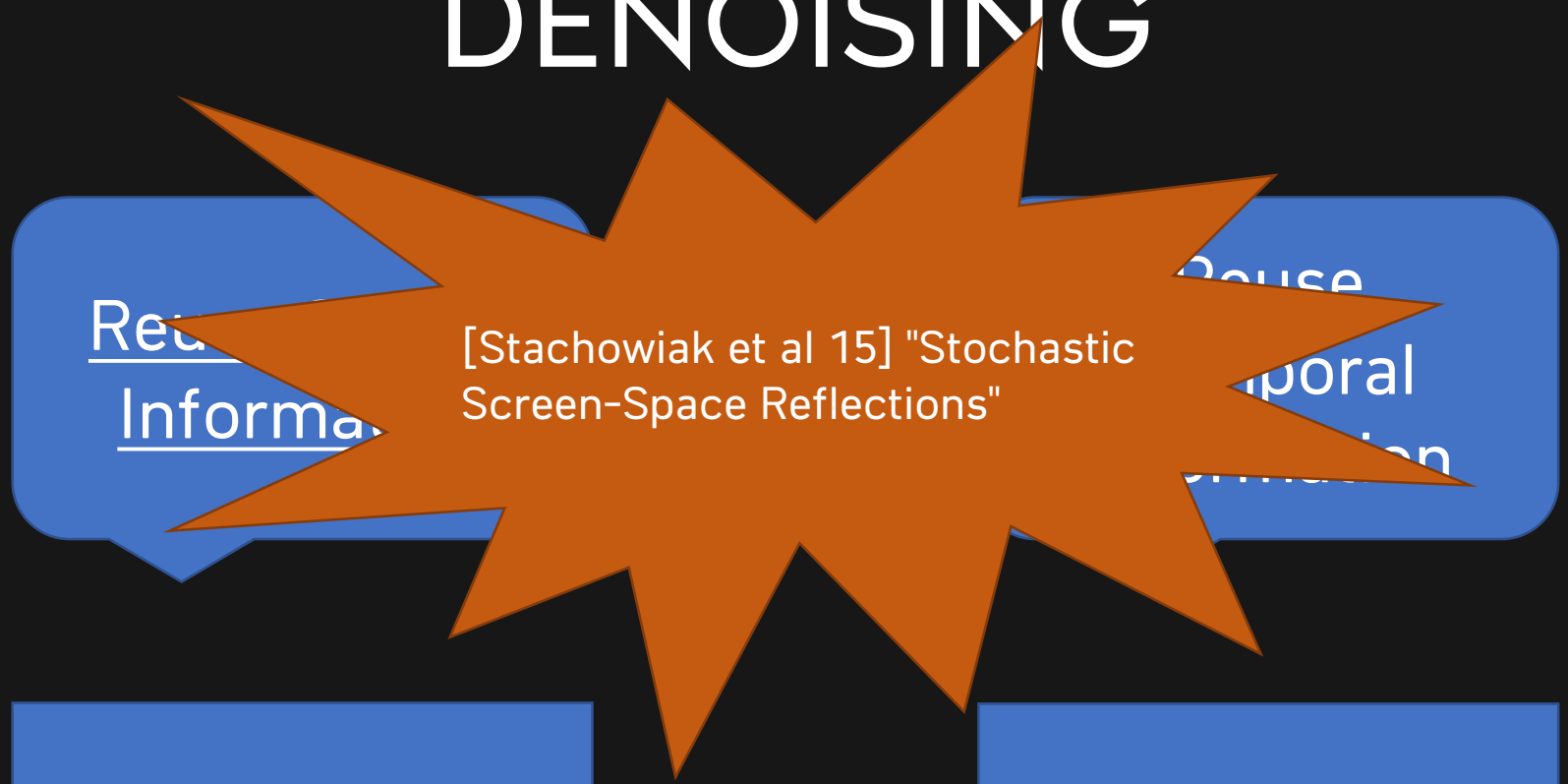


PROBLEM

IMPROVING RAYTRACING PIPELINE



DENOISING

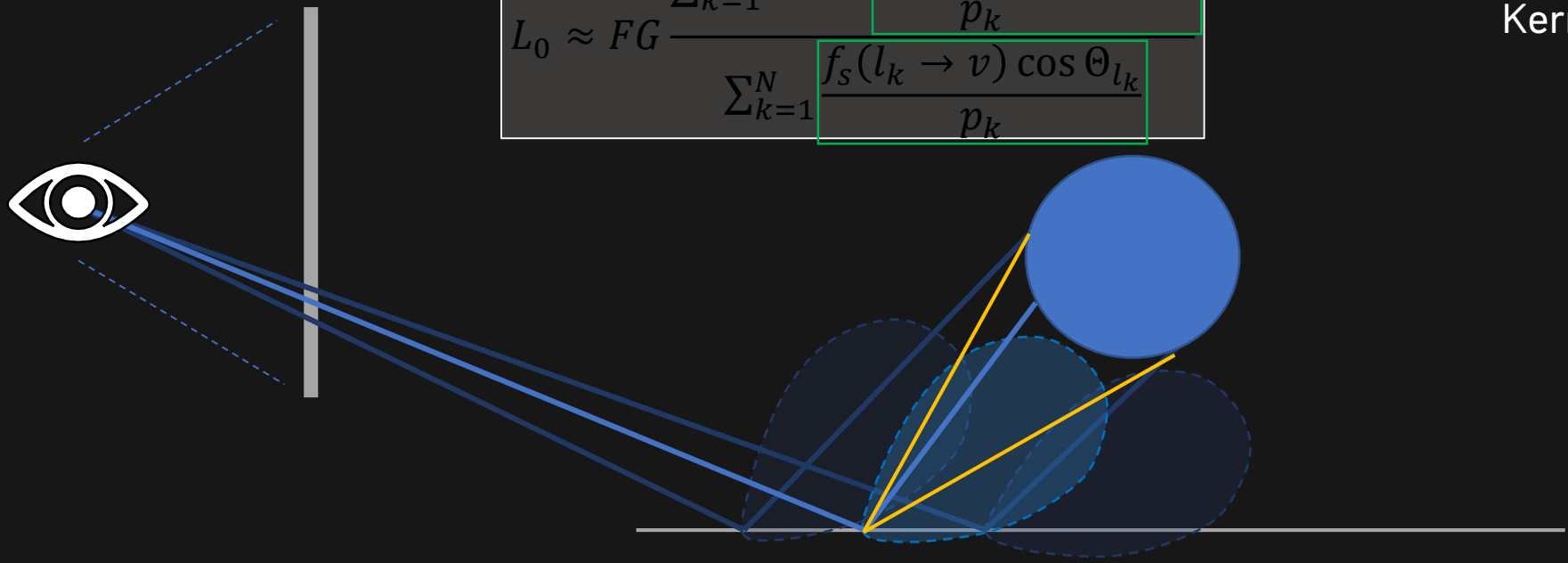


BRDF Filter

Temporal Filter

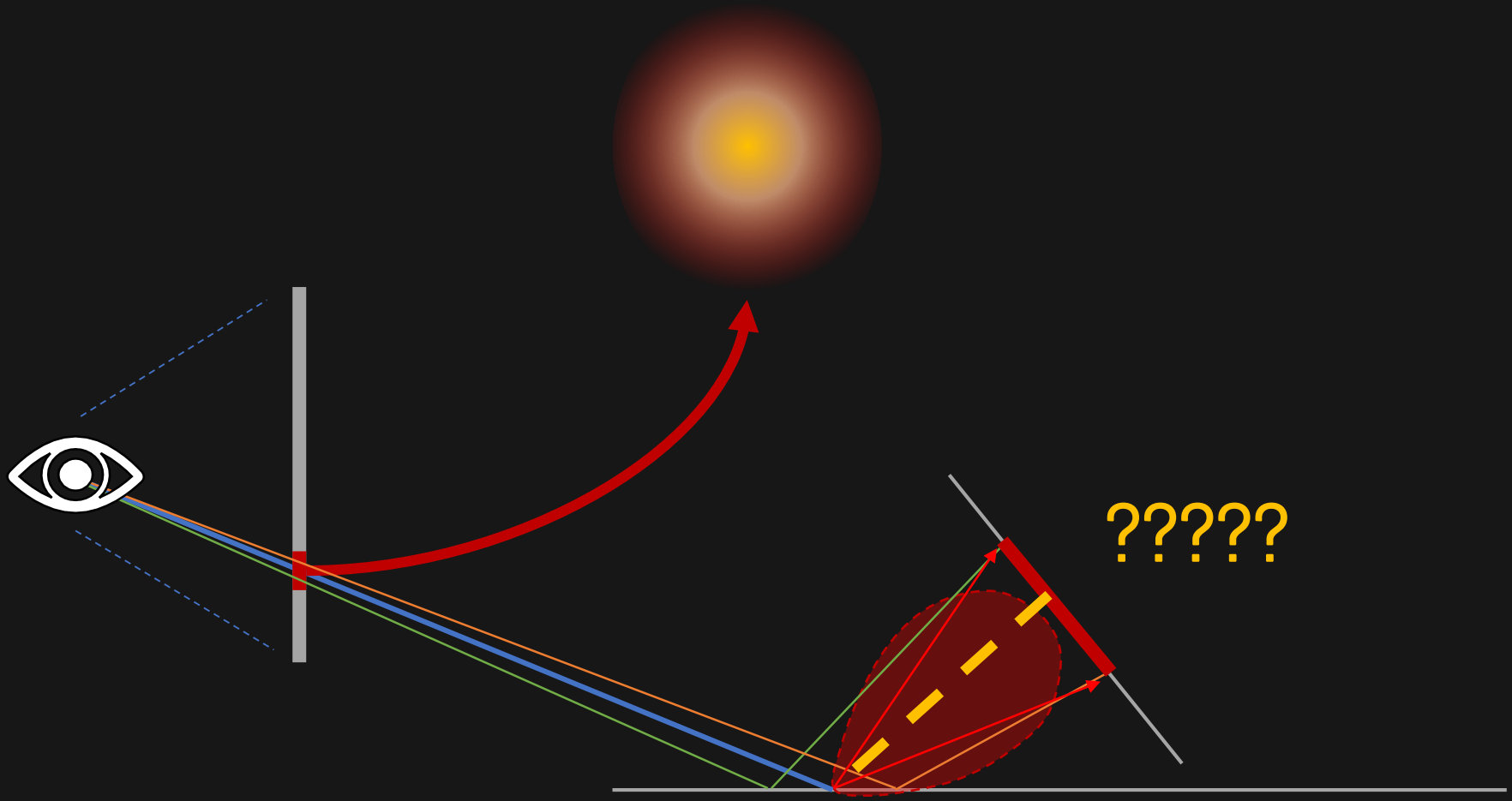
BRDF DENOISE FILTER

$$L_0 \approx FG \frac{\sum_{k=1}^{N} \frac{L_i(l_k) f_s(l_k \rightarrow v) \cos \Theta_{l_k}}{p_k}}{\sum_{k=1}^{N} \frac{f_s(l_k \rightarrow v) \cos \Theta_{l_k}}{p_k}}$$

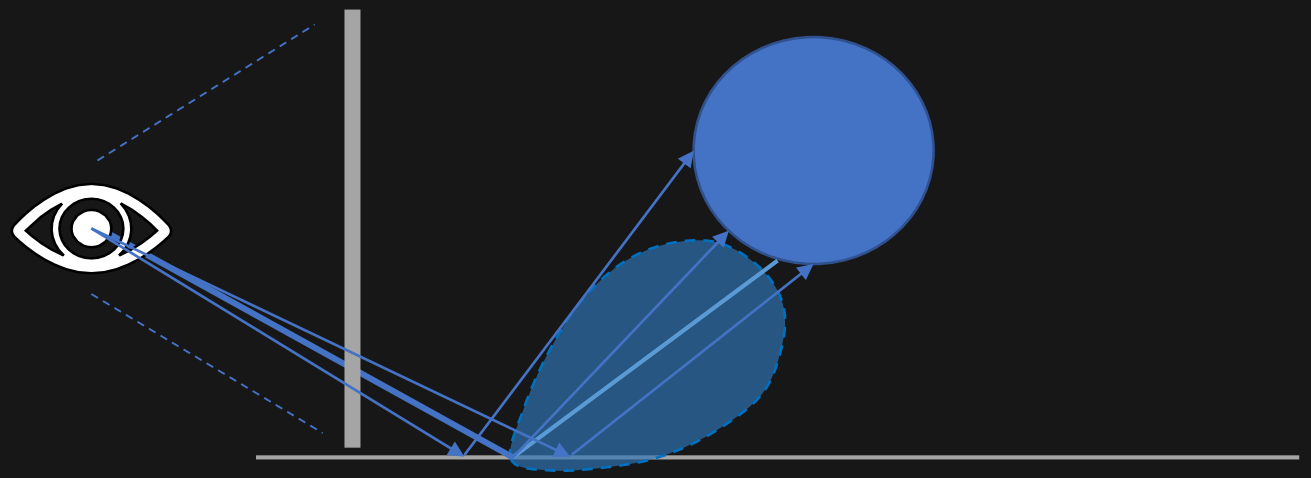


Kernel Size????

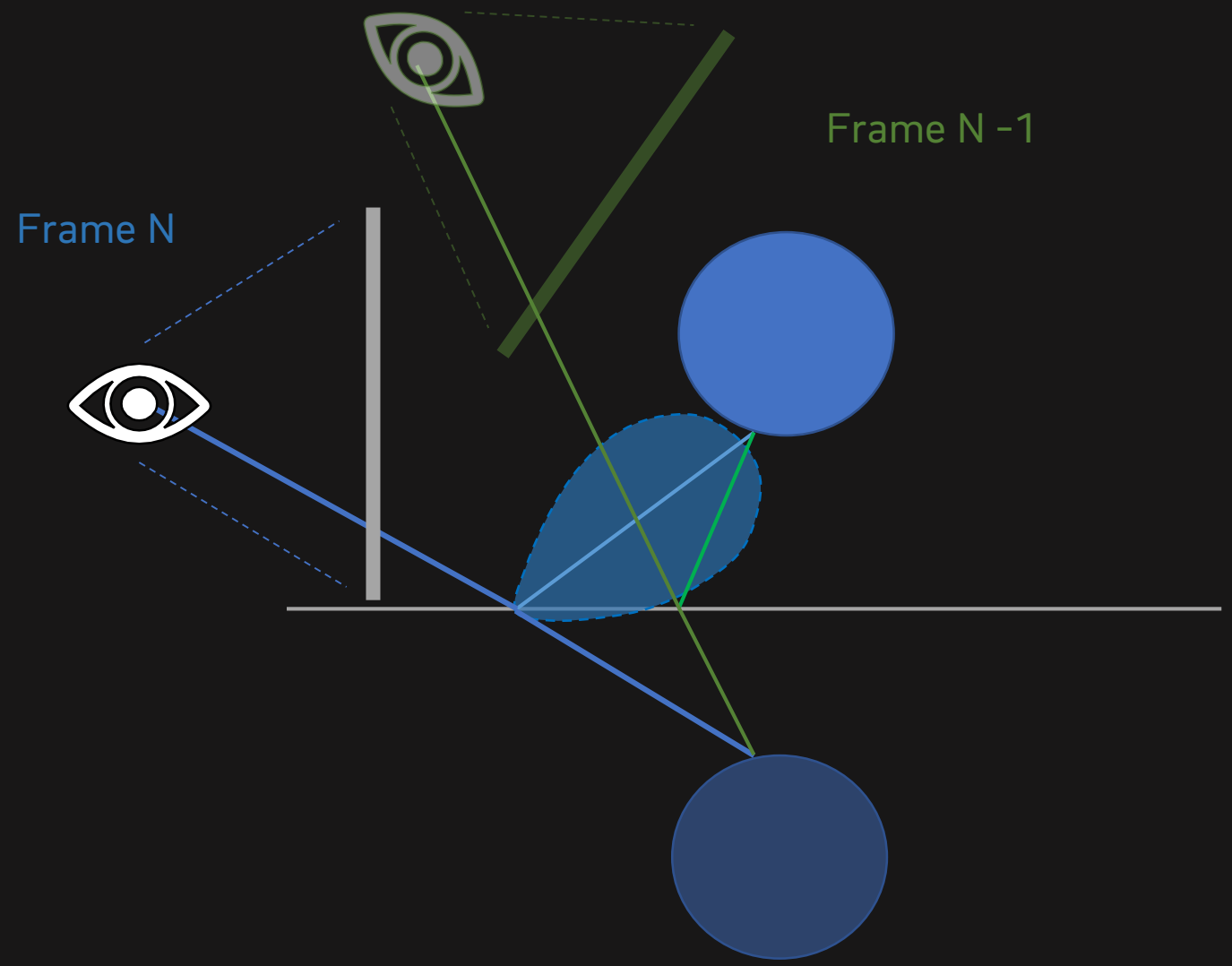
BRDF DENOISE FILTER



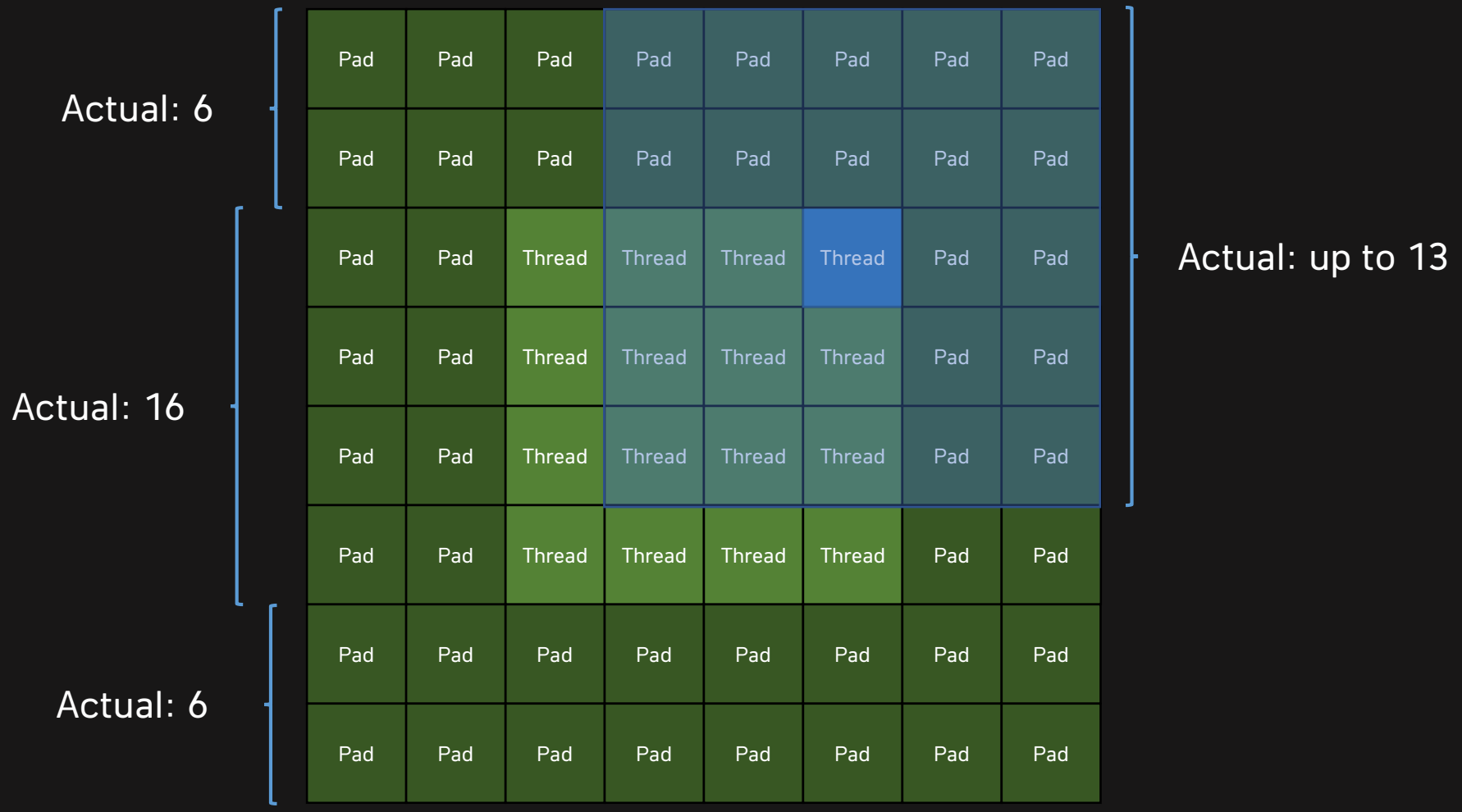
BRDF DENOISE FILTER



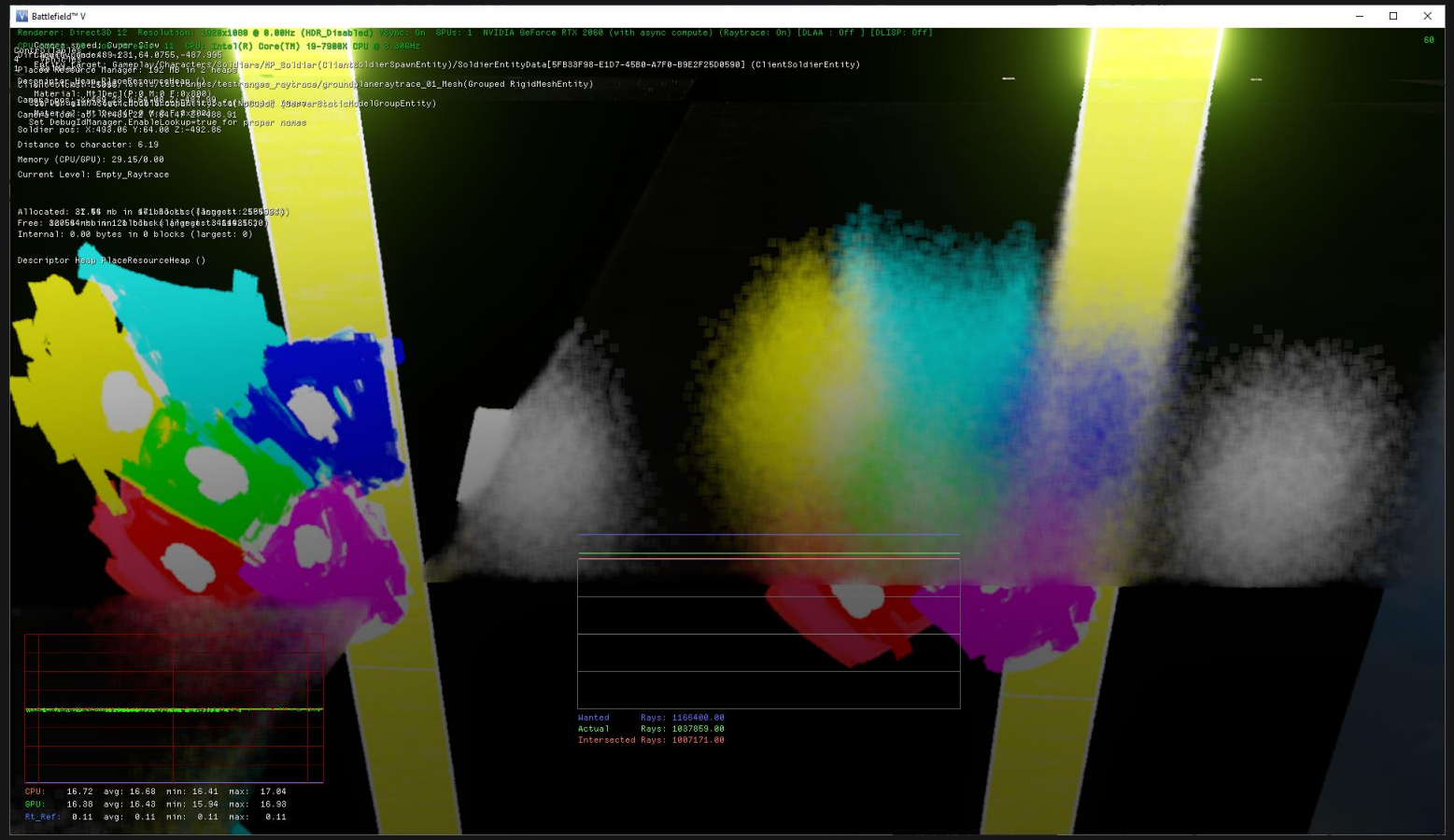
BRDF DENOISE FILTER



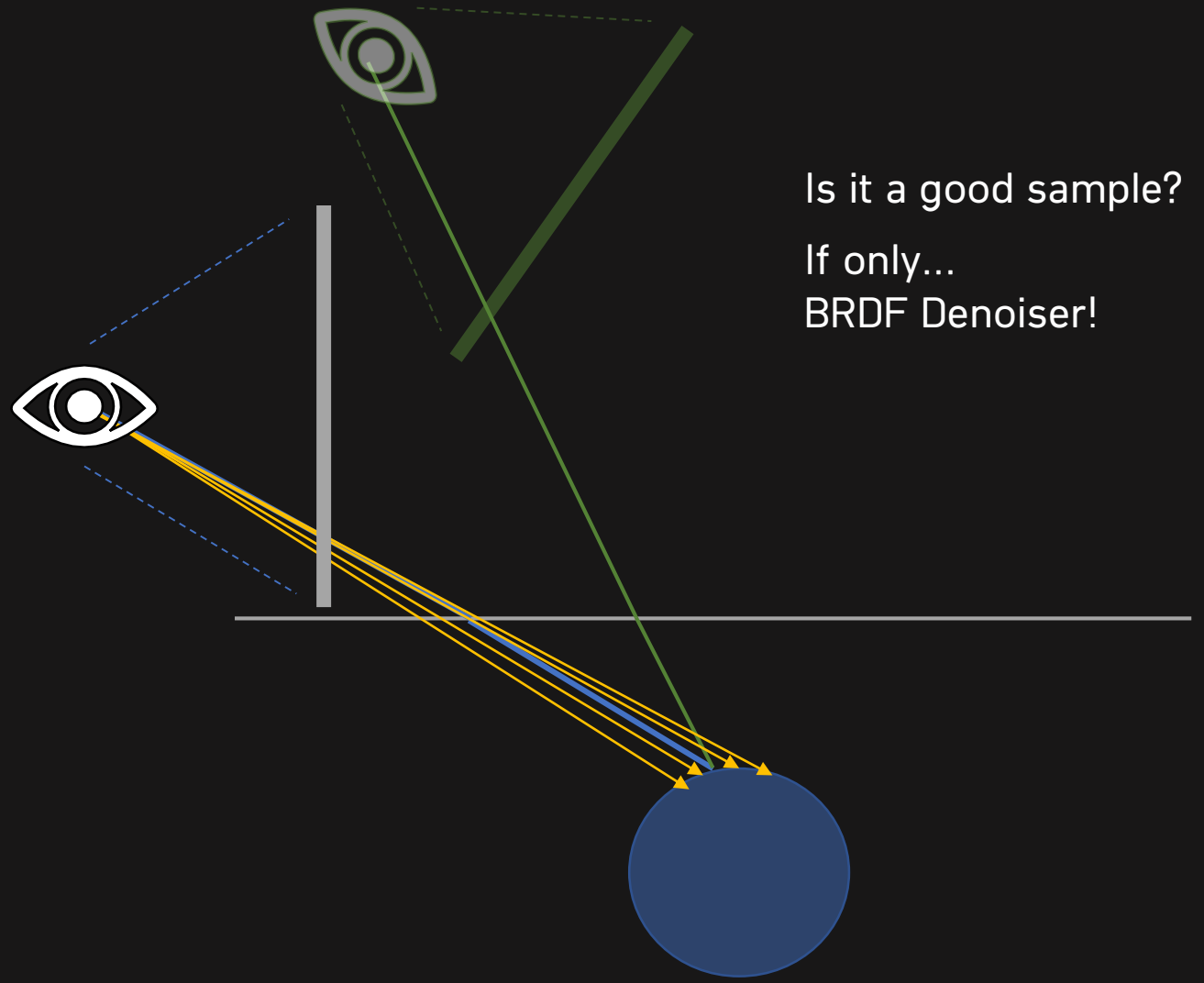
BRDF DENOISE FILTER



BRDF DENOISE FILTER

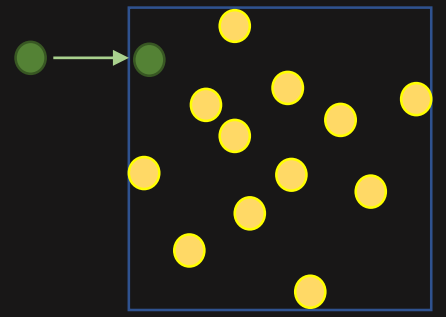


TEMPORAL DENOISE FILTER



Is it a good sample?

If only...
BRDF Denoiser!



TEMPORAL DENOISE FILTER

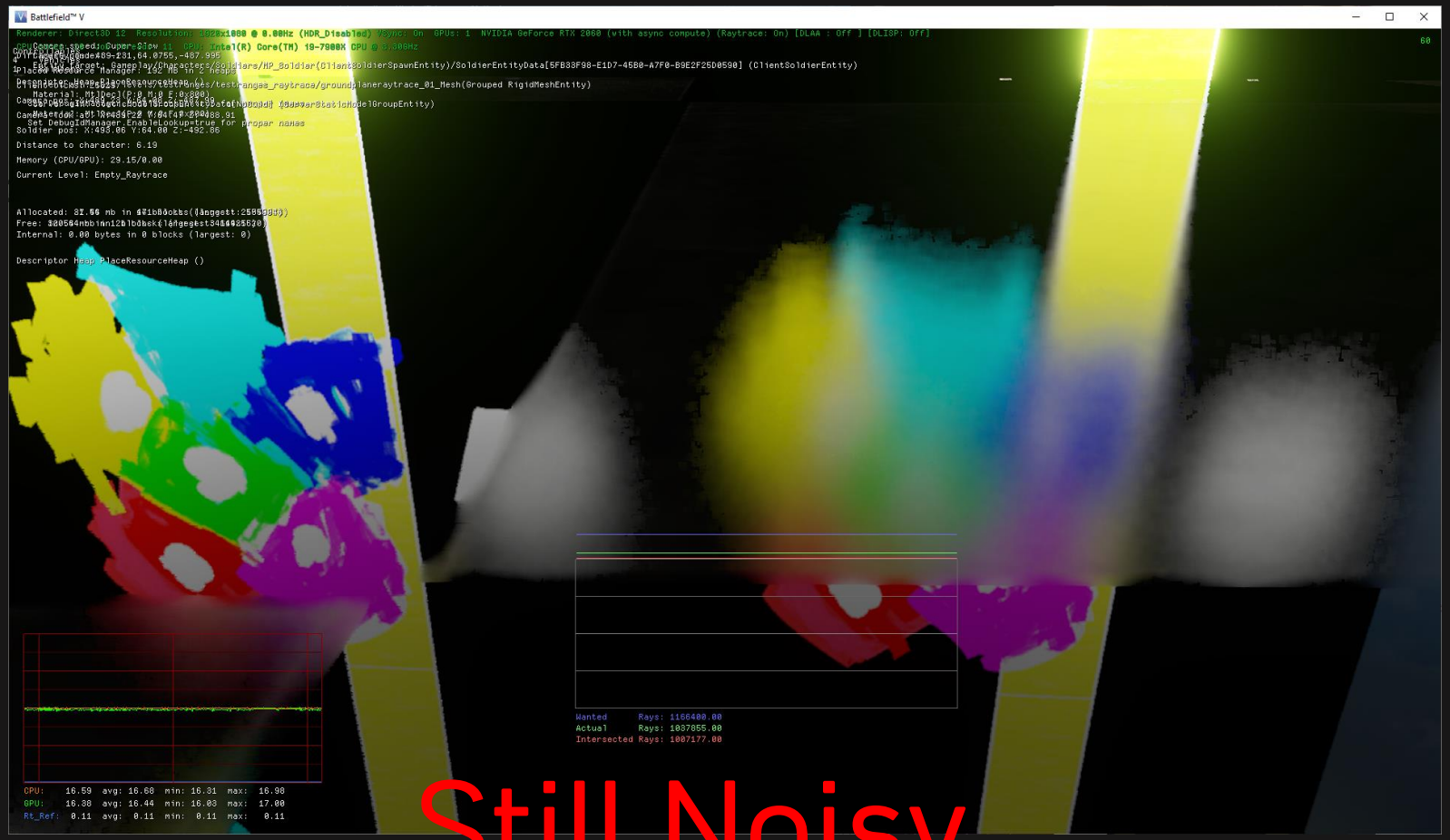
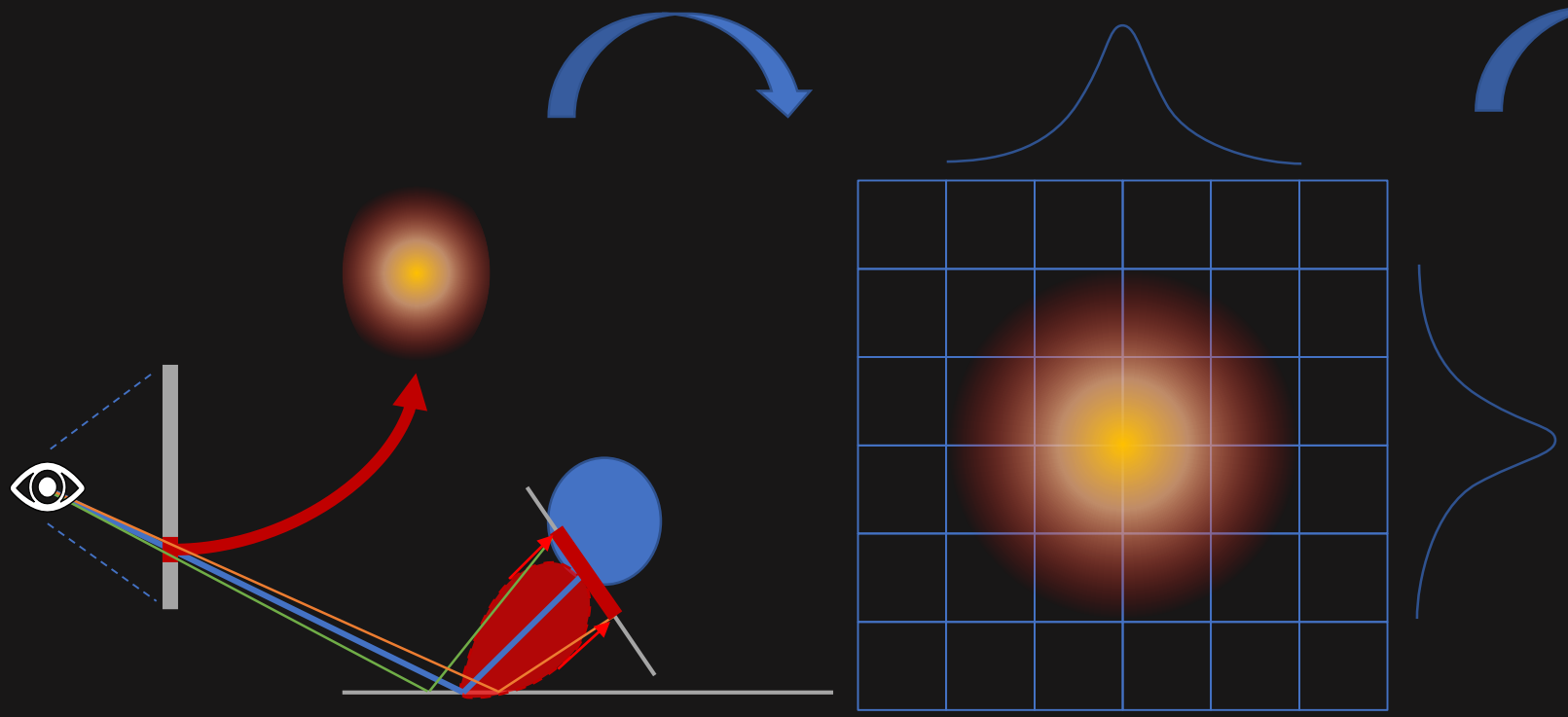


IMAGE DENOISE FILTER



Generate LUT
{ angle, roughness }
to
{ width, height }
for unit length ray

IMAGE DENOISE FILTER

IMAGE DENOISE FILTER

$$\frac{1}{\sqrt{2}} * \frac{1}{\sqrt{2}} =$$

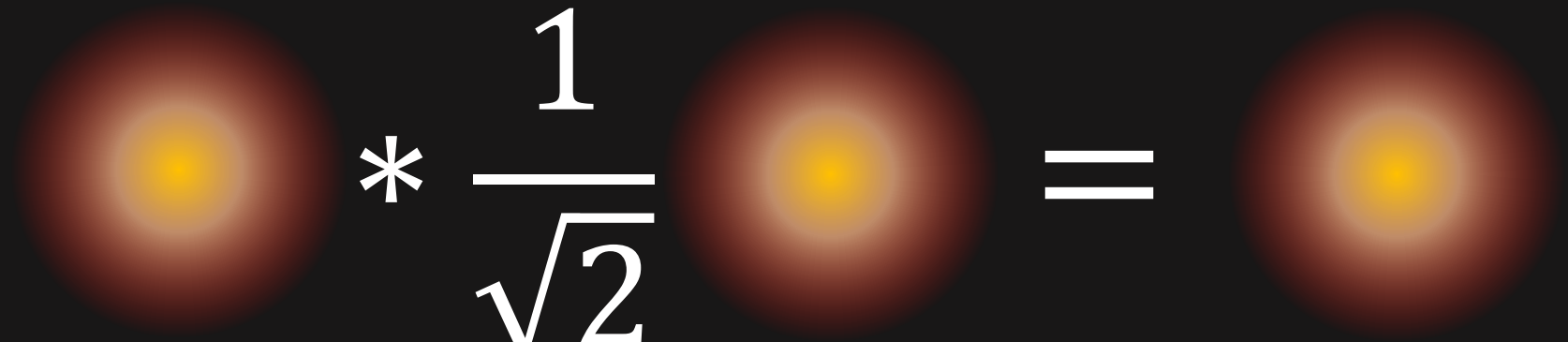
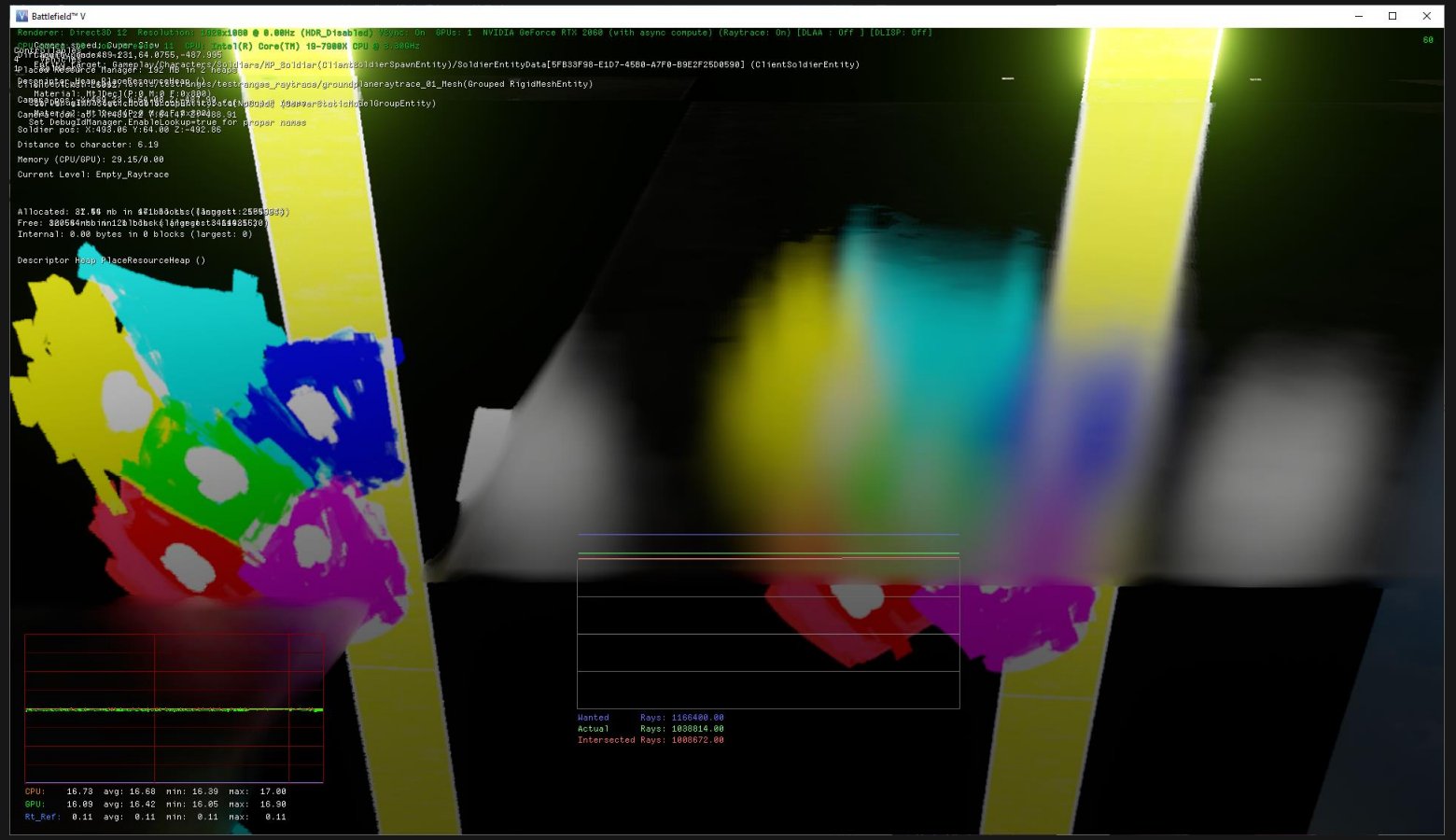
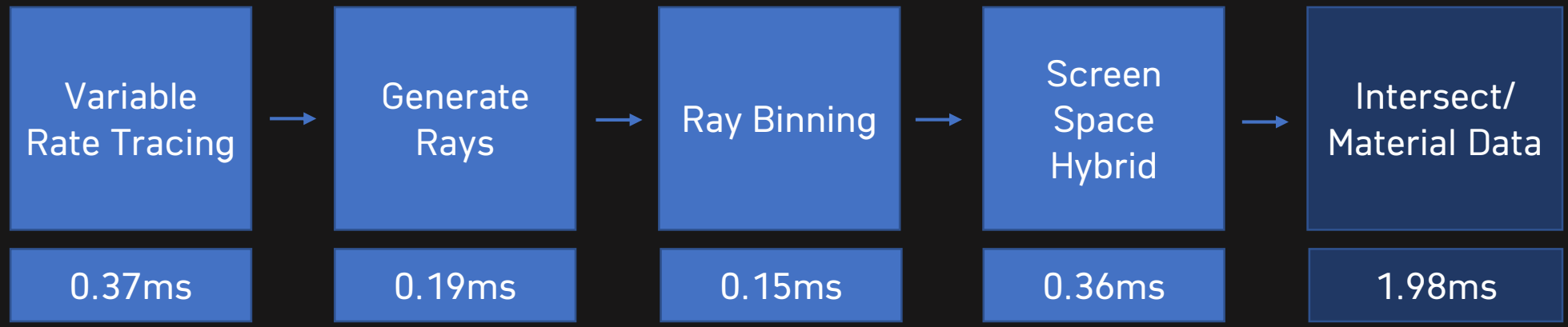


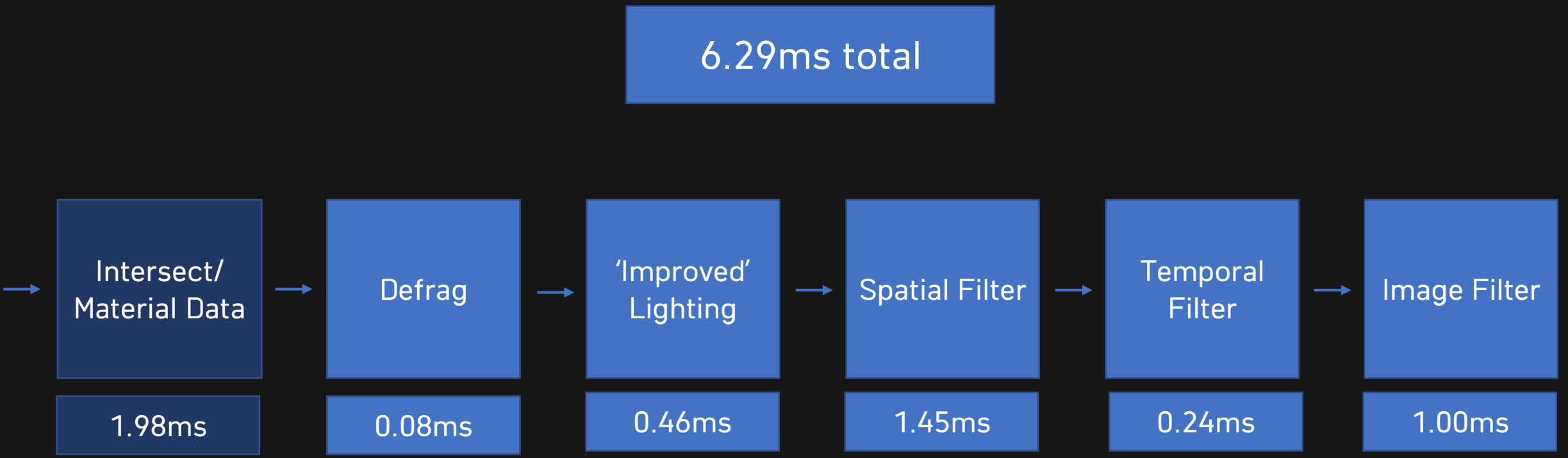
IMAGE DENOISE FILTER



NEW PIPELINE



NEW PIPELINE




BOOGMAN & REEKERS
HOTEL
13 · CENTRAAL STATION
337
DE BITTERE

DXR – a.k.a ”BLACK BOX”

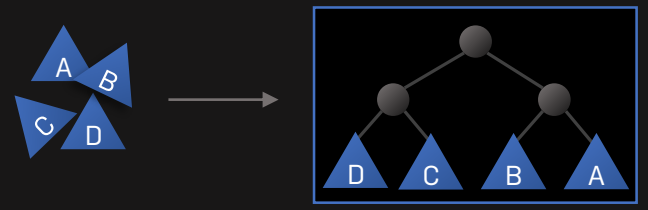
No DXR

Intersection

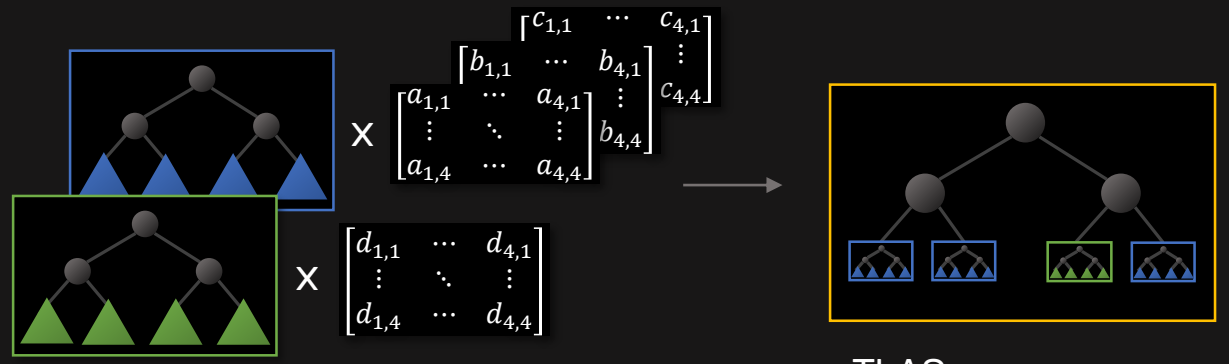
Shading

DXR BASICS

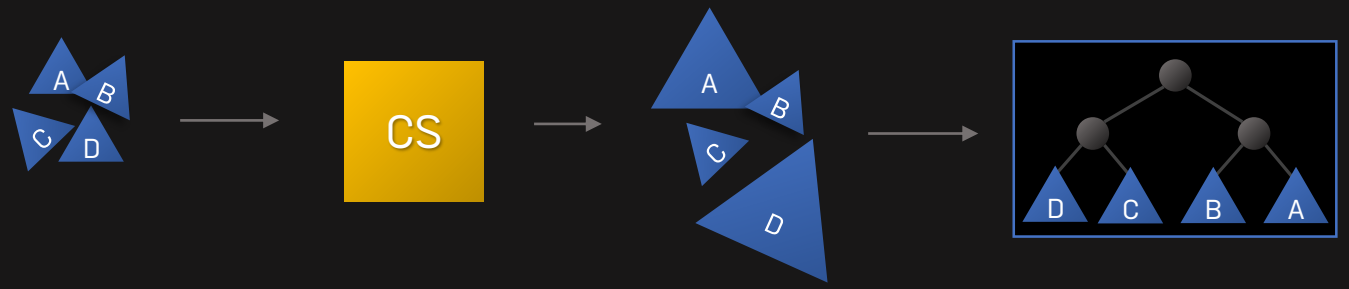
- BLAS - Bottom Level Acceleration Structure
- TLAS - Top Level Acceleration Structure
- CS
  - Skinning, Destruction
  - Compute shader
  - Update each frame
  - Blas can update incrementally



BLAS

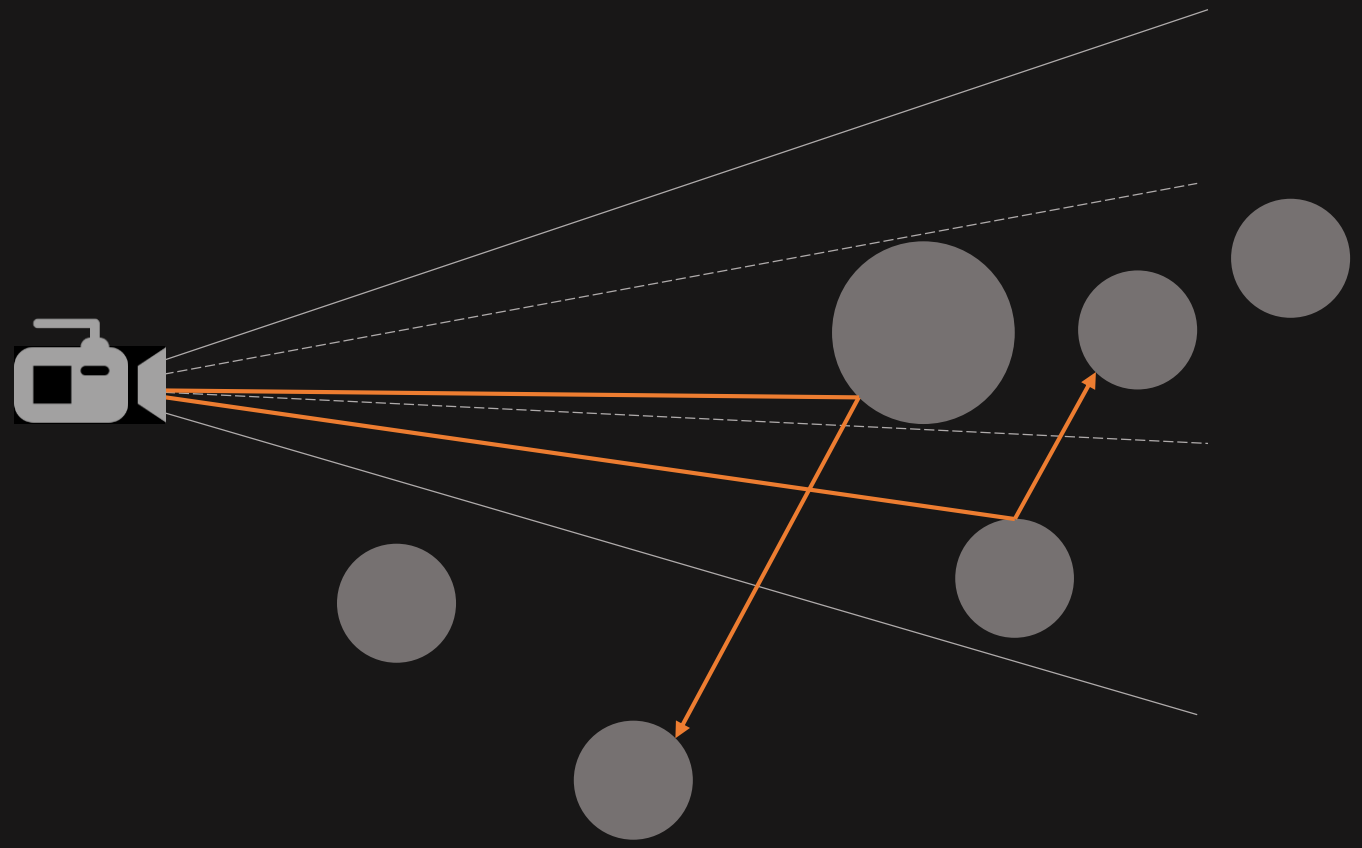


TLAS



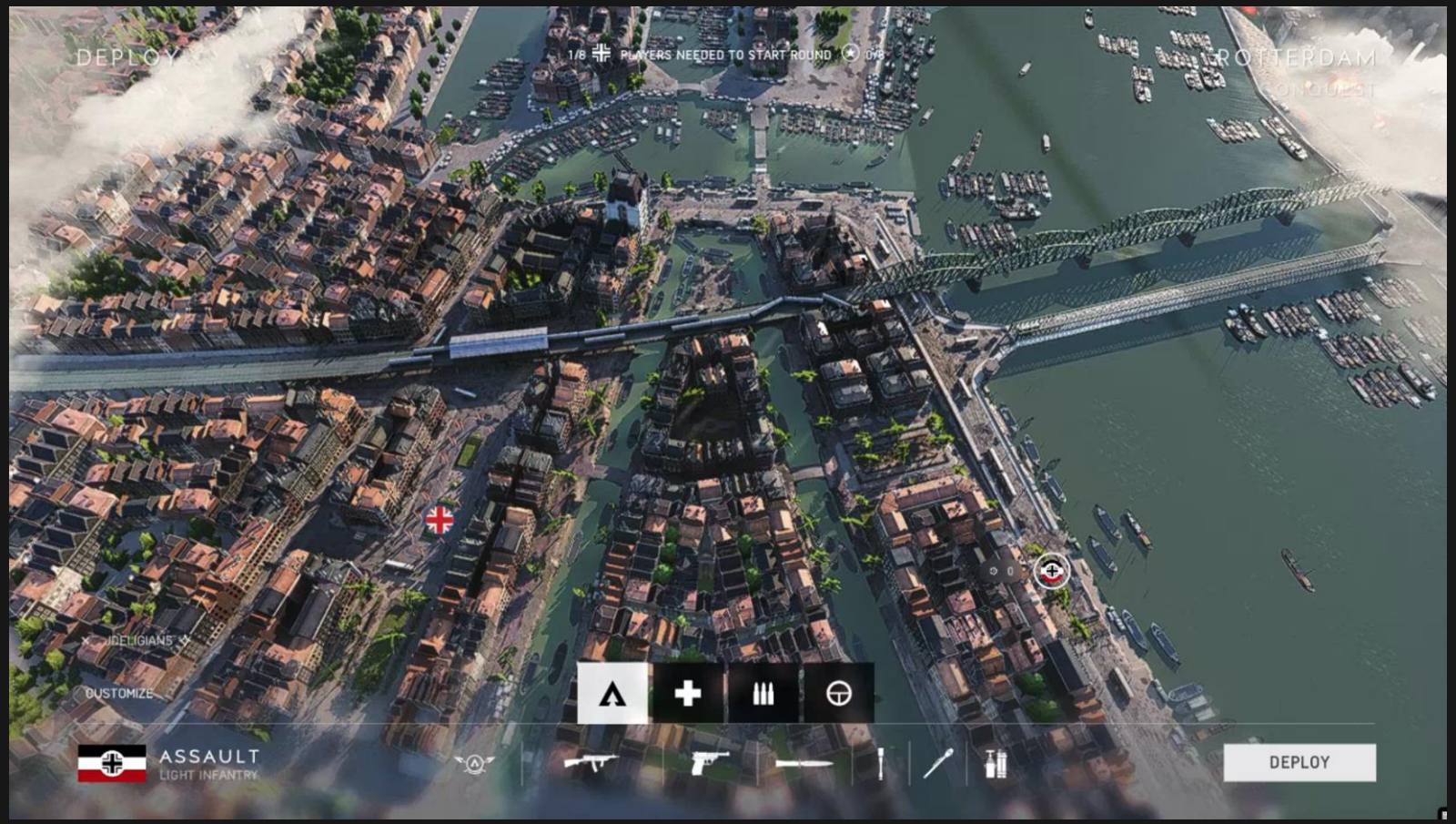
ACCELERATION STRUCTURE

- Which objects?
- Frustum Culling
- Occlusion Culling
- Easy... no culling!



ACCELERATION STRUCTURE – FIRST PASS

- Rotterdam
- 20200 TLAS instances...
- 5000 BLAS rebuilds...
- GPU rebuild 64 ms (!)

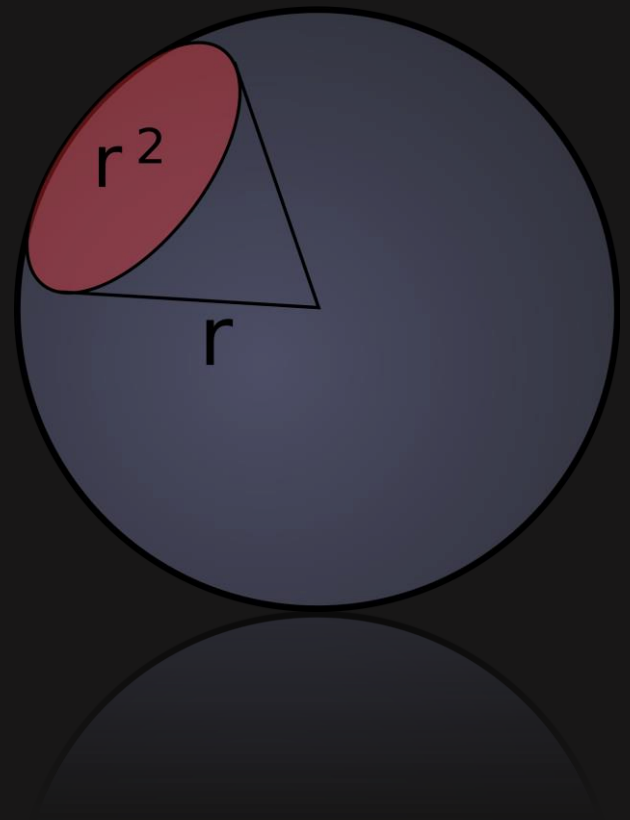


WHAT TO DO?

- Idea: Reduce instance count
- Use a culling heuristic
- Accept (some) minor artifacts

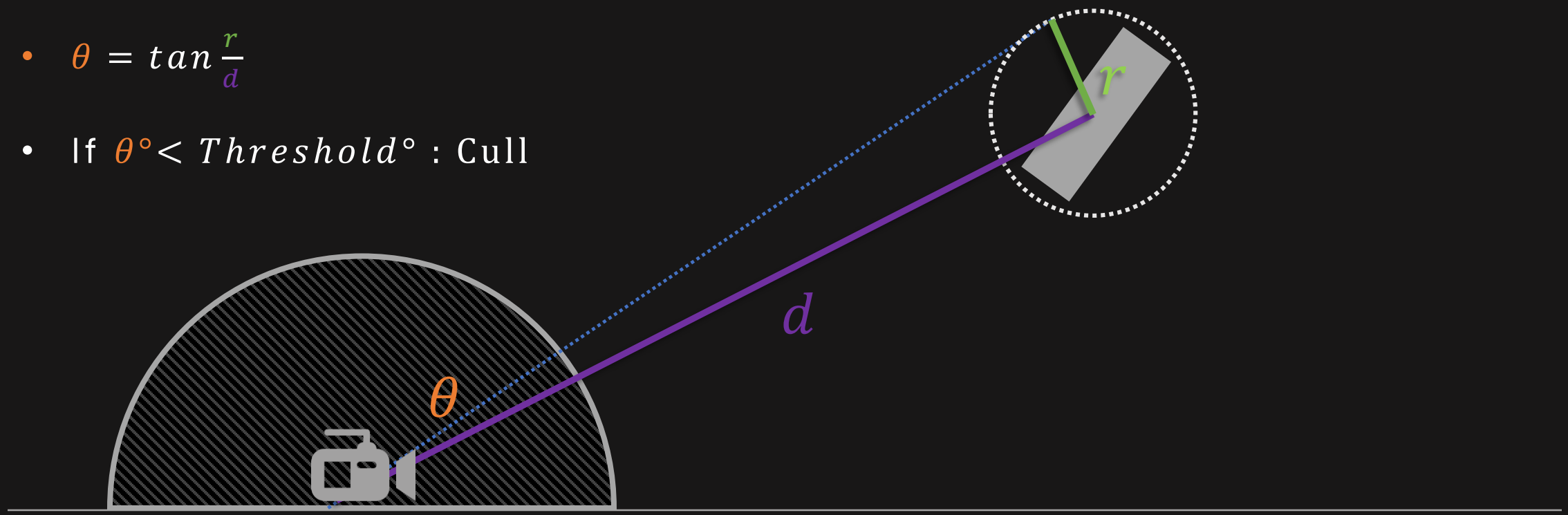
CULLING HEURISTIC

- Assumtion:
  - Far away objects not important
    - Except for large objects
    - Bridge, building etc
- Need some kind of measurement...

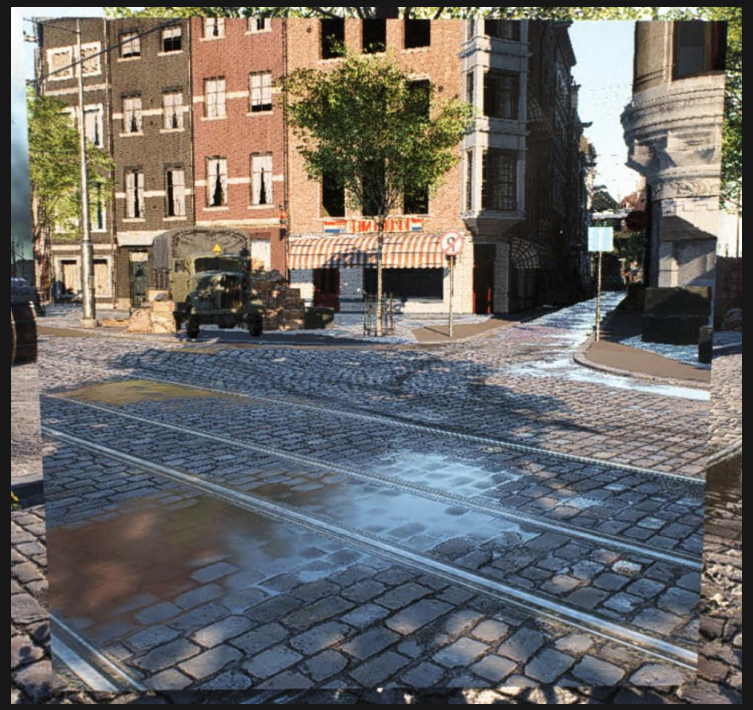


CULLING

- Project bounding sphere
- $\theta = tan\frac{r}{d}$
- If $\theta° < Threshold°$: Cull

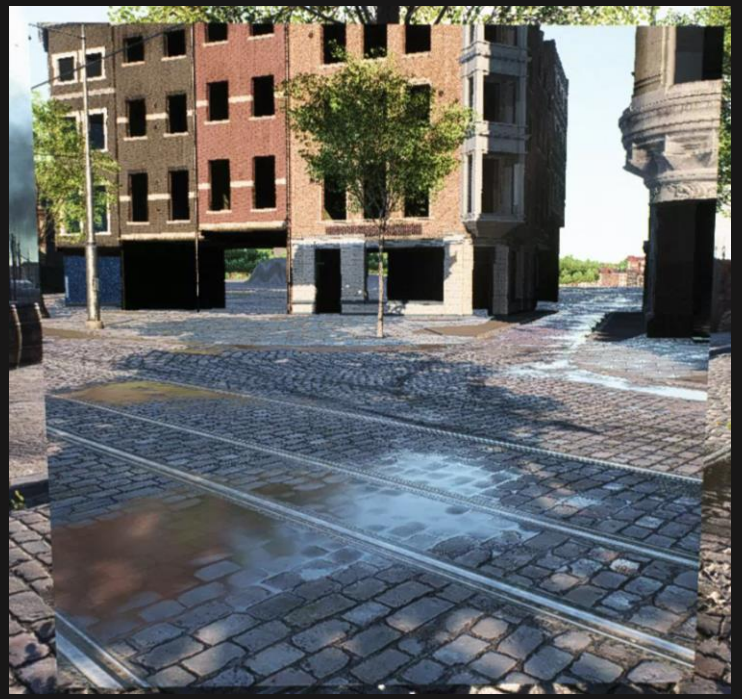


CULLING



$reference\ -no\ culling$

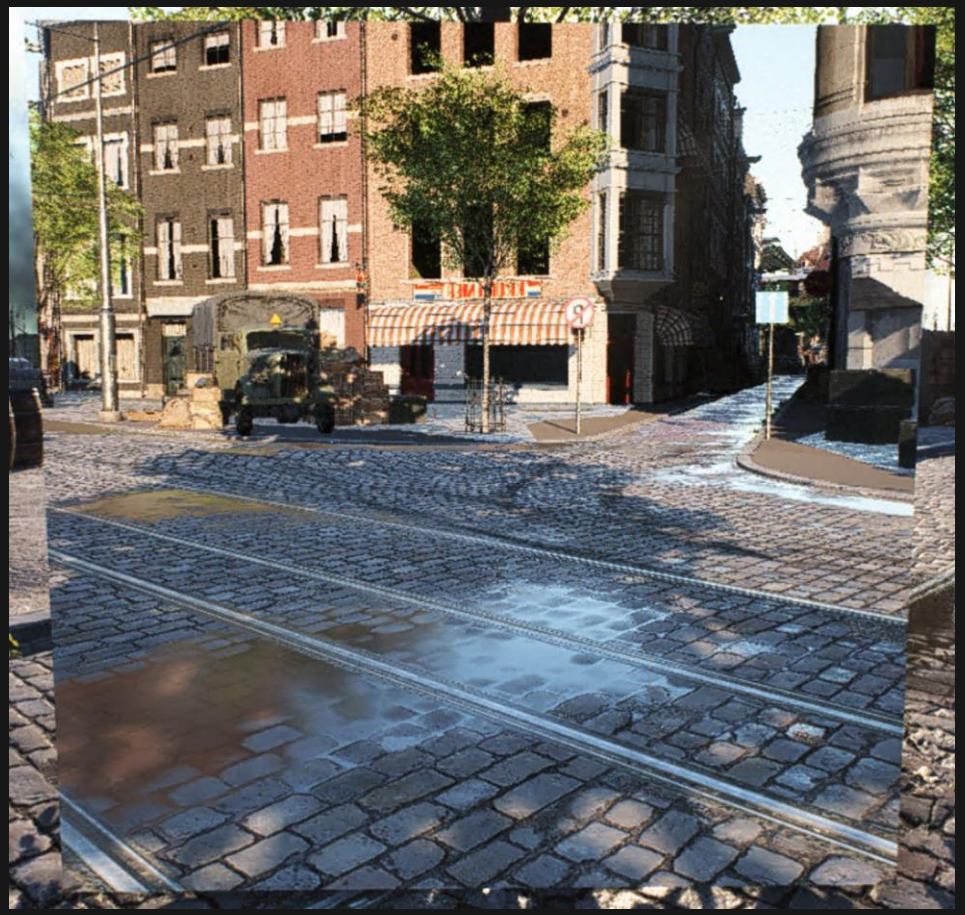
θ = 4°



θ = 15°

CULLING

Culled Objects



reference – no culling

$\theta = 4°$

CULLING - RESULTS

- 4 deg culling
- 5000 -> 400 BLAS rebuilds each frame
- 20000 -> 2800 TLAS instances
- TLAS + BLAS build (GPU): 64 ms -> 14.5 ms
- Pros
  - Faster
- Cons
  - Occasional popping
  - Missing objects

BLAS UPDATE OPTIMIZATIONS

- Still expensive! More ideas:

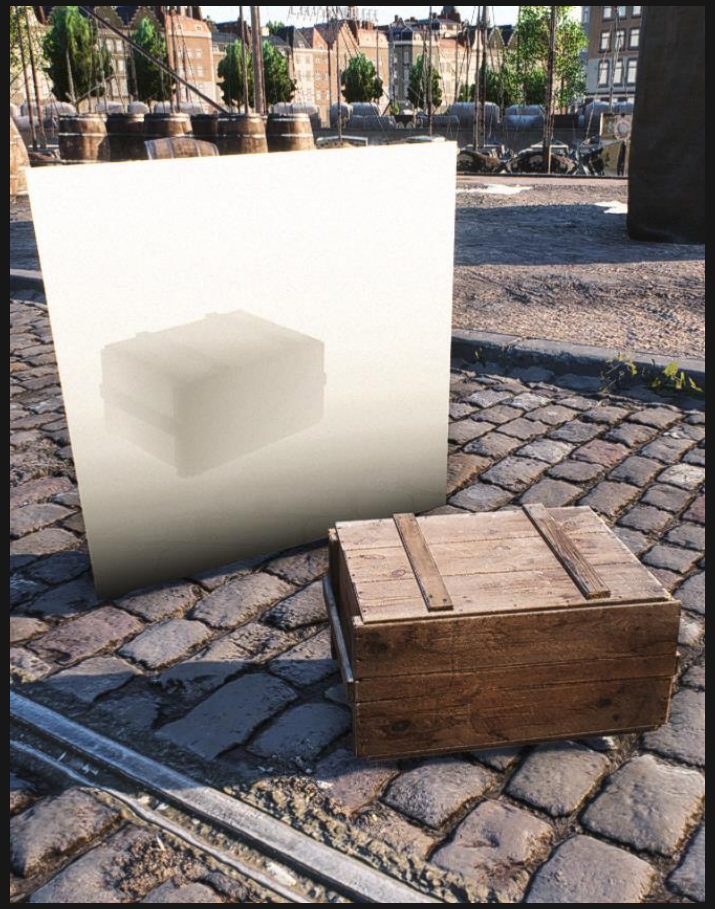
1. Stagger full and incremental BLAS rebuild
    - N frames incremental before full rebuild
2. D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_PREFER_FAST_BUILD
3. Avoid redundant rebuilds
    - Check CS input (bone matrix)
    - 400 -> 50

- Overlap BLAS update with GFX
    - Gbuffer, shadowmaps

RESULTS

- TLAS + BLAS build (GPU): 14.5 ms -> 1.15 ms
- RayGen (GPU): 0.71 ms -> 0.81 ms (staggered refit + flags)
- Much better ☺

SHADING (OPAQUE)

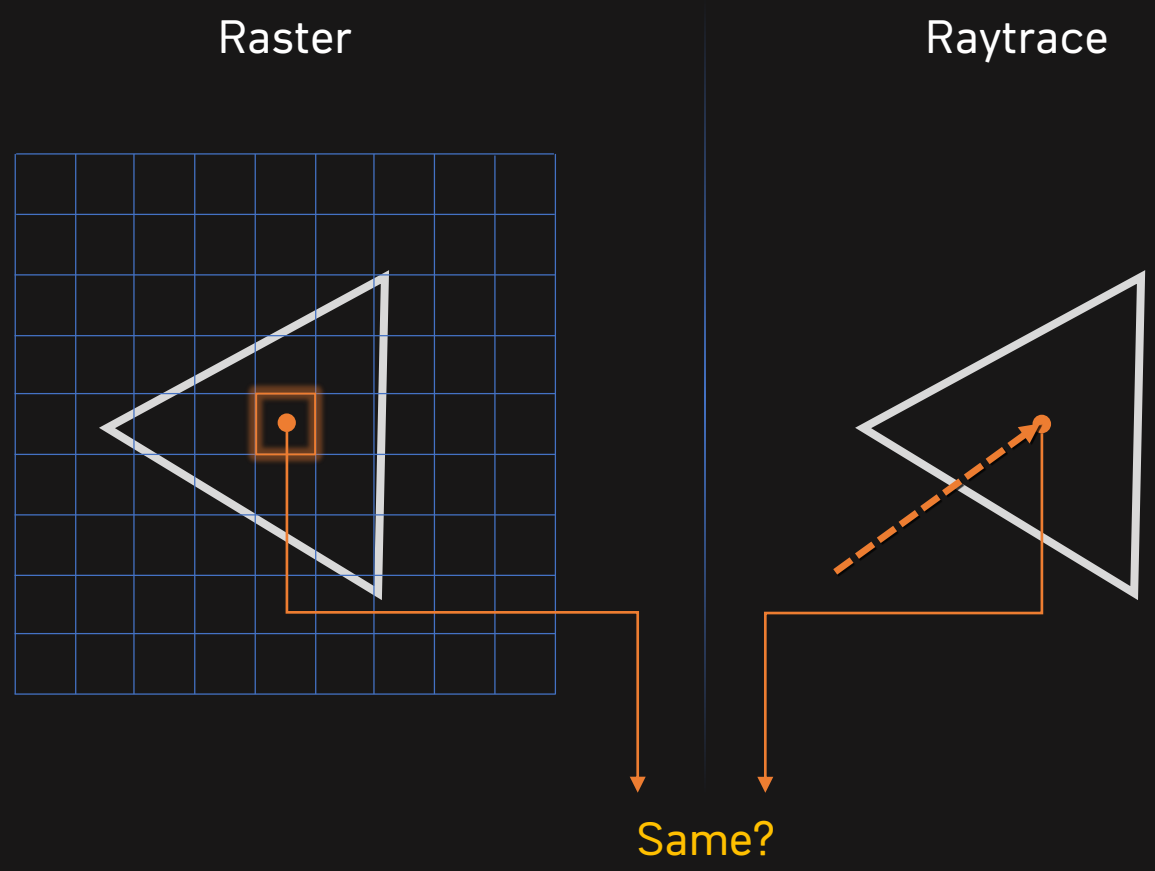


RT ON | SHADING OFF

RT ON | SHADING ON

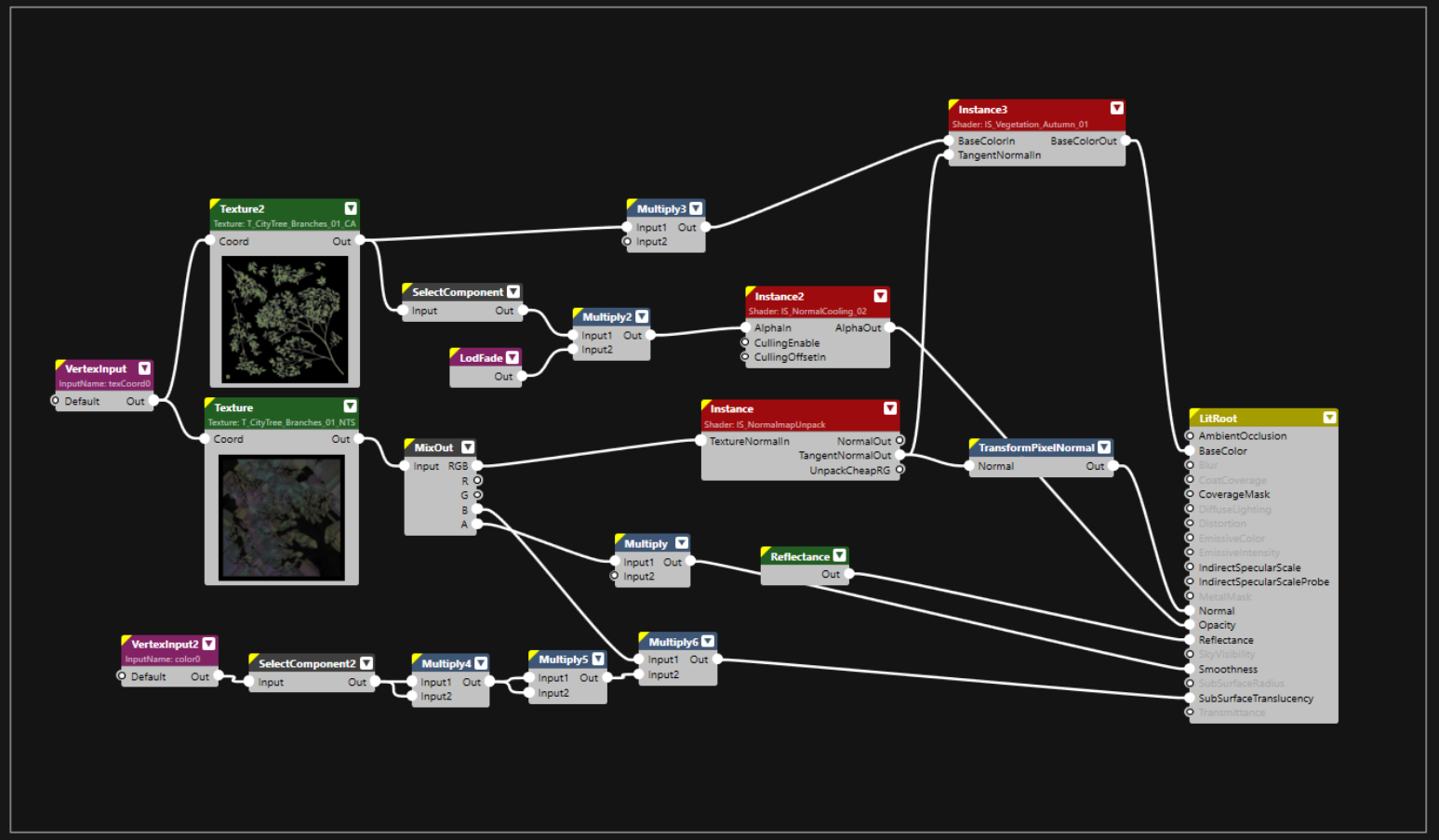
RAYTRACING REQUIREMENTS

- Shader output must match!
- ClosestHit Shader
- AnyHit Shader

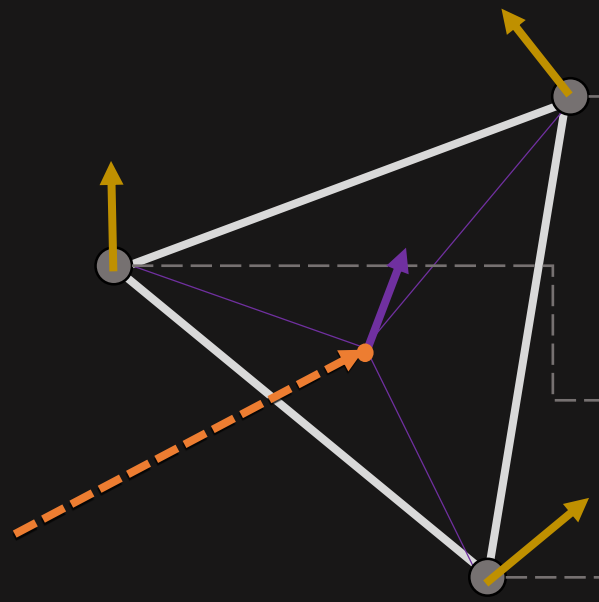


SHADERS IN FROSTBITE

- VS – Handwritten
- PS - Shader Graphs
  - Graph -> .hlsl
- Manual conversion… no
  - 1000s of shaders
- Auto VS + PS to HitGroup



HIT SHADER TEMPLATE



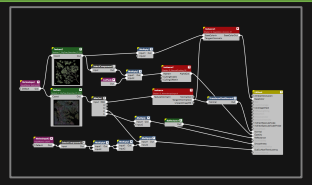
```
[shader(”closesthit")]
void chMain()
{
    IA0 = iaMain(id + 0)
    IA1 = iaMain(id + 1)
    IA2 = iaMain(id + 2)
    V0 = vsMain(IA0)
    ...
}
```

Vertex buffer
UV, Normal

VS – VertexFragment
World Space Normal

#define Sample(s, uv

PS –



...clip?

Texture MIP?

```
#define ddx(x) x
#define ddy(x) x
... ¯\_(ツ)_/¯
```

... ddx/ddy?

ALPHA TESTING

- AnyHit Shader:
- If (AlphaTest(alphaValue))

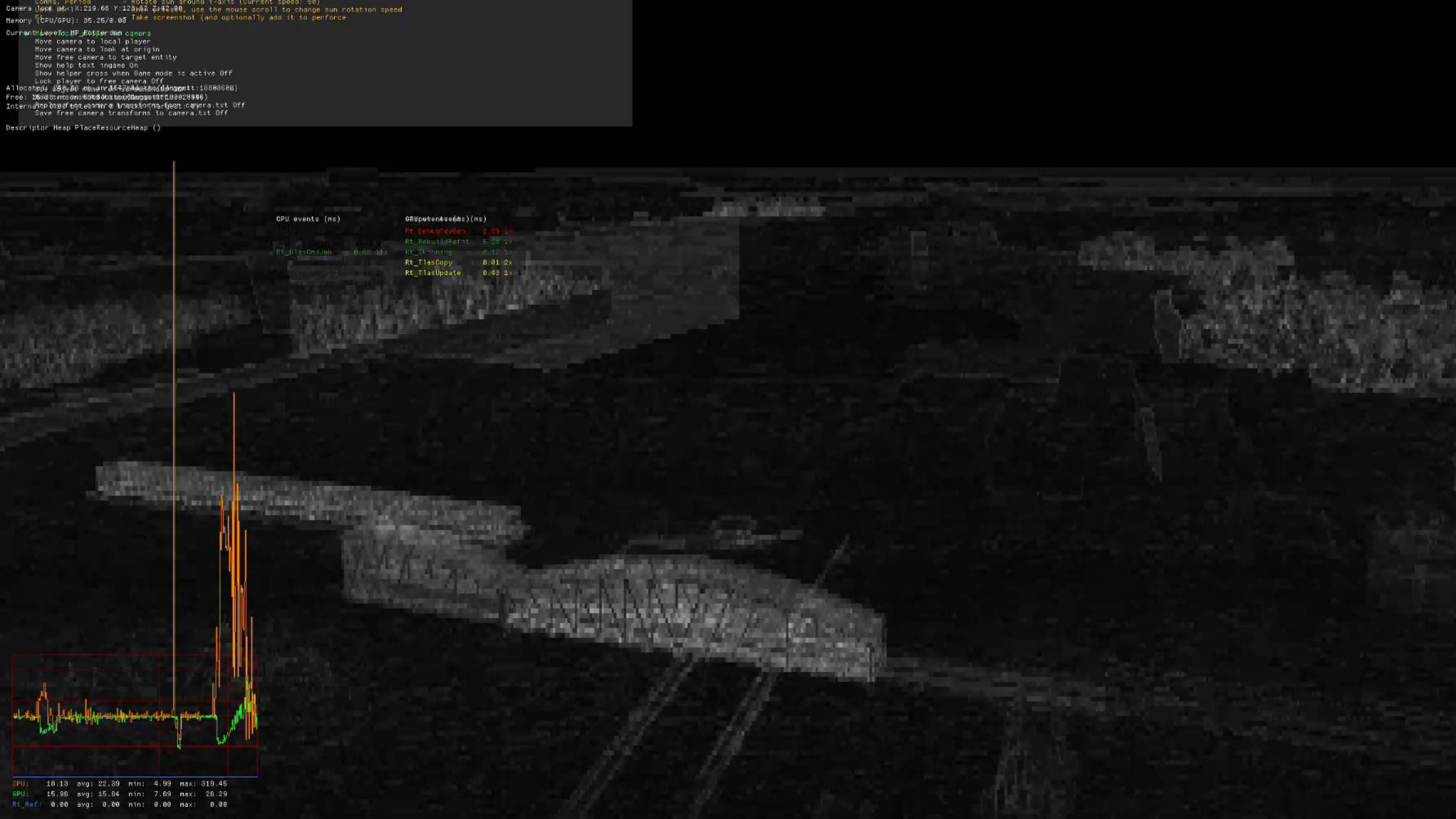
```
IgnoreHit();
```

ALPHA TEST

ANY HIT OFF

ANY HIT ON

ALPHA TEST


Memory (CPU/GPU): 35.25/0.00
Take screenshot (and optionally add it to perforce
Move camera to local player
Move camera to look at origin
Move free camera to target entity
Show help text ingame On
Show helper cross when Game mode is active Off
Lock player to free camera Off
Save free camera transforms to camera.txt Off
Descriptor Heap PlaceResourceHeap ()
CPU events (ms)
Rt_TlasCopy 0.01 2x
Rt_TlasUpdate 0.43 1x
CPU: 18.13 avg: 22.39 min: 4.99 max: 319.45
GPU: 15.96 avg: 15.04 min: 7.69 max: 26.29
Rt_Ref: 0.00 avg: 0.00 min: 0.00 max: 0.00

SUMMARY OPAQUE

- Closest Hit Shader
  - Always
- Any Hit Shader (Optional)
  - Alpha tested
- Compute Shader (Optional)
  - Skinning, destruction etc

RAY PAYLOAD

- Payload: returned on ray intersection
- Same format as Gbuffer RTV
- Contains Material Data
  - Normal
  - Base Color
  - Smoothness
  - ...etc

```
struct GbufferPayloadPacked
{
   uint data0;  // R10G10B10A2_UNORM
   uint data1;  // R8G8B8A8_SRGB
   uint data2;  // R8G8B8A8_UNORM
   uint data3;  // R11G11B10_FLOAT
   float hitT;    // Ray length
};
```

VERIFYING CORRECTNESS

1. Rasterizer output
2. Shoot primary rays in to scene
3. Compare Payload with Gbuffer
4. Non zero output? Bug!
5. Fix bug

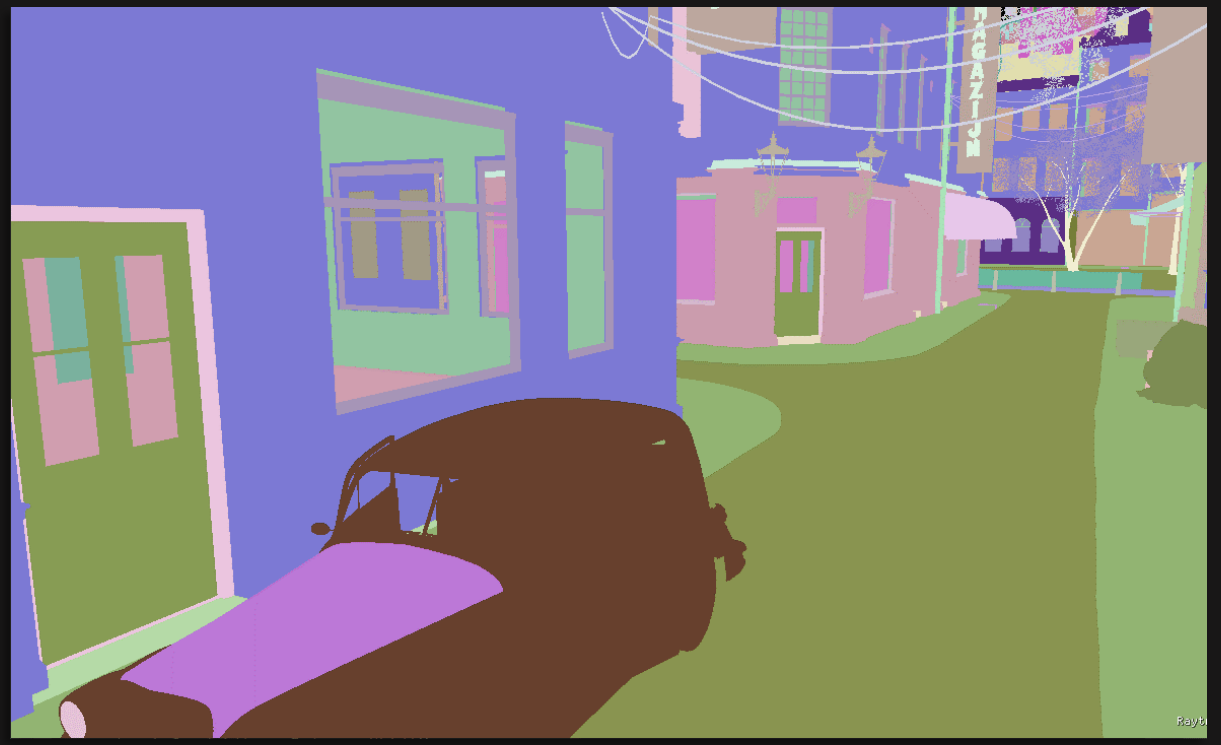
Gbuffer (BaseColor)
Reference

Raytraced (BaseColor)
Primary Rays

Delta

SHADER COMPILATION

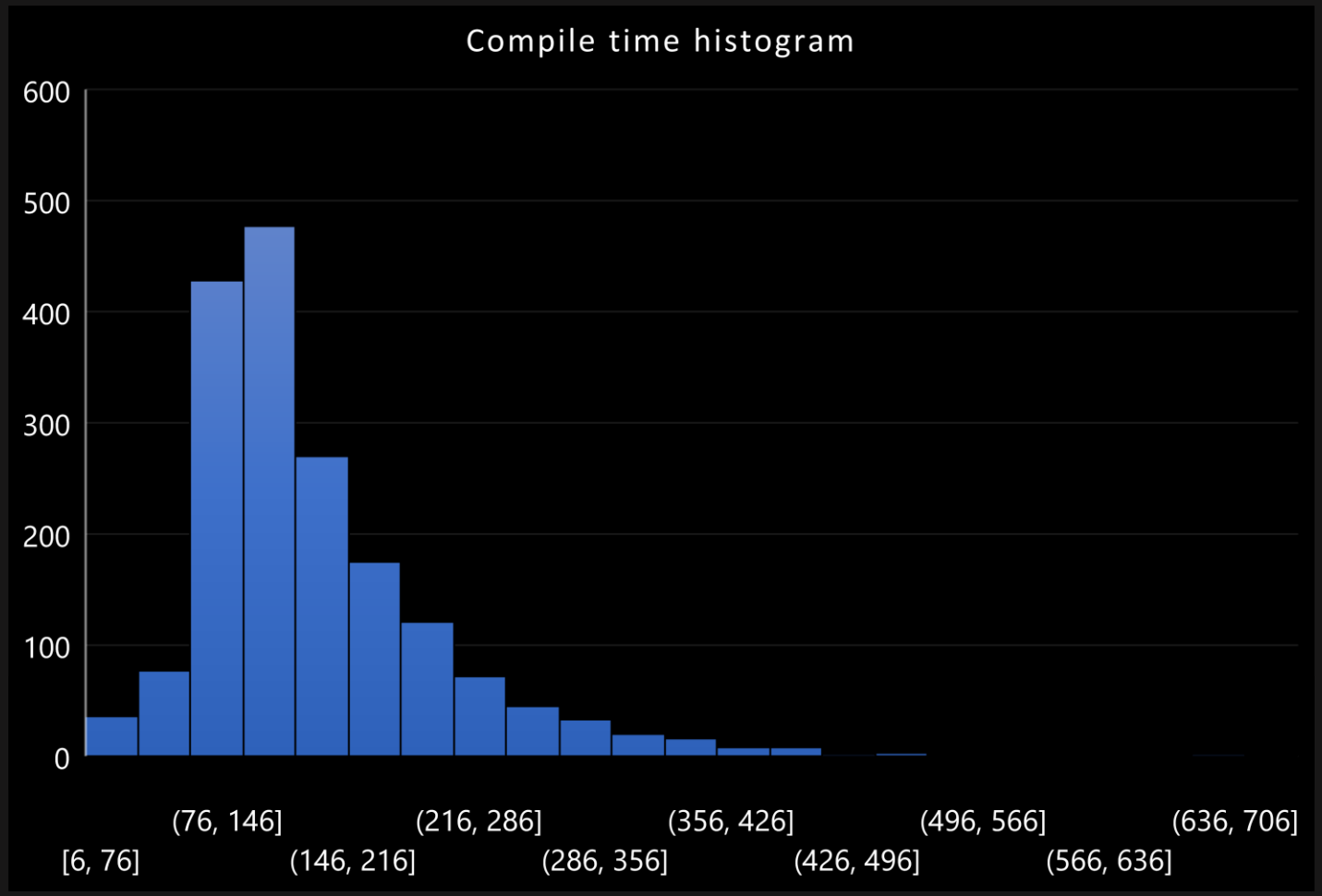
- All shaders generated ☺
- ~3000 per level
- ~250 per frame
- Single RT PSO
- Runtime compile times?



Color coded Closest Hit Shaders

PSO GENERATION

- Dx12 GFX PSO...
- ... DXR 3000 shaders ☹
- Compile times?
  - Majority > 100ms
- Cold cache
  - 7 min 30 sec thread time
  - 6 threads: 1 min 30 sec
- Warm cache
  - 1 min 30 sec thread time
  - 6 threads: 15 sec

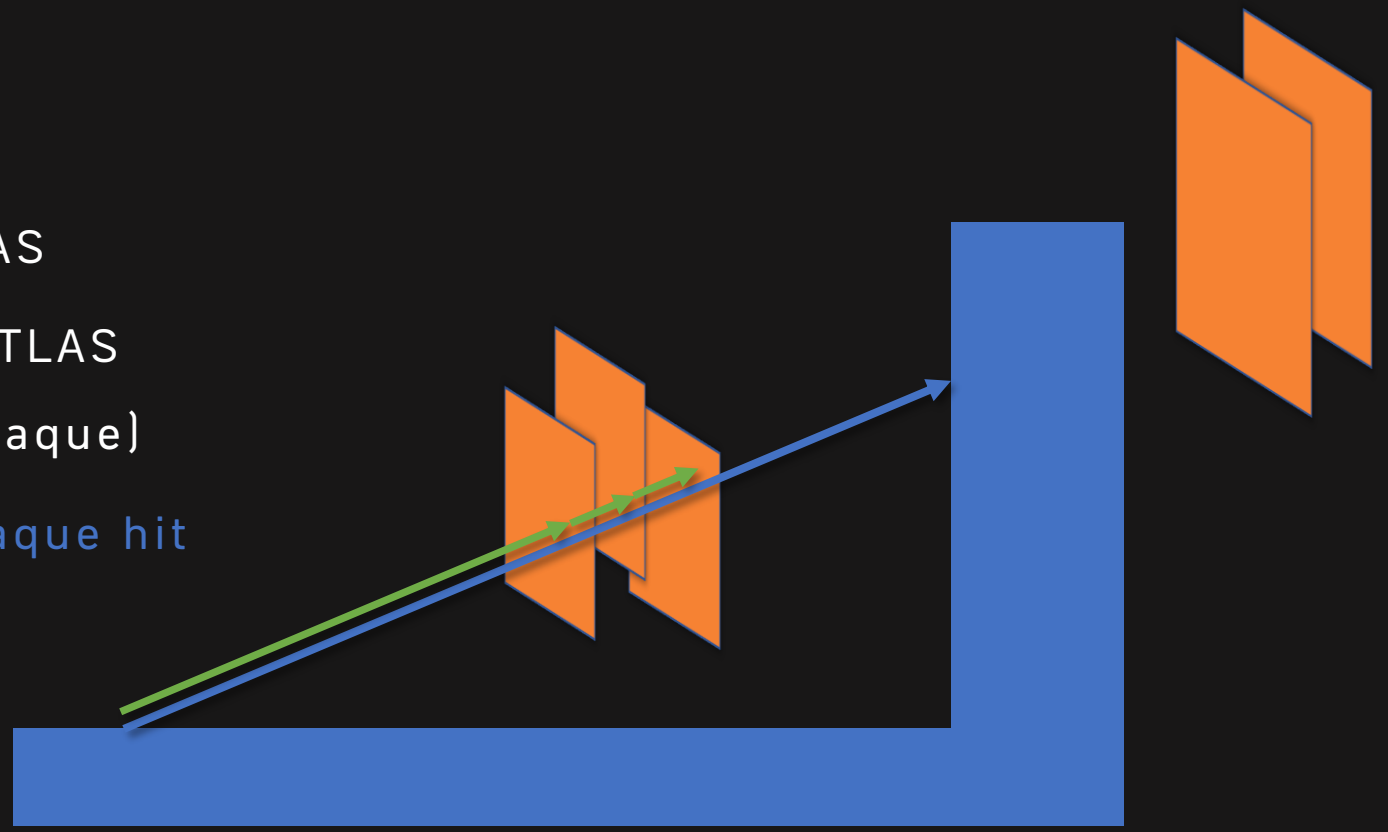


PARTICLES

Smoke, Fire and Exposions. Important elements in BFV!

PARTICLES

- Particle = Transparent+Billboard
- Basic algorithm
  1. Shoot ray in Opaque TLAS
  2. Shoot again in Particle TLAS
     (Max ray length from Opaque)
  3. Blend particles with opaque hit



THE PROBLEM WITH PARTICLES

- Camera aligned billboards ☹
- Rotate odd particles 90 deg around Y?

Billboards visible when viewed from the side.

ROTATED BILLBOARDS

Before: Billboards visible in reflection

After: Rotating odd quads produces a more volumetric look

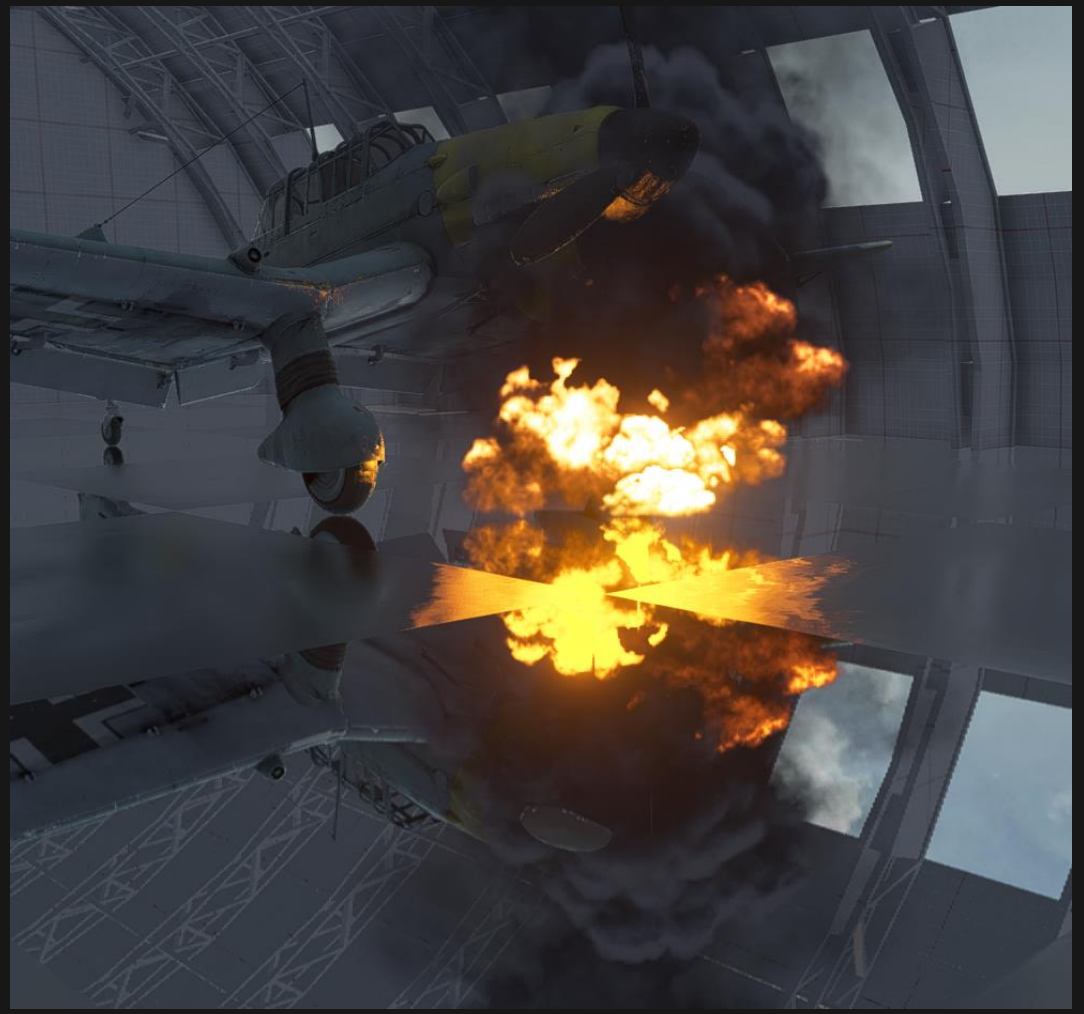
PERFORMANCE

- Accumulate intersections along ray
- 1 rpp => N rpp ☹
- RayGen loop
- Sounds… expensive?

RayGen Shader

```
*... init ray using opaqueHitT and currT*
for (hitCount = 0; hitCount < MaxIntersectionCount; ++hitCount)
{
  ...

  ForwardPayloadPacked forwardPayloadPacked;
  initForwardPayloadPacked(forwardPayloadPacked);
  TraceRay(g_tlasPartices, 0, 0xFF, 0, 1, 0, ray, forwardPayloadPacked);
  ForwardPayload forwardPayload = unpackForwardPayload(forwardPayloadPacked);
  if (forwardPayload.hitT <= 0.0f) // Miss, tracing done
    break;

  * ... update ray using forwardPayload.hitT, accumulate color, alpha *
}
```

THE (SECOND) PROBLEM WITH PARTICLES



RayGen loop 0.96ms

OPTIMIZING PARTICLES

- ~~Loop~~ Idea: AnyHit shader?
- Same… but different
- Inspired by WBOIT*
  - Weight = max(luminance,r,g,b alpha)
  - Emissive, fire

RayGen Shader

```
... init ray using maxT and currT
TraceRay(g_tlasPartices, 0, 0xFF, 0 , 1, 0, ray, forwardPayloadPacked);
* ... process payload and calculate weighted average *
```

Any Hit Shader

```
struct Attributes { float2 barycentrics; };
[shader("anyhit")]
void main(inout ForwardPayloadPacked payloadPacked, in Attributes attributes)
{
  *... Calculate color, transparency *
  payloadPacked.alpha += alpha * weight
  payloadPacked.color += color * weight;
  payloadPacked.weight  += weight;
  IgnoreHit();
}
```

*Weighted Blended Order-Independent Tranparency:
http://jcgt.org/published/0002/02/09/

PARTICLES - RESULTS

'Naive' Closest Hit 0.96ms
Slow but accurate

Order Independent AnyHit 0.34ms
Really fast, but slightly different look

THANK YOU!

...ANY QUESTIONS?