
Santa Monica Studio


GOD OF WAR
RAGNARÖK

Rendering “God of War: Ragnarök”

Stephen McAuley
Technical Director
Santa Monica Studio

What's My Story?

- Joined Santa Monica Studio in August 2020
- Joined as Lead Rendering Programmer:
  - At the time, the rendering team consisted of six people
  - We grew to ten over the course of the project
- Previously spent nine years at Ubisoft Montreal on the Far Cry franchise
  - Recent experience is in open world games
  - Linear, narrative games are new to me

Santa Monica Studio

What Will We Be Talking About?

- Not just a technical presentation
- This is about my experience on God of War: Ragnarök as a lead programmer
  - Leadership and production methods
  - Technical explanations of rendering features and optimizations

Santa Monica Studio

It's also worth mentioning I joined SMS during the pandemic, so we were all learning about work from home, plus having to get to know each other remotely.

Let's begin at the beginning itself, when I started on God of War: Ragnarok.

Getting Oriented

- What do we need to do to ship the game?

Santa Monica Studio

Getting started on a project is a pretty difficult task, and it can seem insurmountable. But I tried to ask the fundamental question: ”what you need to do to ship the game?”

Now, this is a question you ask upwards, of your managers, which is what I did. But coming on board as a lead I also had to get to know my team.

Getting to Know My Team

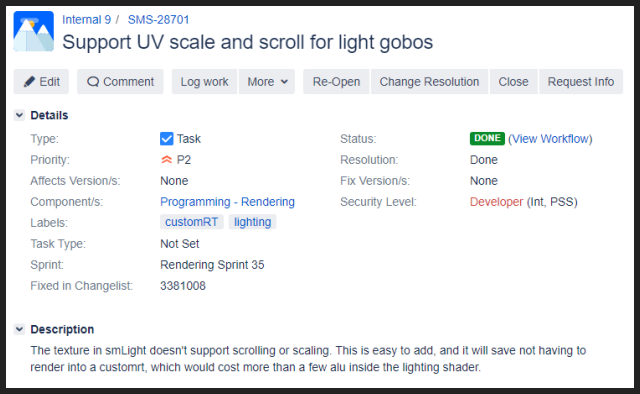
- What are their stories?
- What are they working on currently?
- What would they like to work on in the future?
- What would they like to solve on this project?
- What are their longer term hopes and dreams?

Santa Monica Studio

I actually used very similar questions to these in my initial 1:1s with my team, as I tried to get to know them. There's a mix of questions here, where you try to understand someone personally, the internal aspect of things; but then you want to find out how they're relating to the world, their external interactions. Similarly, you want to balance knowing what they're working on now in the present, but also what their plans and hopes are for the future too.

Getting to Know JIRA

- What are the tasks and bugs in JIRA?
  - How do they correlate with what I've been told?

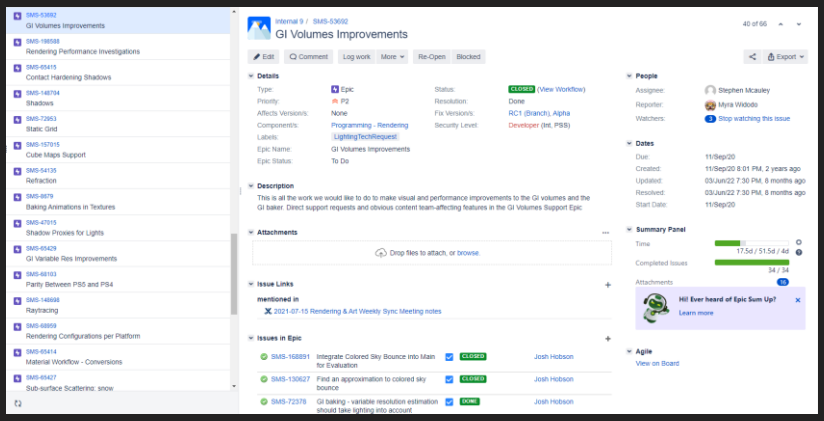


Santa Monica Studio

The next thing I did… which admittedly sounds very boring… is to read JIRA. Extensively. Going through all tasks on the rendering team, some dating back many years, and seeing where they matched with current work, our game features, and also our dreams of improving things for the future.

Organizing my Thoughts (with JIRA!)

- Ensuring our major features had Epics:
  - Then bucketing tasks accordingly

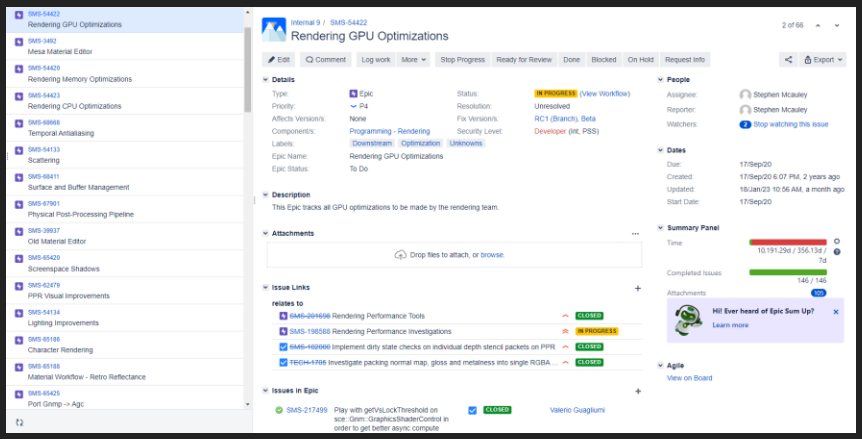


Santa Monica Studio

The next step was one of organization. Once I had a bit of a grip on the major features we wanted to develop, I ensured we had Epics for each feature and started bucketing tasks for each of them. This personally really helps me organize my thoughts, as I have a clear list somewhere of what the rendering team needs to deliver. It also helped me keep track of all the work that was already there and understand where it fitted in (or in some cases, where it didn't).

Organizing my Thoughts (with JIRA!)

- This even applied to future work, such as optimizations:



Santa Monica Studio

I even categorized optimizations (and memory savings) really early in the project, partially to organize the database, but also to ensure we had tasks to draw on when we entered that stage of the project.

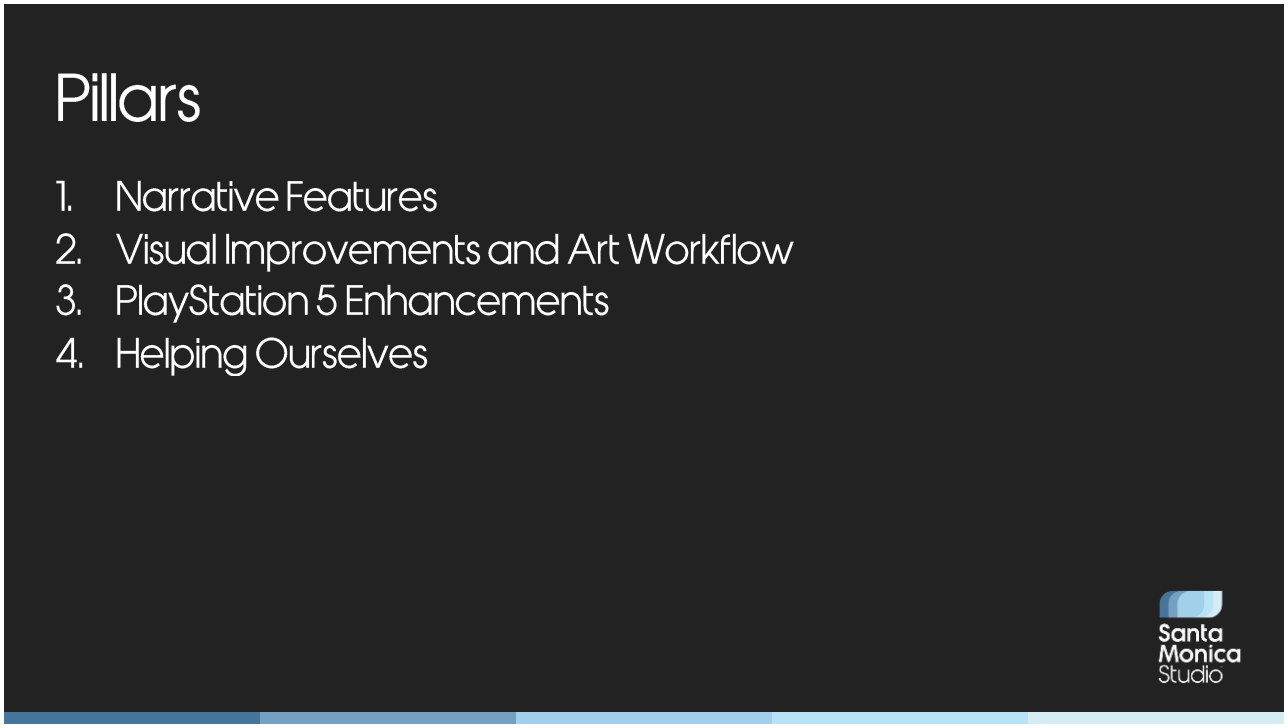
Back to the Original Question

- What do we need to do to ship the game?

Santa Monica Studio

But all this work is in service of the original question: what do we need to do to ship the game? This time though we've got a little closer to understanding it and started to tackle the task in front of us.

Rather than just breaking the work down in my head into features that had to be delivered, I thought about four major pillars.



The first pillar is Narrative Features. These are absolutely the most important thing to do – if we cannot tell the story we want to tell, then the game cannot ship.

Next are visual and art workflow improvements. These are what we do as a rendering team to achieve the quality bar we need.

Then we have PlayStation 5 Enhancements, which were a pillar particular to God of War: Ragnarök. We didn't just want to improve visuals overall, but as a cross-platform title we wanted to work on particular enhancements for the PS5.

The final pillar is about Helping Ourselves. These are things that we want to do as a rendering team to help us work better. That could include improving compile times, refactoring code or writing debugging tools.

I'm going to walk you through each pillar to explain some of the work we did for each, diving into details where appropriate.

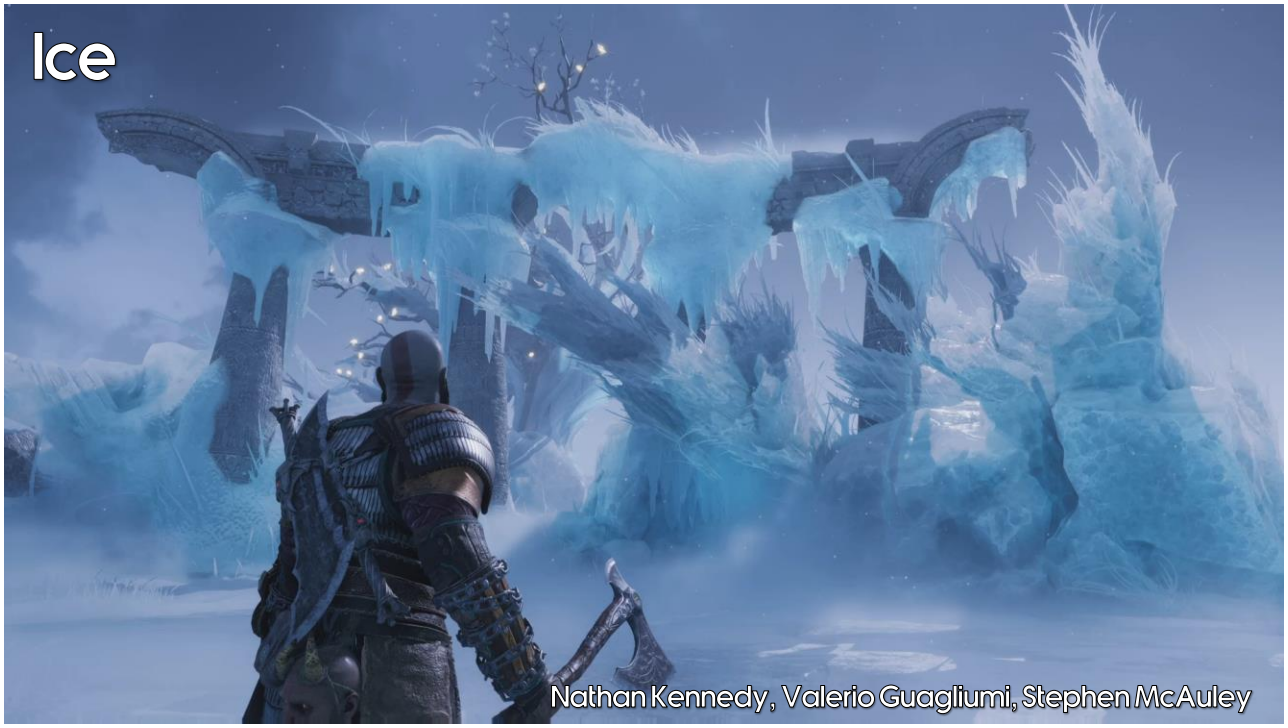
Let's start then with narrative features. At SMS we are an incredibly narrative-driven studio. We need to be able to tell a story, and thus as a rendering team our first priority has to be ensuring we have the features needed to do that.

I'm going to run through the most important of these features, but this isn't a section I'm going to go into technical detail on.

The first feature is deep snow. We need to be able to represent Fimbulwinter, so we need lots of snow and the ability for it to react to characters as they walk through. We did have this feature on God of War 2018, but with the technique used there the snow wasn't able to go deep enough.

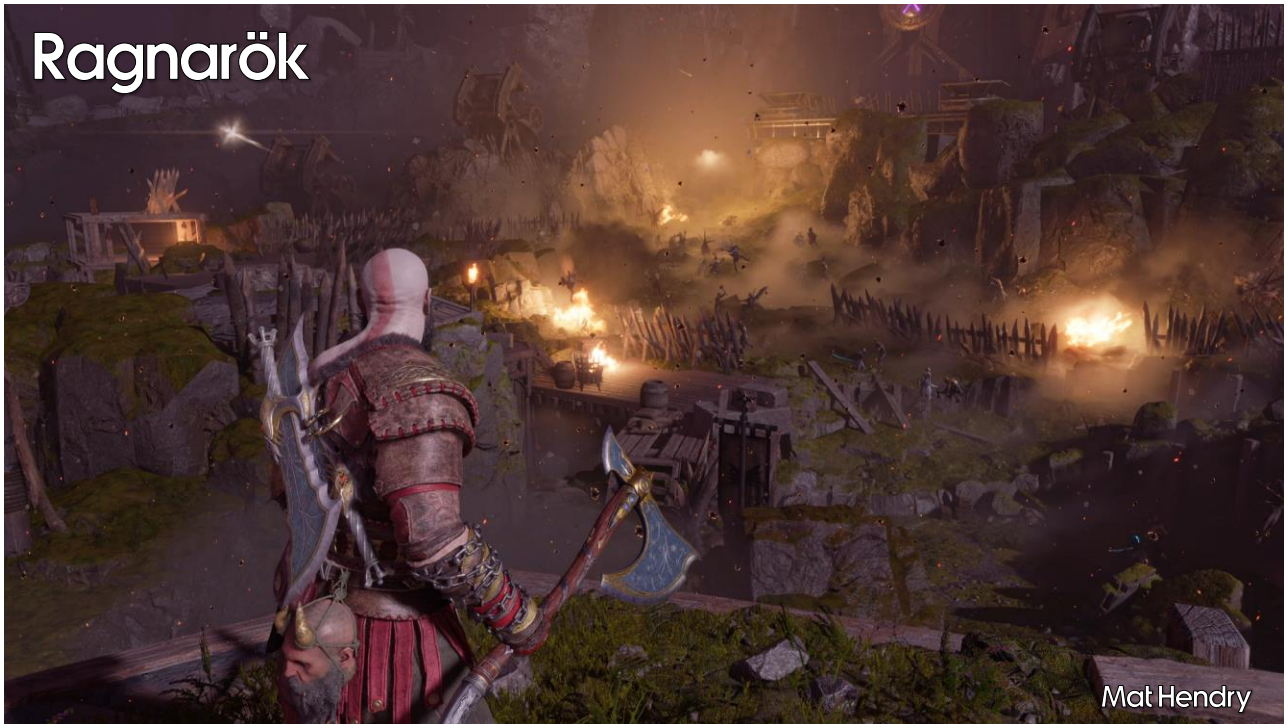
Hopefully, many of you saw Paolo's excellent talk on Monday about this system – otherwise, please reference that if you need more details.

Also, as you notice, I'm crediting the many excellent people who worked on these features on the bottom right of the screen there.



Obviously when representing Fimbulwinter, we don't just need snow, we also need ice. This is some really fantastic technology that I'm sad I don't have time to talk more about today, but hopefully in the near future.

We also had the ability to change the time of day from day to night in Vanaheim, represented by this amazing cinematic. Time of day is a little new to us, so we had Bryan working on animating the global illumination, and Jon and Amy working on the dynamic sky that you see.



Finally, there is Ragnarök itself. The challenge here was having so many animated characters on screen to represent the battle. In the far field we relied on flipbooks, in the near field fully animated characters, but in the middle ground we prerecorded animations into buffers and played those in a loop on the relevant meshes, saving any CPU animation cost.

You might have thought I raced through that first section. And that's true, but it's also reflective of my experience. Many of these features and goals were predetermined before I joined, and my job was just to shepherd them to completion. This next topic though, of visual improvements and art workflow, was where I first started to get my hands dirty and try to make a difference on the project.

Let's use a cricket analogy… (but if you're more familiar with baseball, that analogy works equally well).

In cricket, we say that it's not the batters who win the game, it's the bowlers. The batters can set up the win for sure, but at the end of the day you need to get 20 wickets to win, and only the bowlers can do that.

In game development, I view the programmers as the batters and content creators as the bowlers. Sure, as rendering programmers we can produce amazing technology, but that's just giving our artists an easier platform to go out an achieve an amazing looking game. That's why I include improving art workflows alongside visual improvements – it's not just about better technology or visual features, it's also about helping our artists achieve higher visual quality faster.

[Photo attribution: Robert Drummond, https://www.flickr.com/photos/47229455@N04/36141616805/, CC0 license]

Art Workflow Concerns

- Lighting
- Reflection and refraction

Santa Monica Studio

I want to talk through two major art workflow concerns we dealt with. The first, lighting, was one of the first things I identified as a problem on the project – where the goal was to save the time of lighting artists. The second concern, how we set up reflection and refraction, came later in the project, and the goal here was to save the time of environment artists.

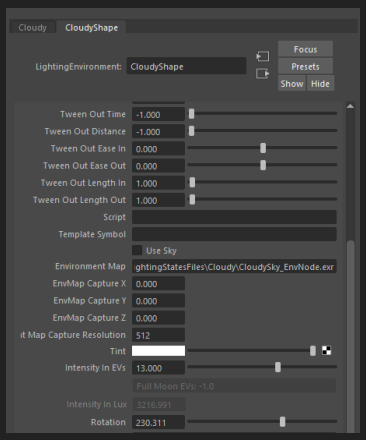
Art Workflow Concerns

- Lighting
- Reflection and refraction

Santa Monica Studio

Let's start by looking at lighting.

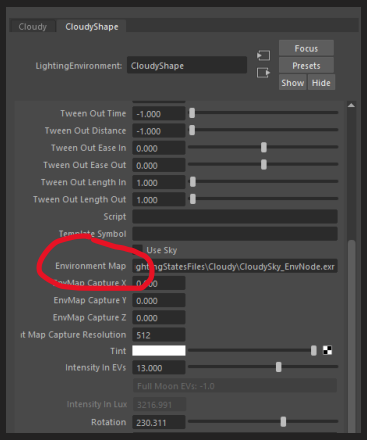
Create a lighting environment node:



Santa Monica Studio

Let me walk you through how we originally set up lighting for a simple outdoor scene…

Capture and plug in environment map

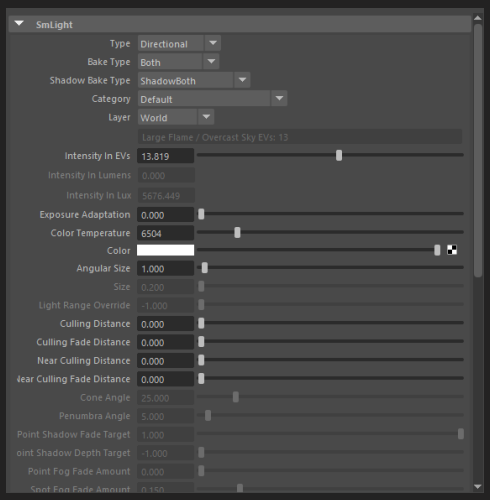


Environment map is used for sky lighting

Santa Monica Studio

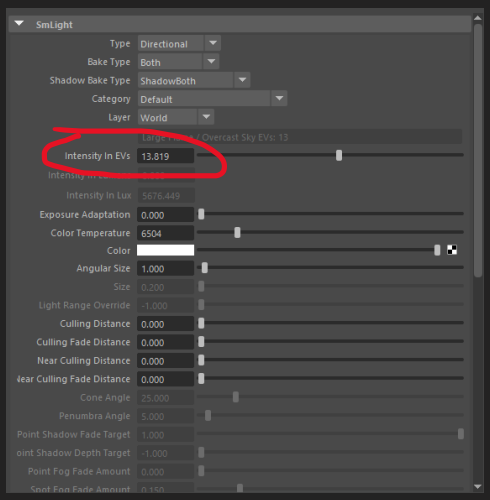
Hope that this never goes out of sync with the sky model!

Create directional light node



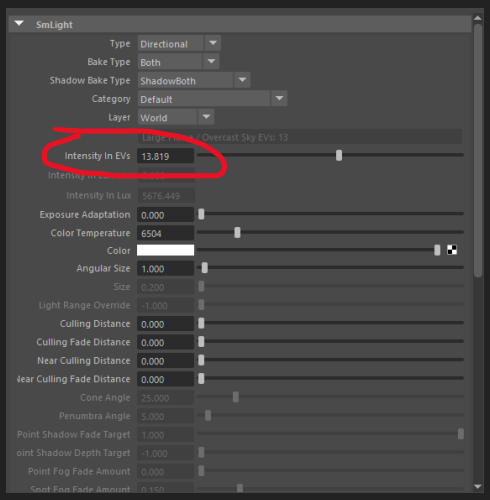
Santa Monica Studio

Set directional light intensity



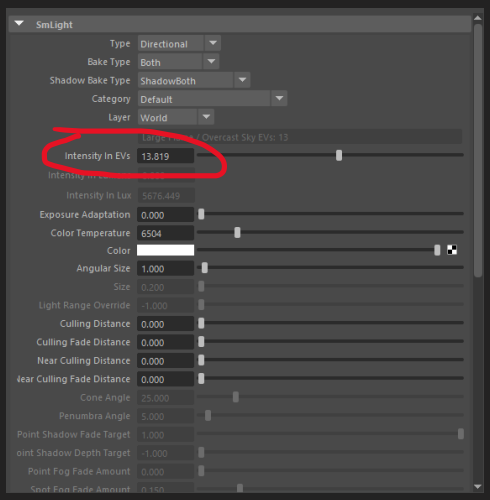
Santa Monica Studio

But what intensity?



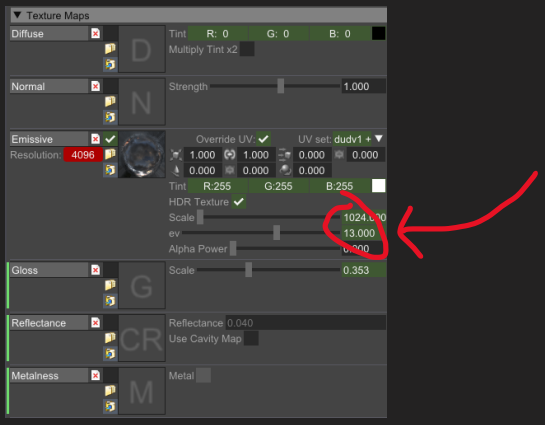
Santa Monica Studio

It probably should be relative to the sky...



Santa Monica Studio

Open the sky material



Find the sky material's emissive intensity

Santa Monica Studio

This is in our material editor, Mesa, outside of Maya.

But then what intensity should the sky be?

Santa Monica Studio

God of War 2018 used "relative" lighting units:

- Assume the base lighting of the scene is around 0 EVs
- All other lights are placed relative to that

Santa Monica Studio

Inspired by CG:

- Import pre-lit scene
- All lighting is relative to what is already there

Santa Monica Studio

Advantages:

- No problems with FP16 limits
- Fewer indoor/outdoor/day/night contrast problems

Santa Monica Studio

Disadvantages:

- Not a games industry standard
- New hires did not understand the workflow

Santa Monica Studio

This was difficult for new people on the team to understand – in fact, no one did, which ended up being very hard to standardize across lighters and our technical art team.

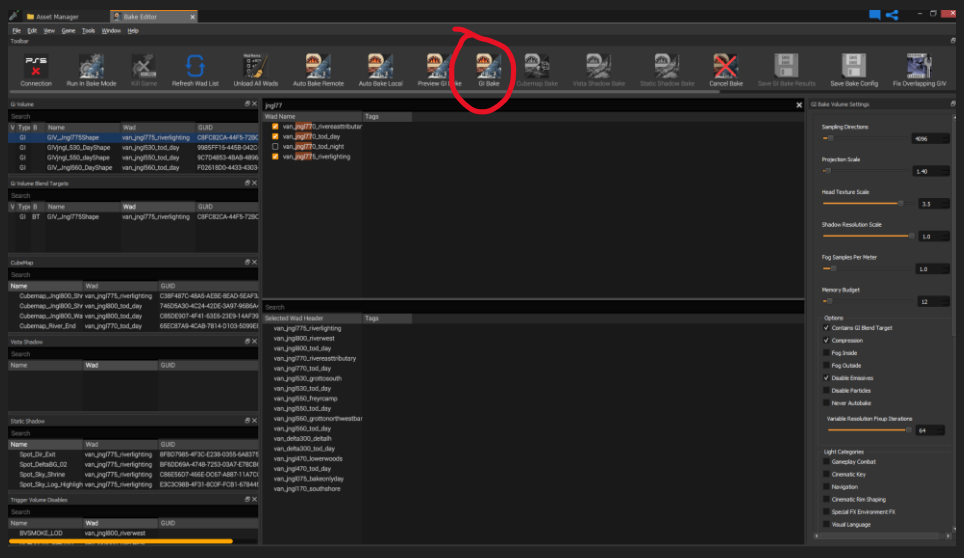
Disadvantages:

- In games, there is no base lighting to start from
- Therefore, no consistency across the game

Santa Monica Studio

This also encouraged a bad workflow. If your lights are relative, they need to be relative to *something*. The easiest thing is to say that's your "ambient" lighting, which is traditionally how you would work in days gone past. You first add your ambient lighting, then you add your direct lights on top. However, in a world with global illumination, that creates a dangerous feedback loop. Because your lights cause the global illumination, you have to place the lights first, not the "ambient".

Manually bake global illumination

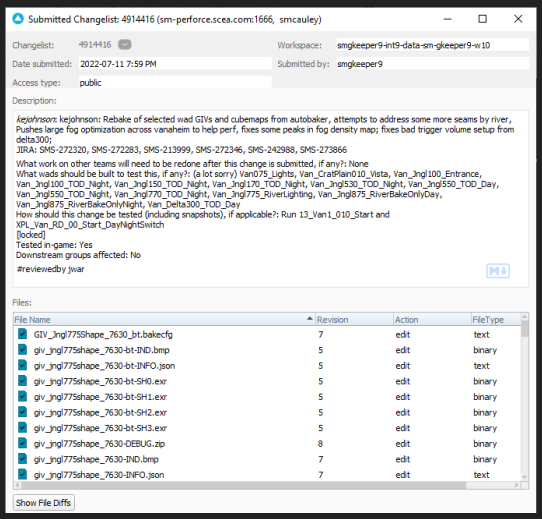


For more info on our GI system, see [Hobson2019]

Wait...

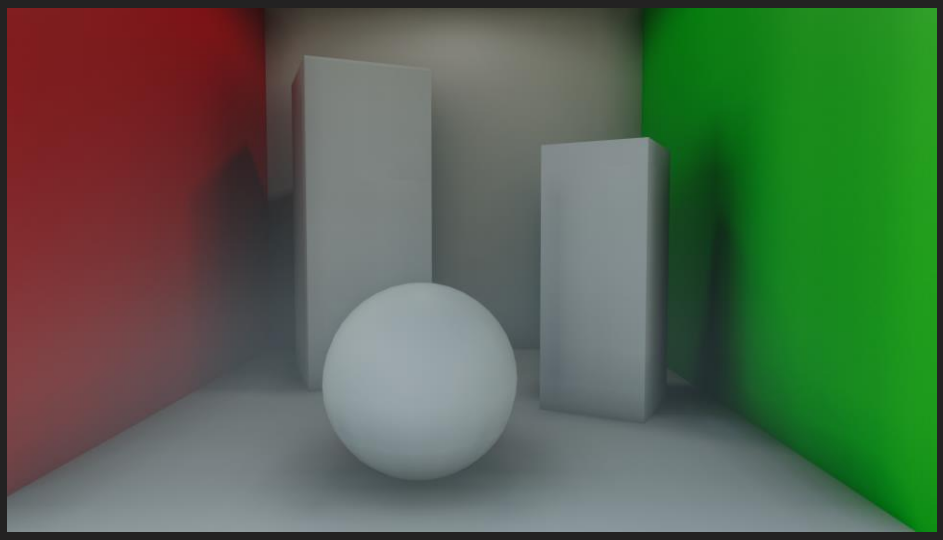
Santa Monica Studio

Submit lighting data into Perforce



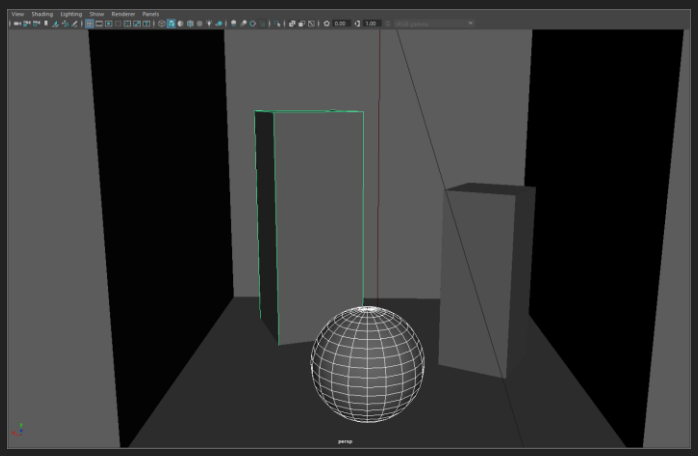
Santa Monica Studio

You have global illumination!



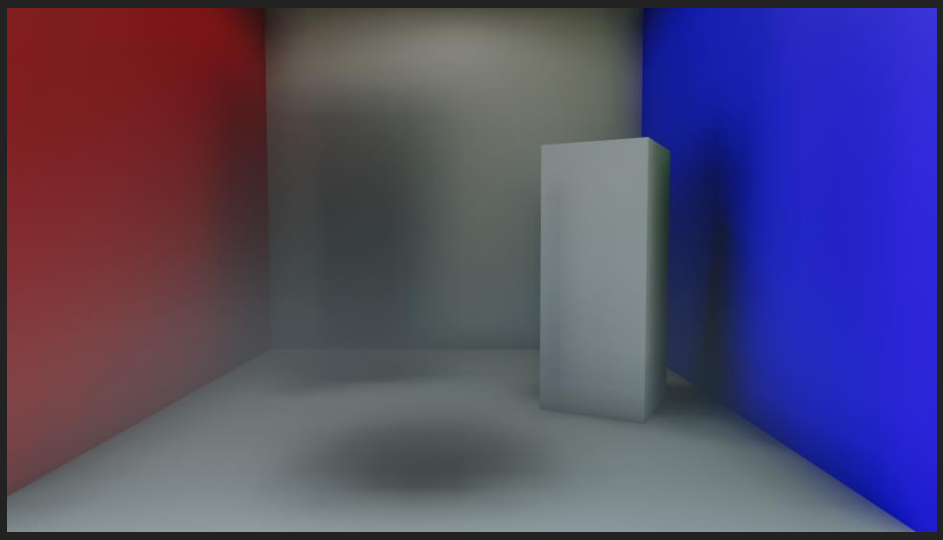
Santa Monica Studio

The next day, an environment artist moves some objects around...



Santa Monica Studio

...and your GI is invalidated.



Santa Monica Studio

Let's summarize the problems:

1. There were many steps to even a simple outdoor lighting setup
2. A "correct" setup was hard to achieve
3. Global illumination baking was manual and frequently out-of-date
4. No one understood the lighting units used

Santa Monica Studio

Now, I used to work on open world games, like Far Cry 5, so this was very new to me. I was used to a physical sun and sky system, and a light baking process that executed over night. It meant that the lighting was always up to date and you could change time of day and get different looks in an instant.

We came up with four solutions:

1. Generate sky lighting from the actual sky used in game
2. Created a simple way to generate outdoor lighting setups
3. Changed all units to physical
4. Automated GI baking pipeline

Sky lighting

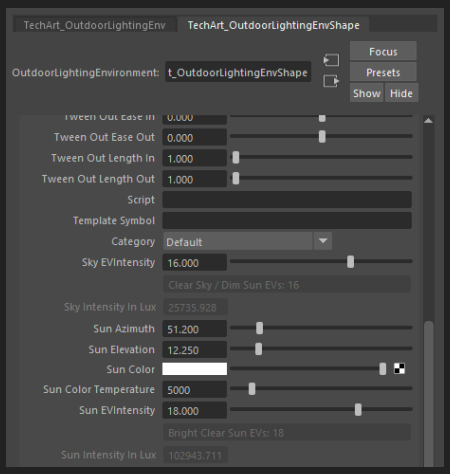
Render sky into cubemap

```
void mainCS_process(in uint2 dispatchThreadID : SV_DispatchThreadID, uniform CubemapToSHData cubemapToSHData)
{
    SHCoeffsRGB shCoeffsRGB = (SHCoeffsRGB)0;

    for (int i = 0; i < kCubemapToSHNumSamplesPerThread; ++i)
    {
        int sampleIndex = dispatchThreadID.x * kCubemapToSHNumSamplesPerThread + i;
        float2 lowDiscrepancyPoint2D = HammersleySequence2D(sampleIndex, kCubemapToSHNumSamplesPerThread * kCubemapToSHNumSamplingThreads);
        float3 direction = UniformHemisphere(lowDiscrepancyPoint2D).xzy; // flip z and y as UniformHemisphere assumes z is up, but for our cubemaps y is up

        float3 envMapSample = cubemapToSHData.cubeTexture.SampleLevel(cubemapToSHData.linearClampSampler, direction, 0).rgb * cubemapToSHData.globalSkyLightingScale;

        SHCoeffs shDirection = SHEval2Direction(direction);

        for (int j = 0; j < 9; ++j)
        {
            shCoeffsRGB.r[j] += shDirection.c[j] * envMapSample.r * (4.0f * PI / float(kCubemapToSHNumSamplesPerThread));
            shCoeffsRGB.g[j] += shDirection.c[j] * envMapSample.g * (4.0f * PI / float(kCubemapToSHNumSamplesPerThread));
            shCoeffsRGB.b[j] += shDirection.c[j] * envMapSample.b * (4.0f * PI / float(kCubemapToSHNumSamplesPerThread));
        }
    }

    // we do a cross lane add so we can drastically reduce the size of our output buffer
    SHCoeffsRGB shCoeffsRGBAllLanes = ( SHCoeffsRGB )0;
    const float activeCount = CrossLaneAdd( 1.0f );
    for( int k = 0; k < 9; ++k )
    {
        shCoeffsRGBAllLanes.r[k] = CrossLaneAdd( shCoeffsRGB.r[k] ) / activeCount;
        shCoeffsRGBAllLanes.g[k] = CrossLaneAdd( shCoeffsRGB.g[k] ) / activeCount;
        shCoeffsRGBAllLanes.b[k] = CrossLaneAdd( shCoeffsRGB.b[k] ) / activeCount;
    }

    if( ( dispatchThreadID.x % kCubemapToSHThreadsPerOutSample ) == 0 )
    {
        cubemapToSHData.shCoeffsRGBBuffer[dispatchThreadID.x / kCubemapToSHThreadsPerOutSample] = shCoeffsRGBAllLanes;
    }
}
```

Convert to SH for use in lighting

Santa Monica Studio

Outdoor lighting environment node

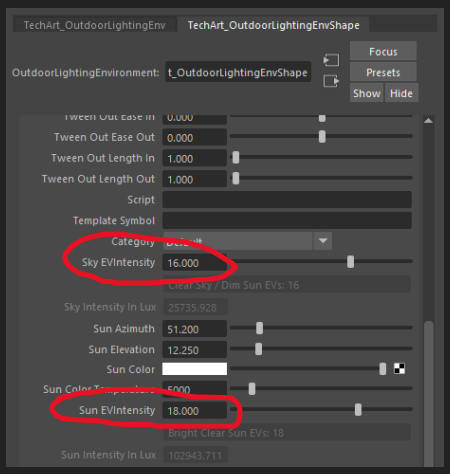


Santa Monica Studio

Next was the “outdoor lighting environment node”, which is a simple way to set sun and sky brightness in the same place.

Outdoor lighting environment node

Set sun and sky intensities in one place



Santa Monica Studio

This sky EV intensity also modifies the brightness of the sky material.

Converting to absolute lighting units

- One night in February 2021…
  - We ran scripts over the entire game converting lights into physical values
    - Had to convert exposure data at the same time
  - Assumed the game was generally day-time
  - Used sunny-16 rule to transform existing data into more physical day-time lighting

Santa Monica Studio

Thanks to Josh Hobson, Nathan Kennedy and Amy Chen for staying up late with me to do the conversion! Bourbon (and rye in my case) helped get Josh and I through the night!

Converting to absolute lighting units

- We also modified the exposure bias and tonemapping

- Our tonemapping gave the scene an incorrect midpoint:
  - Scene-referred 0.18 should map to display-referred 0.10 (Hines2010)
  - Our tonemapping kept the mid-point at 0.18
    - This is 80% higher, or ~0.85 stops in EV space
  - Artists compensated by biasing the default exposure value

- Our conversion adjusted the exposure bias by 0.5
  - Changed the tonemapping function in code at the same time

Santa Monica Studio

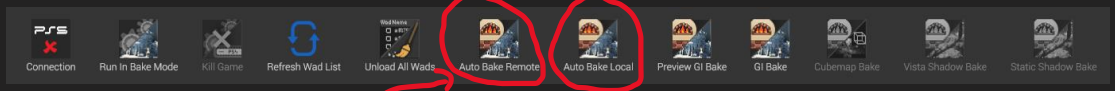
To show you the difference, I've taken some captures from a test level we have. This is the original tonemapping with no exposure bias. It's not too bad, but it is just a tiny bit too bright.

This is with an exposure bias of 0.5, something our lighting artists typically did to get the lighting they were after. This looks a lot better I think.

Now if we switch to the new tonemapping, now with handling the mid point correctly, you'll see that we don't need the exposure bias any more. It's very slightly darker than the previous scene but actually very, very close to what the artists were trying to get.

So handling your lighting data correctly is important! This is just another small thing that helped our lighting team out by ensuring everything was as good as possible out of the box.

Automated parts of the GI baking process



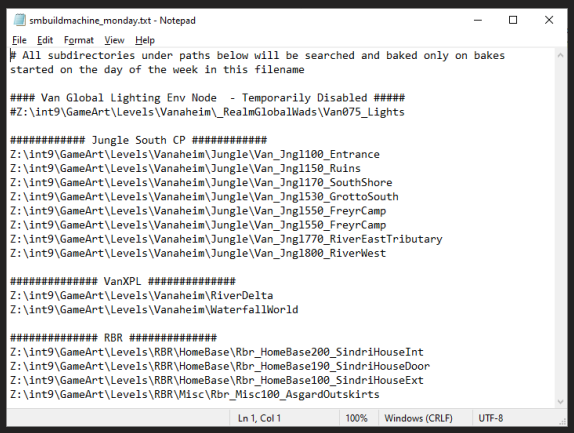
Bake a list of GI volumes locally

Bake GI volumes on the build farm and submit

Santa Monica Studio

Or let the build farm generate GI data automatically:

- A list to build nightly
- A list to build for each day of the week



Santa Monica Studio

We had both a list of GI data to generate every night, plus a separate list for each day of the week, just to manage the build times. It takes a long time to generate GI data.

Conclusion

1. It was quicker to set up outdoor lighting
2. We could provide guidelines to the lighting team
3. Lighting artists saved time baking GI

See [McAuley2018] for a more detailed discussion of outdoor and indoor lighting challenges.

Art Workflow Concerns

- Lighting
- Reflection and refraction

Santa Monica Studio

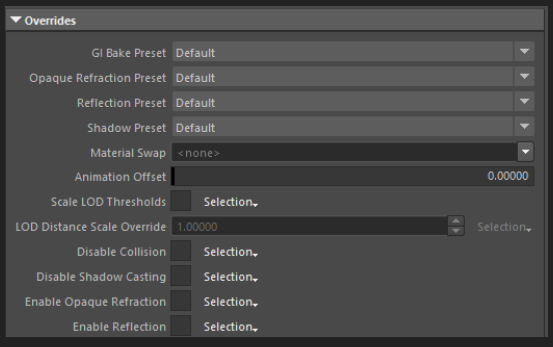
Now that we've solved some of the lighting team's workflow issues, let's move onto environment art and talk about reflection and refraction.

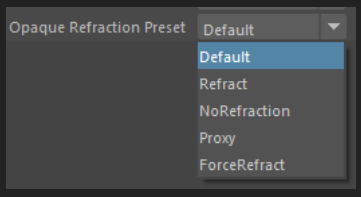
Reflection and Refraction

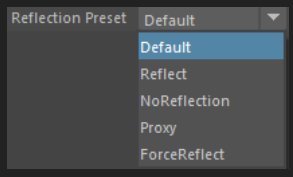
- Reflection and refraction performed by rendering geometry into separate render targets
- Artists would build custom geometry for these passes

Reflection and Refraction

- Geometry tagged in Maya to be in reflection and/or refraction passes:







Santa Monica Studio

Reflection and Refraction: Problems

- Visual quality of reflection and refraction
  - Did not match regular lighting pipeline
  - Also had to maintain a forward lighting pipeline

- No reflection or refraction by default
  - Meshes removed by default for performance reasons
  - Missing reflections and refractions for much of development

Reflection and Refraction: Problems

- Expensive for artists to maintain
  - Geometry had to be manually flagged
  - Proxy meshes had to be authored for performance reasons
- This could be 60 days of work for just one area of the game

Beta estimates in June 2021

Reflection Pass **Beta** **20**

- Clean up and polish worldbuilding
  - Generate shrinkwraps, implement and optimize them - 12
- Material polish / optimization – 3
- Beta performance optimization pass (mainly trigger volumes) – 5

Refraction Pass **Alpha** **10**

- Clean up and polish worldbuilding
  - General improvements (requests from Art Director / Leads) – 5
- Material polish / optimization – 1
- Beta performance optimization pass (mainly trigger volumes) – 3
- Design / Playtest bug support / Visual language - 1

Mid_Cal200_Shore – at Alpha

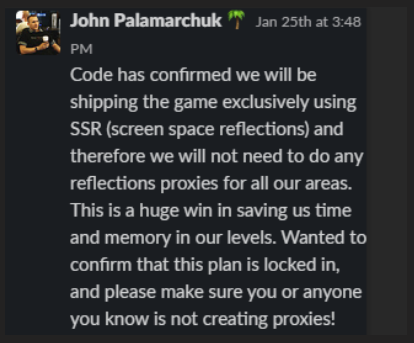
- **Reflection Pass** **Alpha** **20** **Ship** **10**
- **Refraction Pass** **Alpha** **10** **Ship** **10**

Santa Monica Studio

Here are some images one of our producers, Brielle Porter, gave me, about notes she had on reflection and refraction cost.

Reflection and Refraction: Solutions

- Replaced with screen space techniques:
  - Screen space reflections
  - Framebuffer refraction
    - Sample distorted framebuffer for refraction value
- Many, many days of art time saved
- Instant visual improvement



Santa Monica Studio

Screen space techniques are much maligned, for many reasons, but the immediate quality lift and art time saved was phenomenal.

You can see a great quote from our Director of External Development, John Palarmarchuk, being very happy about how much time we're saving the art team.

This is also a great example of building up trust. The art team know that we are listening to their concerns, and we are delivering on features that make their lives easier.

Reflection and Refraction: Water

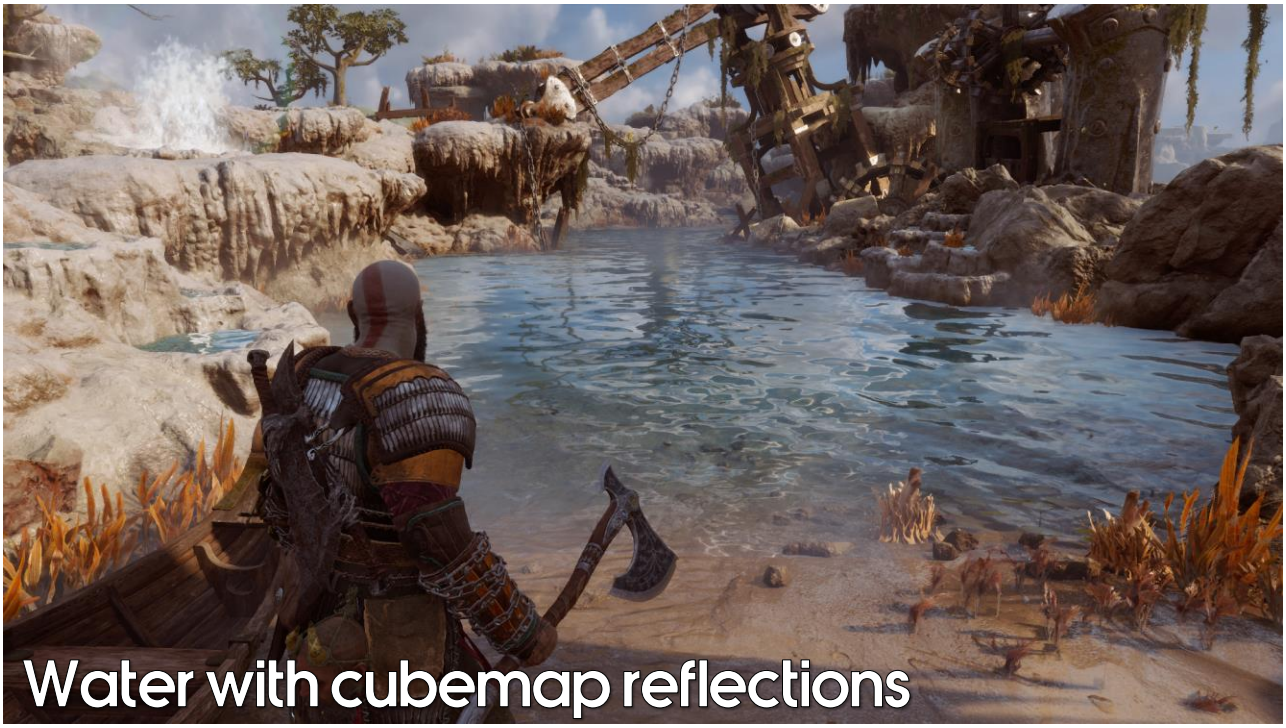
- How do screenspace reflections work on water?
  - Reflection and refraction are primarily used on water
  - Framebuffer refraction pass happens after deferred lighting and screen space reflections

- We perform SSR directly in the water shader:
  - Render the SSR portion of water shading into a half-resolution buffer
  - Sample during main rendering of the water mesh

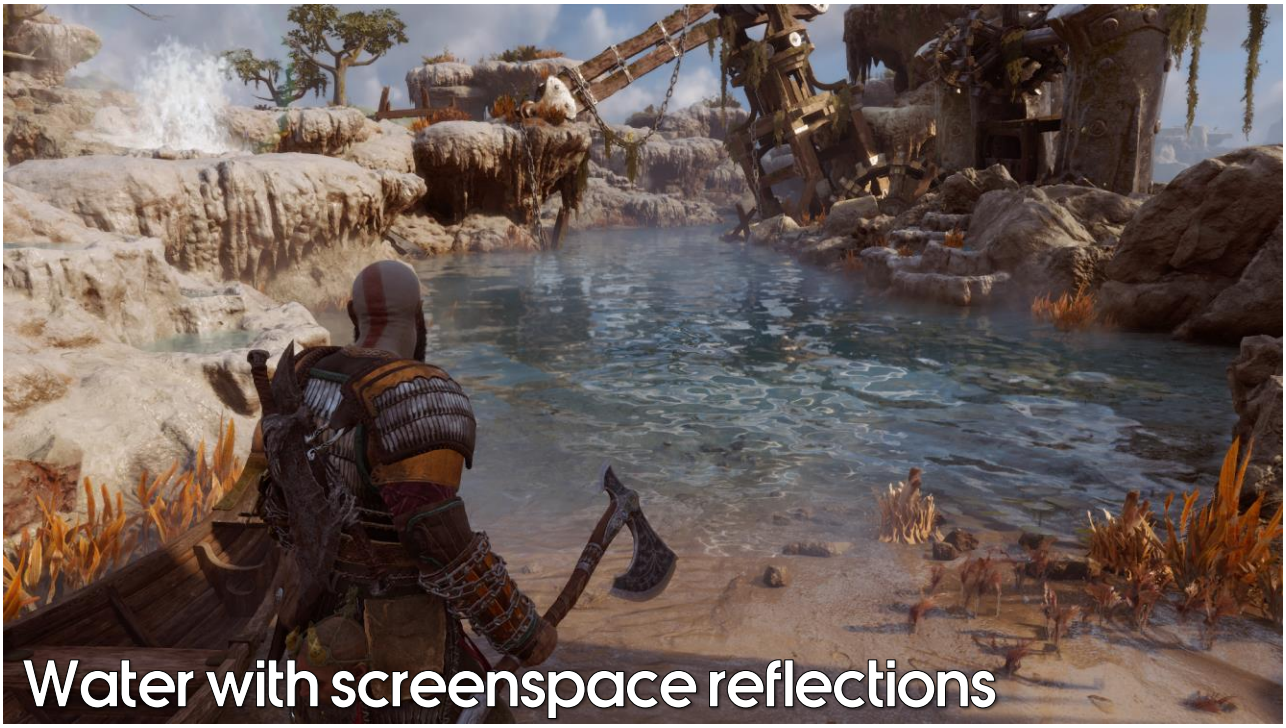
- For other solutions, see (Grujic2018)

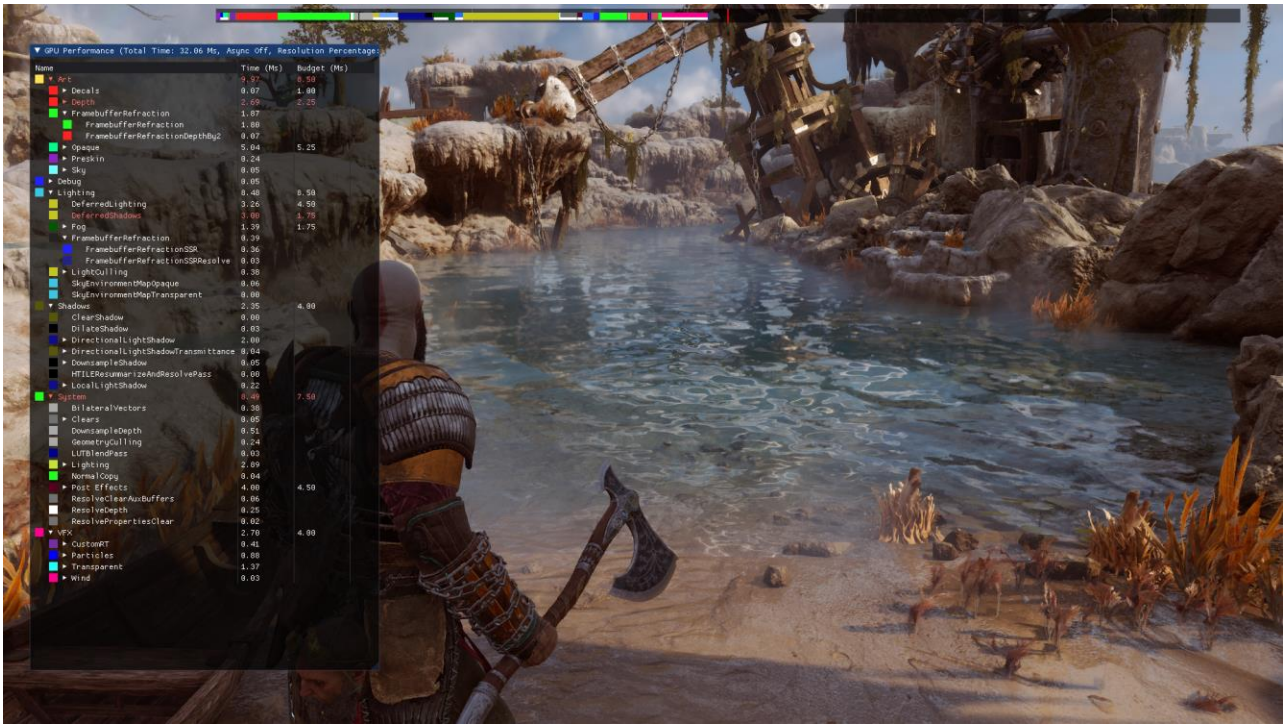
Santa Monica Studio


No water

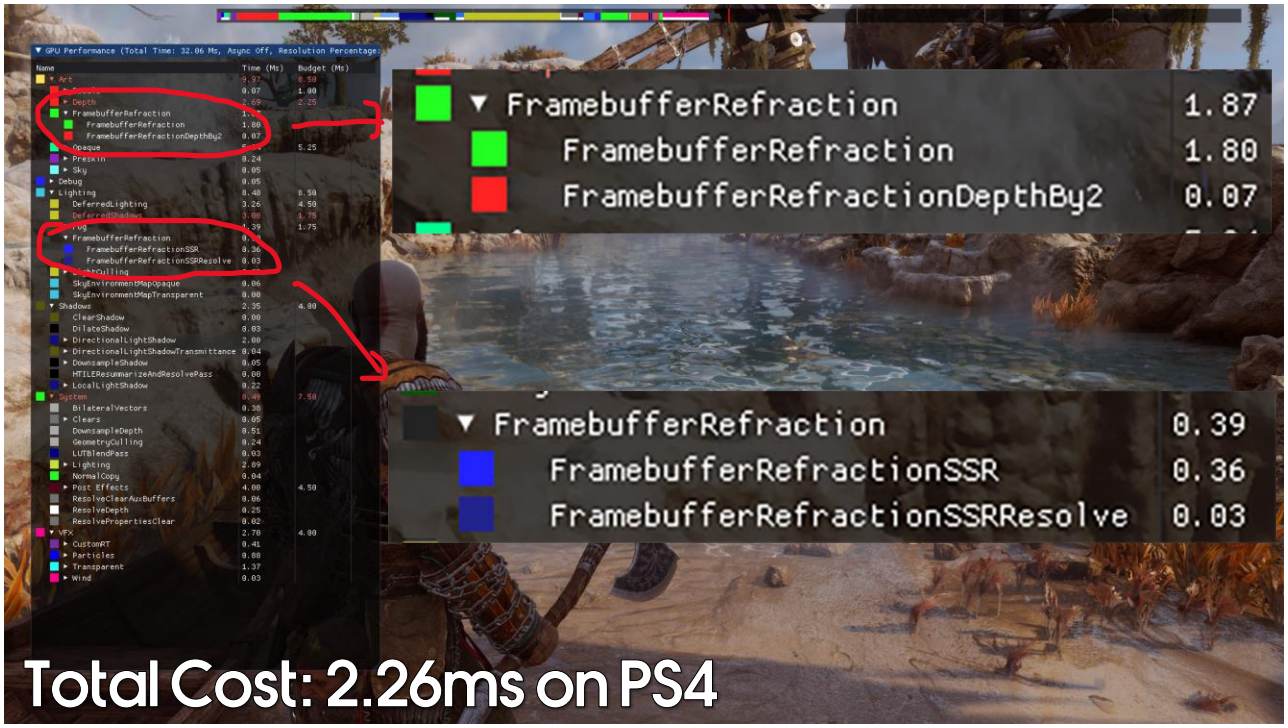

Water with cubemap reflections

Screenspace reflections


Water with screenspace reflections



Let’s take a quick look at the performance of this pass on PS4, so I’m bringing up our GPU profiler. You probably can’t see it too well right now, so let’s enlarge…



Our total cost for the water here is 2.26ms on PS4, which is a little expensive but it does contribute to a lot of the screen.

We split the costs of the passes into two categories, “Art” for the base water surface and “Lighting” for the SSR part. This is partially historical – the old mirror and opaque refraction passes were under the “Lighting” cost, so this allows us to maintain roughly the same burden of this feature across teams.

Reflection and Refraction: Performance

- Our water surface is inevitably expensive:
  - Forward shaded
- Asked artists to:
  - Reduce material complexity of the water surface
  - Cheapen objects under the water
- Ultimately did not cause too many performance problems:
  - Environment art did a great job optimizing
  - Only really expensive when full screen
    - And when full screen, nothing else expensive is being rendered

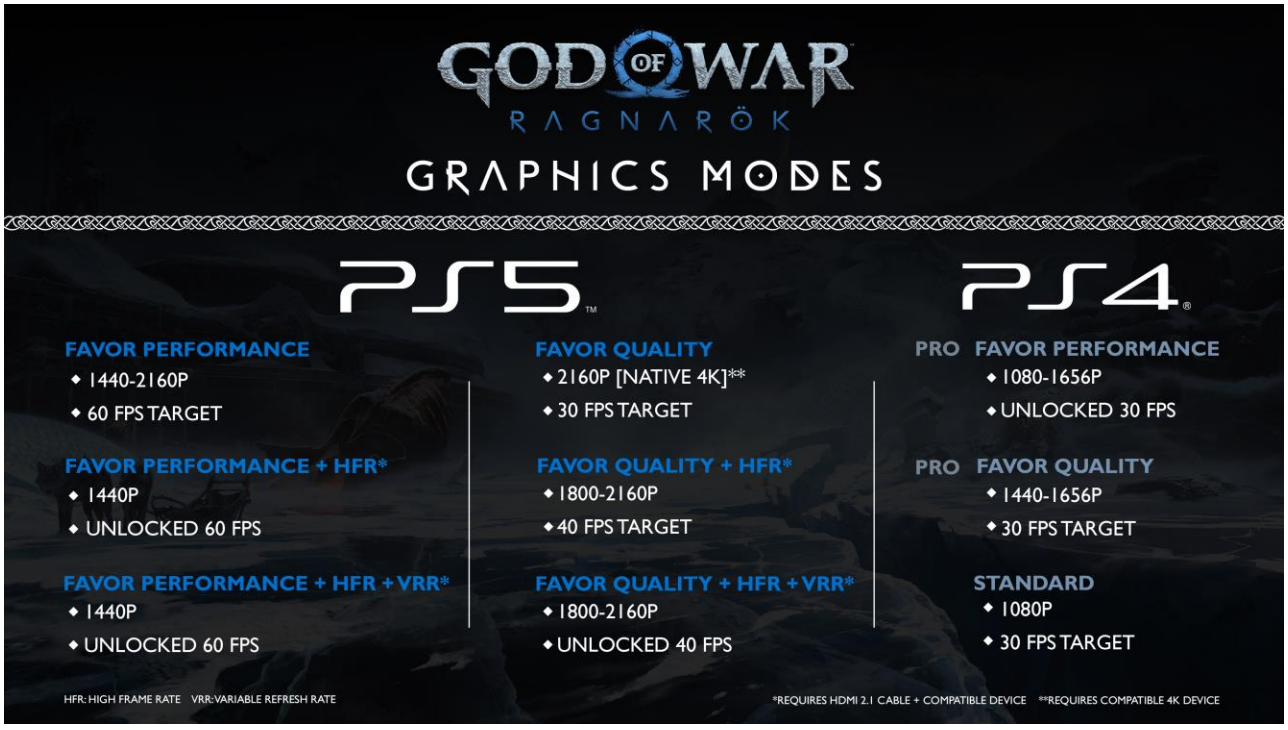
Santa Monica Studio

Obviously, the water surface is always going to be expensive as it's forward shaded, so it has to perform material and lighting evaluation.

Third Pillar:

PlayStation 5 Enhancements

Santa Monica Studio


GOD OF WAR
RAGNARÖK
GRAPHICS MODES
PS5
FAVOR PERFORMANCE
◆ 1440-2160P
◆ 60 FPS TARGET
FAVOR PERFORMANCE + HFR*
◆ 1440P
◆ UNLOCKED 60 FPS
FAVOR PERFORMANCE + HFR + VRR*
◆ 1440P
◆ UNLOCKED 60 FPS
FAVOR QUALITY
◆ 2160P [NATIVE 4K]**
◆ 30 FPS TARGET
FAVOR QUALITY + HFR*
◆ 1800-2160P
◆ 40 FPS TARGET
FAVOR QUALITY + HFR + VRR*
◆ 1800-2160P
◆ UNLOCKED 40 FPS
PS4
PRO FAVOR PERFORMANCE
◆ 1080-1656P
◆ UNLOCKED 30 FPS
PRO FAVOR QUALITY
◆ 1440-1656P
◆ 30 FPS TARGET
STANDARD
◆ 1080P
◆ 30 FPS TARGET
HFR: HIGH FRAME RATE VRR: VARIABLE REFRESH RATE
*REQUIRES HDMI 2.1 CABLE + COMPATIBLE DEVICE **REQUIRES COMPATIBLE 4K DEVICE

PS5-Specific Features

Performance Mode:

- Tessellation
- Screenspace directional occlusion (SSDO)
- Screenspace global illumination (SSGI)
- Simple contact hardening shadows
- Enhanced screen space reflections
- PS5-specific models
- PS5-specific lights

Quality Mode:

- Machine learned texture upsampling (Zhou23)
- Raytraced cubemaps
- Advanced contact hardening shadows
- Enhanced sky lighting
- Enhanced depth of field
- Enhanced tessellation
- Enhanced screenspace reflections
- Higher resolution shadows
- Higher quality LODs

Santa Monica Studio

PS5-Specific Features

Performance Mode:

- Tessellation
- Screenspace directional occlusion (SSDO)
- Screenspace global illumination (SSGI)
- Simple contact hardening shadows
- Enhanced screen space reflections
- PS5-specific models
- PS5-specific lights

Quality Mode:

- Machine learned texture upsampling (Zhou23)
- Raytraced cubemaps
- Advanced contact hardening shadows
- Enhanced sky lighting
- Enhanced depth of field
- Enhanced tessellation
- Enhanced screenspace reflections
- Higher resolution shadows
- Higher quality LODs

Santa Monica Studio

PS5-Specific Features

Performance Mode:

- Tessellation
- Screenspace directional occlusion (SSDO)
- Screenspace global illumination (SSGI)
- Simple contact hardening shadows
- Enhanced screen space reflections
- PS5-specific models
- PS5-specific lights

Quality Mode:

- Machine learned texture upsampling (Zhou23)
- Raytraced cubemaps
- Advanced contact hardening shadows
- Enhanced sky lighting
- Enhanced depth of field
- Enhanced tessellation
- Enhanced screenspace reflections
- Higher resolution shadows
- Higher quality LODs

Santa Monica Studio

PS5-Specific Features

Performance Mode:

- Tessellation
- Screenspace directional occlusion (SSDO)
- Screenspace global illumination (SSGI)
- Simple contact hardening shadows
- Enhanced screen space reflections
- PS5-specific models
- PS5-specific lights

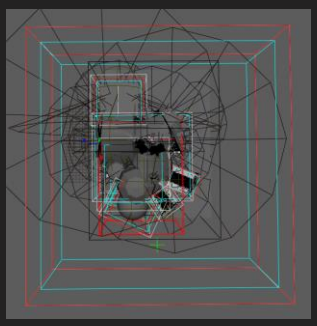
Quality Mode:

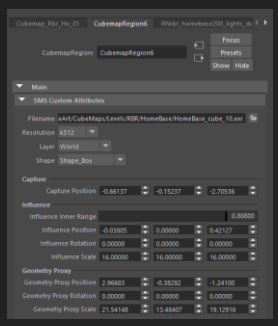
- Machine learned texture upsampling (Zhou23)
- Raytraced cubemaps
- Advanced contact hardening shadows
- Enhanced sky lighting
- Enhanced depth of field
- Enhanced tessellation
- Enhanced screenspace reflections
- Higher resolution shadows
- Higher quality LODs

Santa Monica Studio

Raytraced Cubemaps

- God of War: Ragnarök uses parallax-corrected cubemaps (Lagarde2012):
  - Find the intersection of the reflection ray with a box
  - Use this to adjust cubemap sampling position





Santa Monica Studio

The image on the left shows all the cubemap volumes in the RBR house area of the game (the red colour indicates the outer region, the blue colour the inner region). The image on the right shows our Maya set up for a cubemap region.

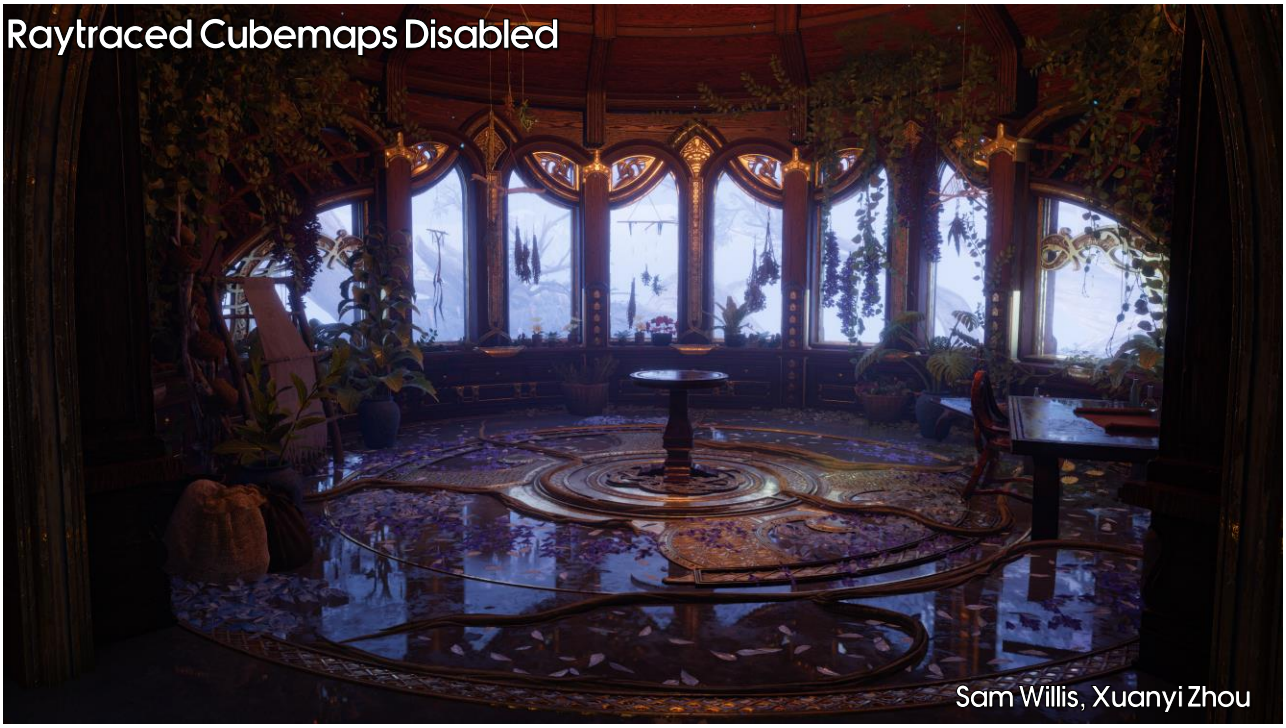
Raytraced Cubemaps

- Improve by finding the intersection of the reflection ray with the scene's geometry
  - Use hardware raytracing to do this
- Combine with screenspace raytracing for reflections
  - Use screenspace data whenever available

Raytraced Cubemaps

1. Screen space ray trace
   - Hit point
   - Color
2. If no hit point:
   - Ray trace the BVH
   - If the hit point corresponds to a screen space position:
     - Sample color from screen space buffer
   - Else:
     - Sample cubemap
3. Combine SSR and BVH results

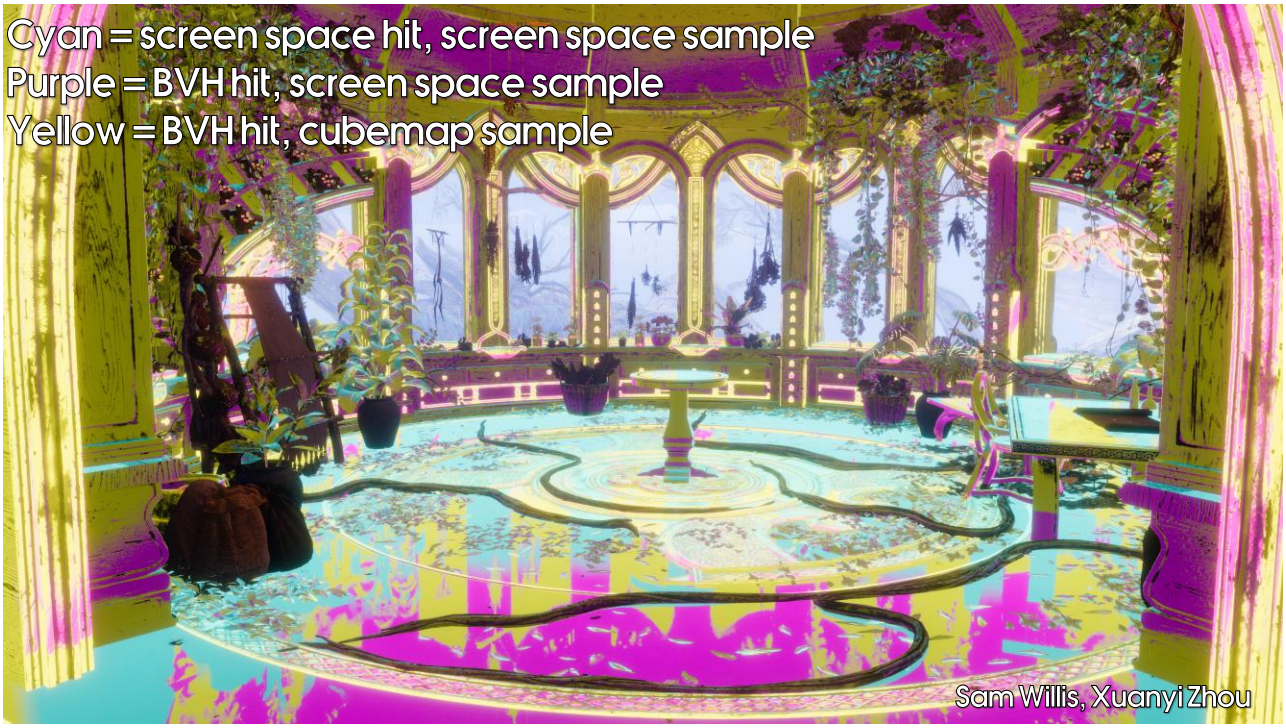
Santa Monica Studio


Raytraced Cubemaps Disabled
Sam Willis, Xuanyi Zhou


Raytraced Cubemaps Enabled
Sam Willis, Xuanyi Zhou

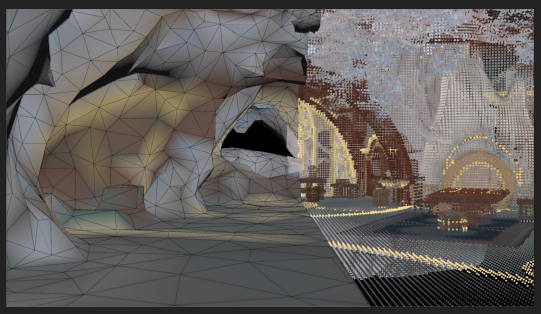
Hit position (in world space, relative to the camera position)

Hit position (in world space, relative to the camera position)


Cyan = screen space hit, screen space sample
Purple = BVH hit, screen space sample
Yellow = BVH hit, cubemap sample
Sam Willis, Xuanyi Zhou

Raytraced Cubemaps

- We do not raytrace actual game geometry
- Instead, create proxy geometry:
  - Take GI surfel data from the GI baking process
  - Turn into a mesh using Houdini

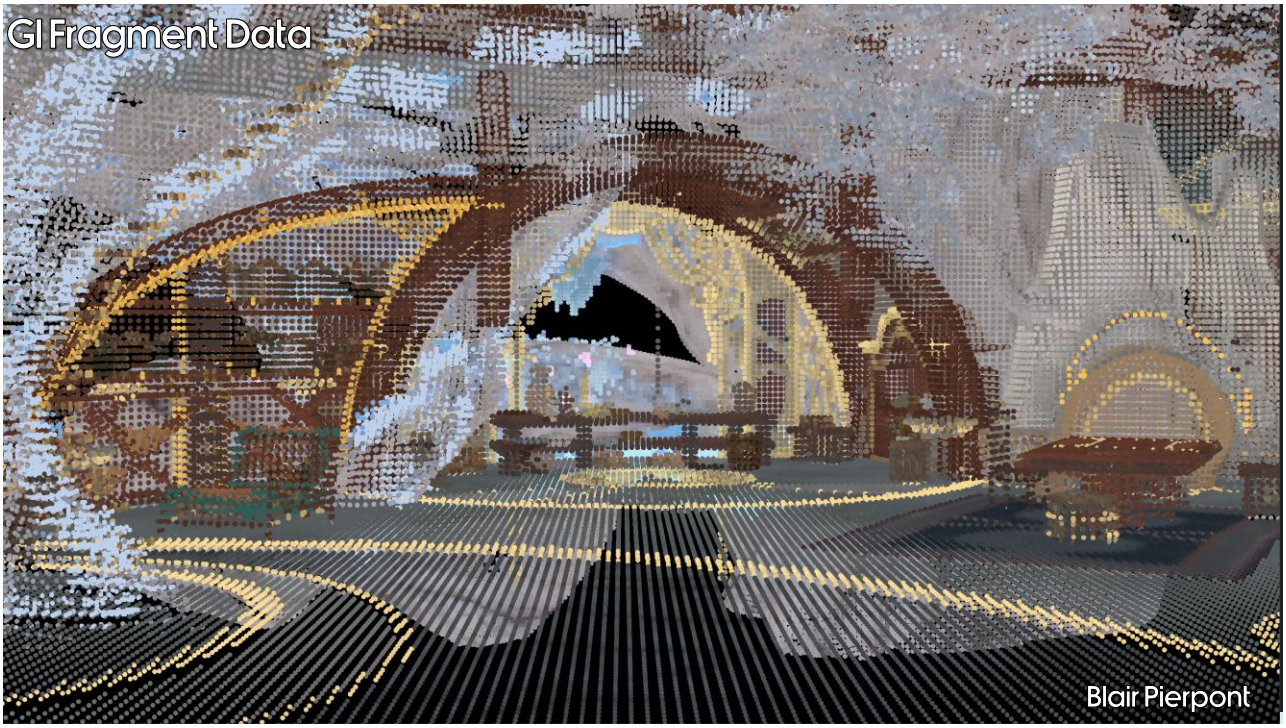


Santa Monica Studio

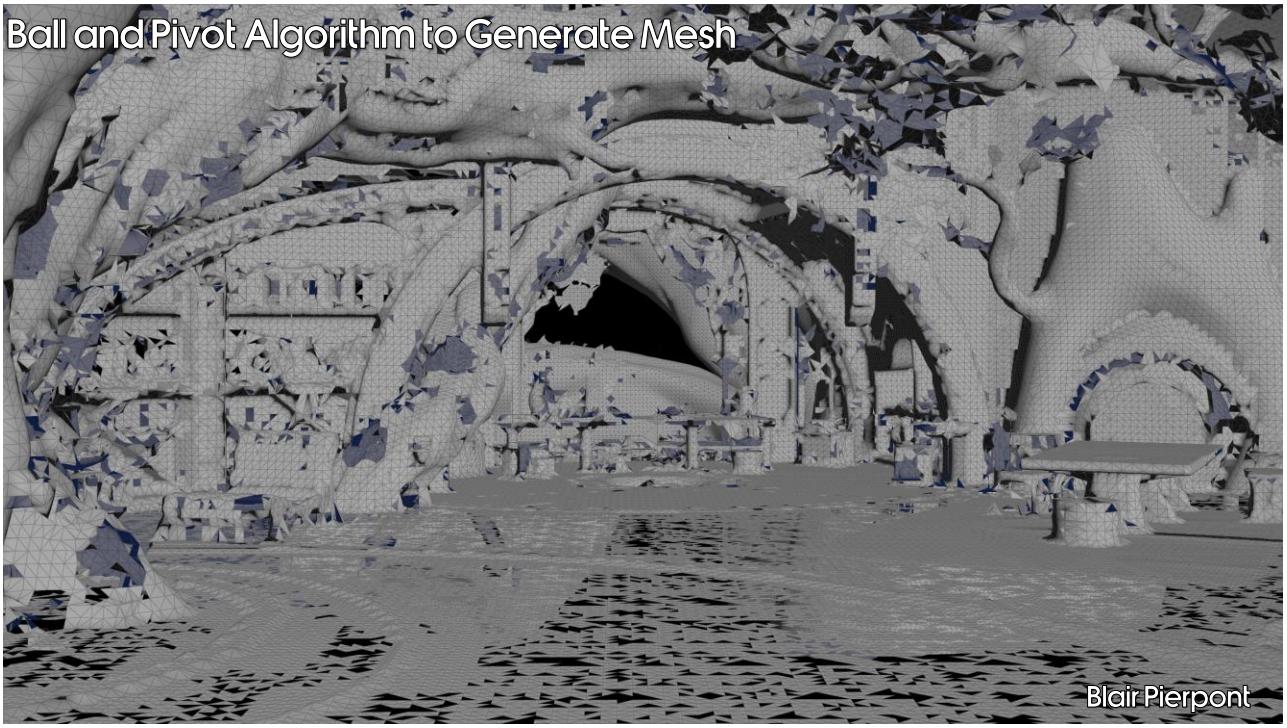
We decided not to use actual game geometry for the sake of simplicity and performance. This is our first time using raytracing so we wanted to contain the scope.

Instead, we created special geometry for areas we wanted to have raytraced cubemaps via Houdini.

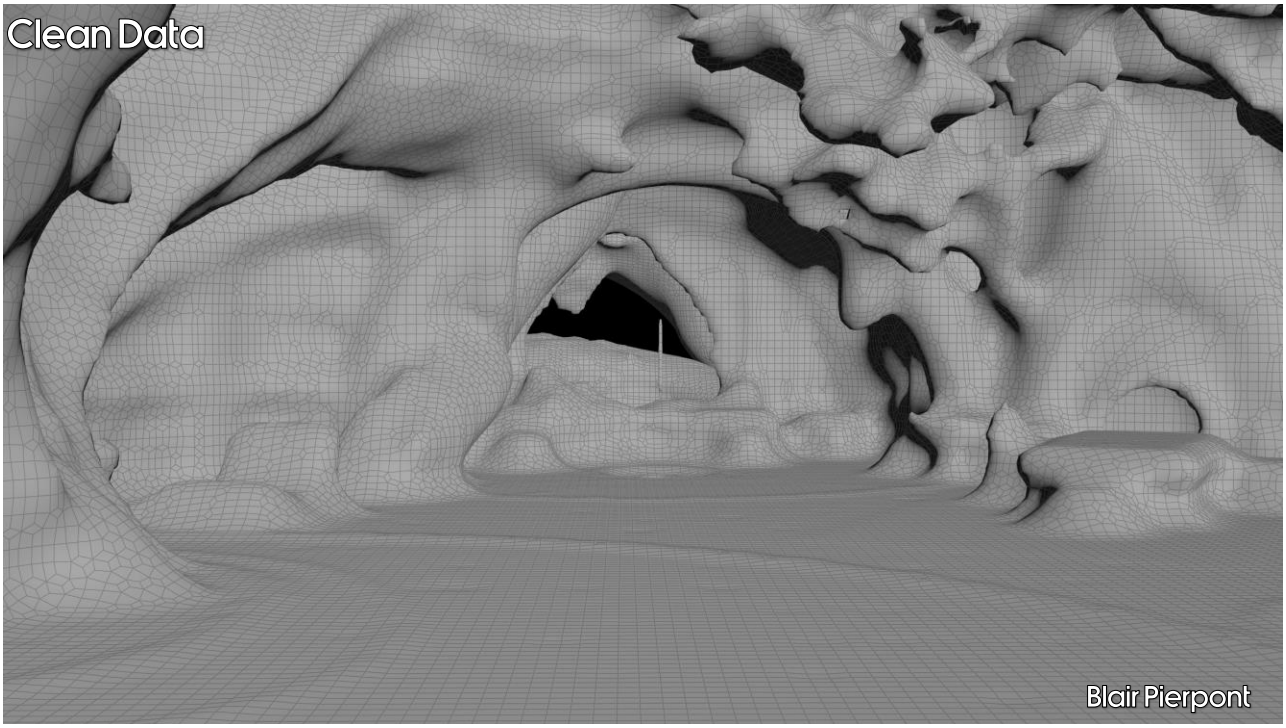
Credit to Blair Pierpont for his excellent work on this!



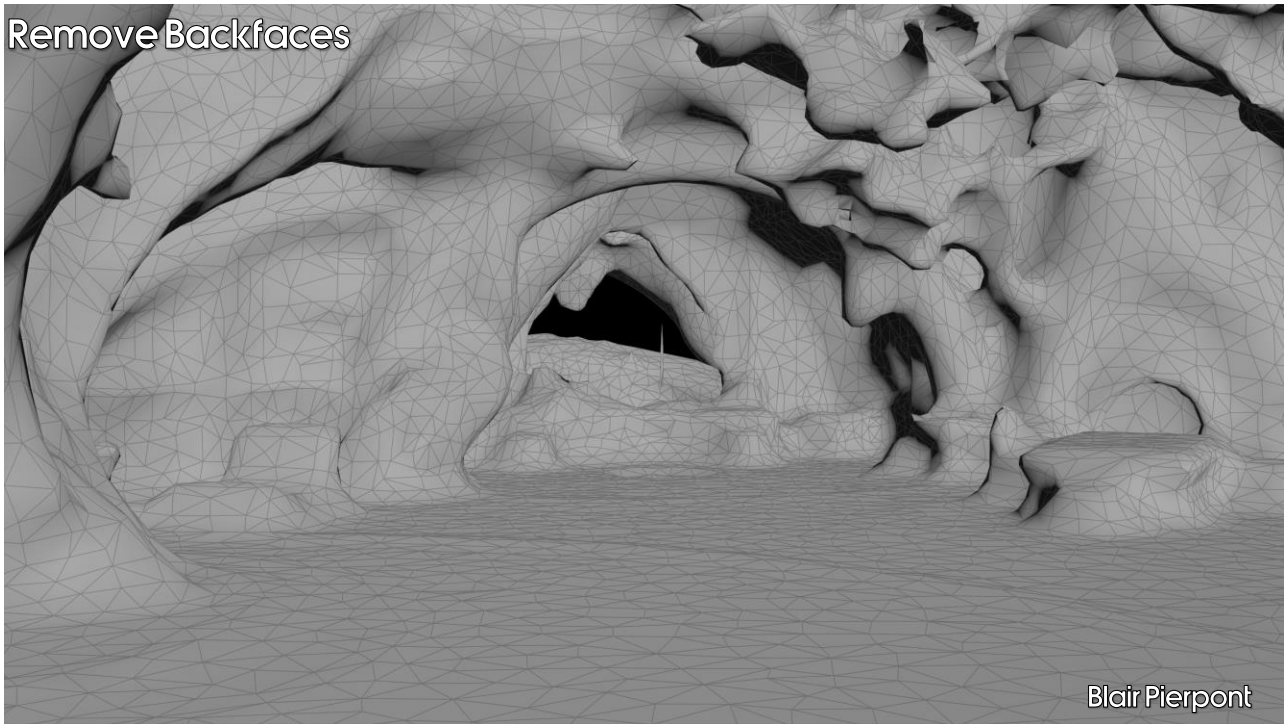
Wrote tool to read gi fragments (pulls in position, albedo, normal).



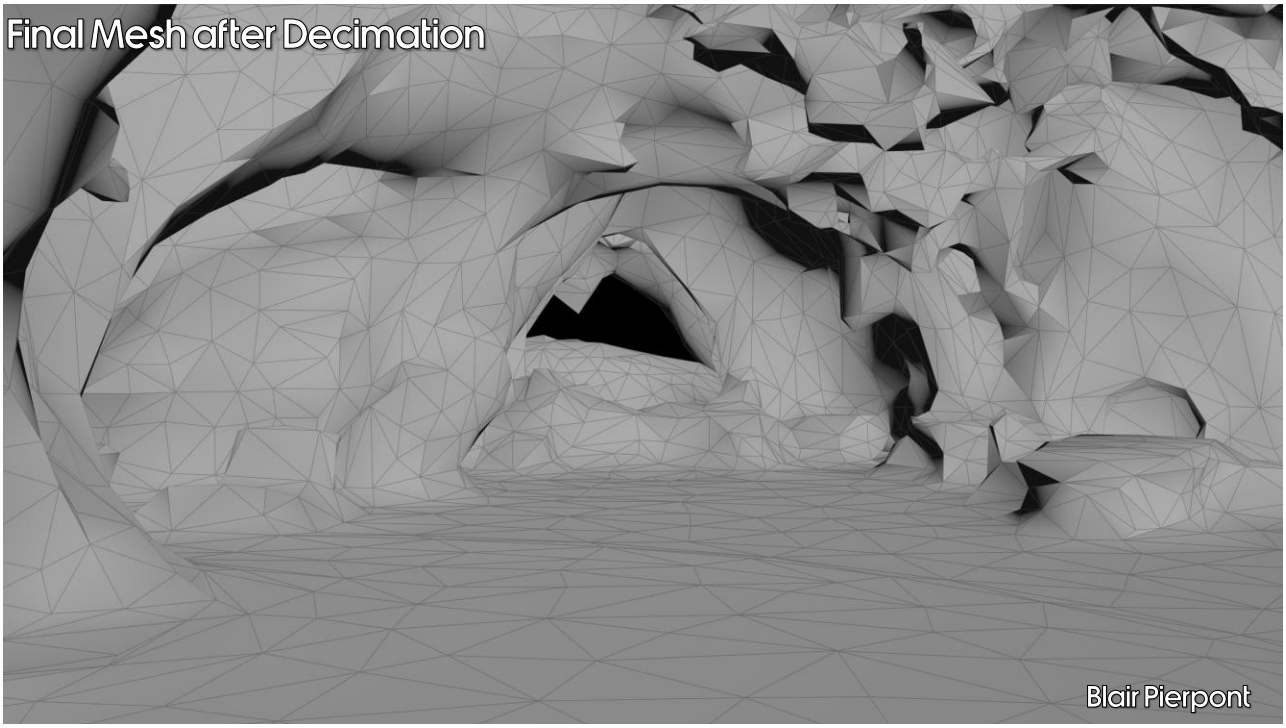
Use ball and pivot algo to generate mesh from point cloud (open3d python library has a convenient wrapper).
Easy to get false positives if the data is very coarse, thin geometry was problematic.



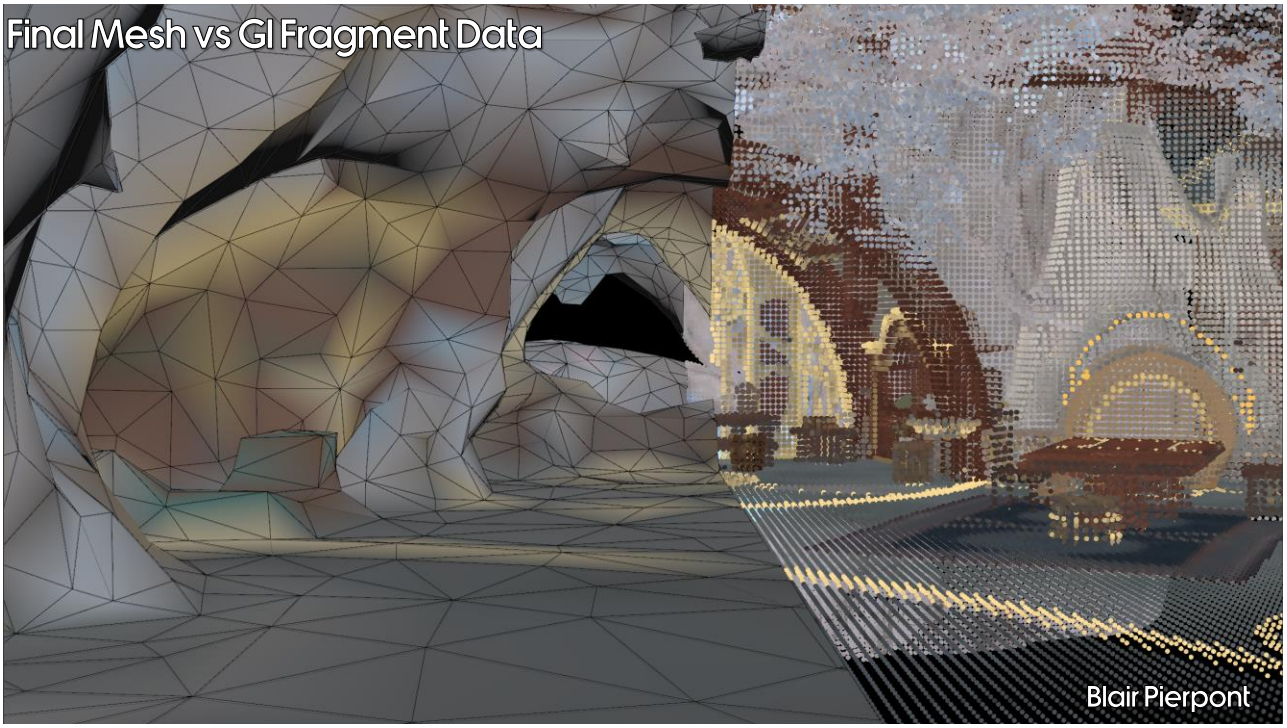
Clean the ball and pivot algorithm resultresult by generating VDB, then rasterize to geometry.



VDB raster result is double-sided, remove backfaces by line tracing from fragments to identify front faces.

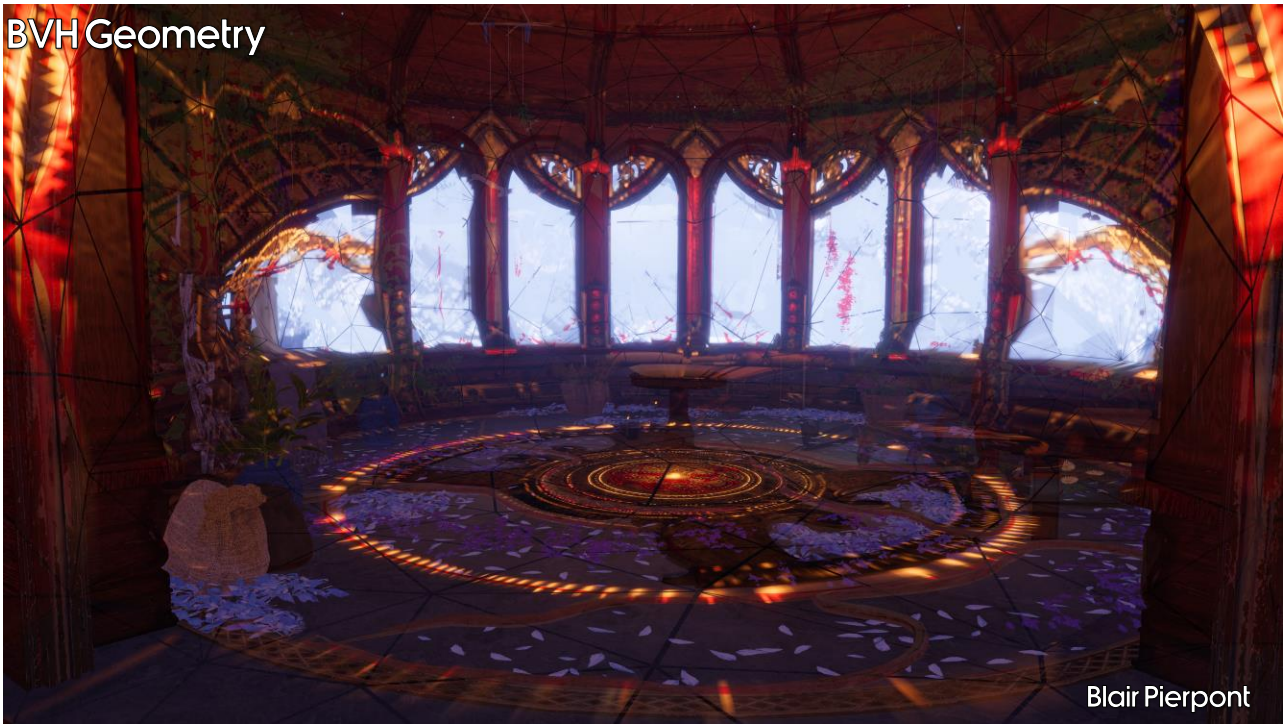


Decimate and retain geometry by curvature using polyreduce in Houdini, original target was 33k tris per wad.
This limit was increased eventually once raytracing perf had improved.
RBR is currently around 150k tris.



And here we have a final comparison of the input data on the right, and the final mesh on the left.

If we go back to the part of RBR we were looking at, here is the raytraced cubemap result…

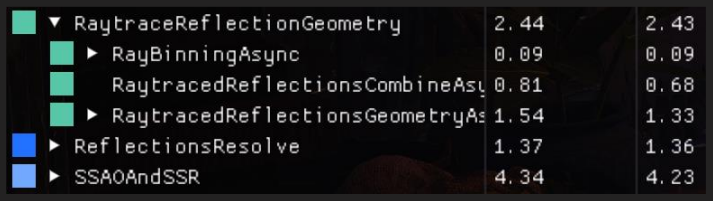


…and here is the BVH geometry projected onto the scene.

Let’s bring up our profiler now and discuss performance.

Raytraced Cubemaps

- SSR, SSDO and SSGI: 4.23ms
  - Combined to avoid sampling G-Buffer data twice
- Ray binning: 0.09ms
  - Rays binned in 64x32 blocks, but very little performance gain
- Raytrace BVH: 1.33ms
  - Half-res
- Combine: 0.68ms
  - Half-res
- Resolve: 1.36ms
  - Full-res



Performance numbers given at 4K on PS5.

Santa Monica Studio

The combine pass does the work of combining BVH and screen space ray tracing results and performing the screen space colour or cubemap lookup.

The resolve pass combines neighbouring and temporal samples into a final reflection result, using importance sampling. See [Stachowiak2015].

Contact Hardening Shadows

- Shadow resolution and shadow bias caused major annoyances on God of War 2018
- Large shadow biases caused lack of shadows on character faces
- This is an example of *light leaking*
- Contact hardening shadows are a solution to this

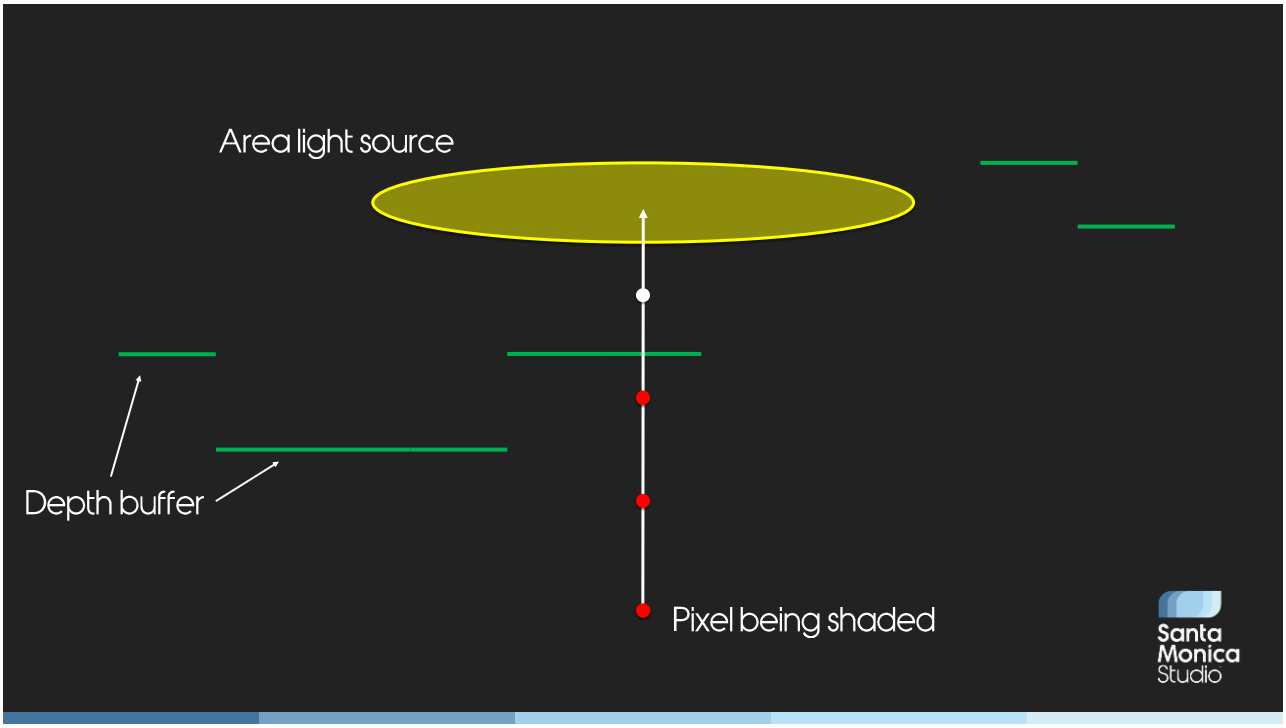
Santa Monica Studio

Look at the light leaking onto Kratos' nose. (Screenshot from God of War 2018, taken by Josh Hobson.)

Contact Hardening Shadows

- Solve contact hardening shadows through raytracing:
  - Trace rays towards the area light
- Idea: Ray trace in shadow map space instead
  - Trace through shadow map
  - Similar to screen space ray tracing techniques
    - Not hardware raytracing

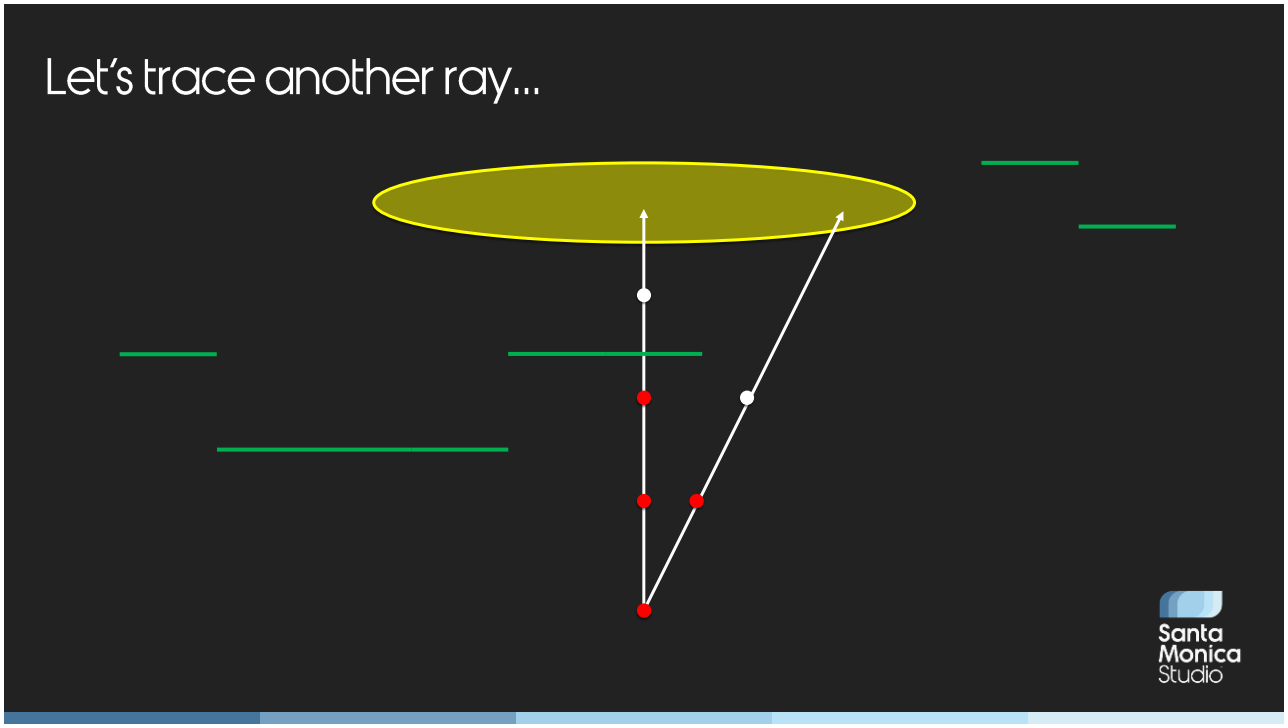
Our technical director Josh Hobson had the idea to ray trace in screen space.



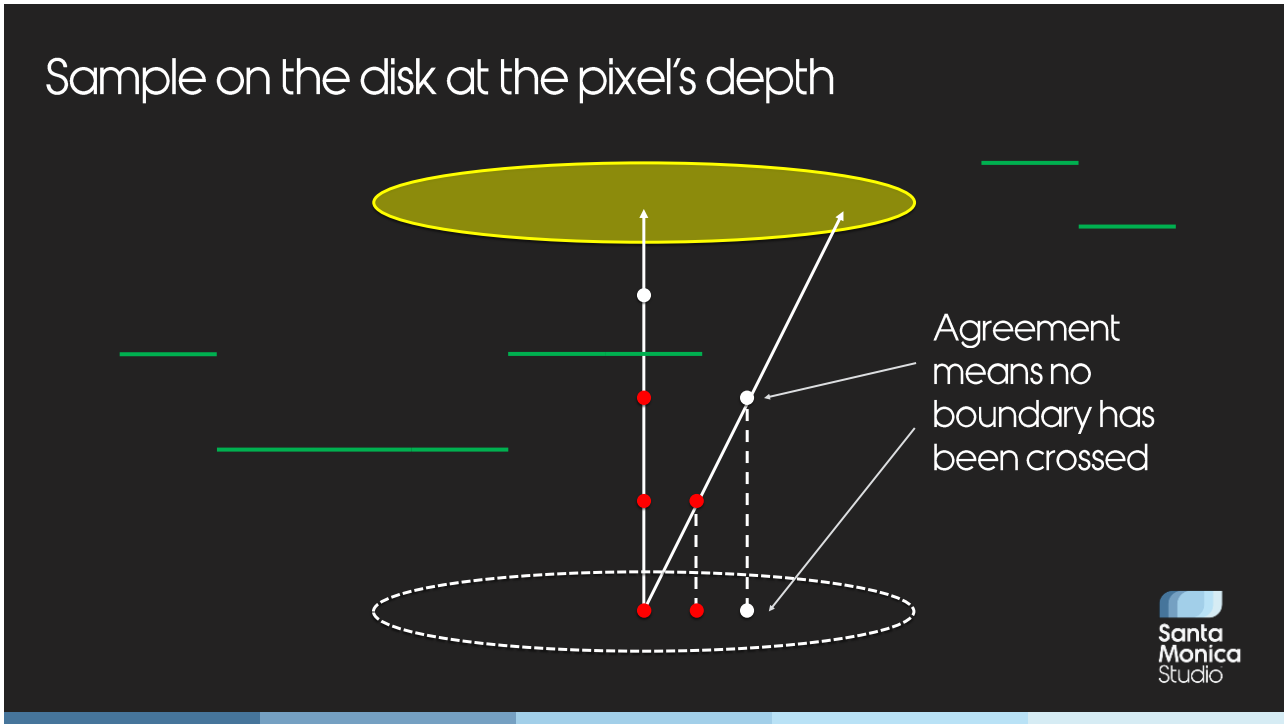
Let's see how this works in practice. Let's consider how we shade a pixel. We want to trace rays to this area light source, through this depth buffer which is represented as a set of green lines – each line segment is a depth buffer texel with a certain depth. It's important to remember that this depth buffer really doesn't represent a contiguous height field, we have to treat it as an isolated collection of fragments.

We're ray tracing to see if we intersect the depth buffer. In this case, let's imagine we trace the simplest ray, one aligned with the direction of the light. In this case you can see that the first three samples are occluded, and the fourth sample is not occluded. As we've changed from shadowed to non-shadowed, we detect that we've hit an object, stop ray marching, and take the shadowed sample.

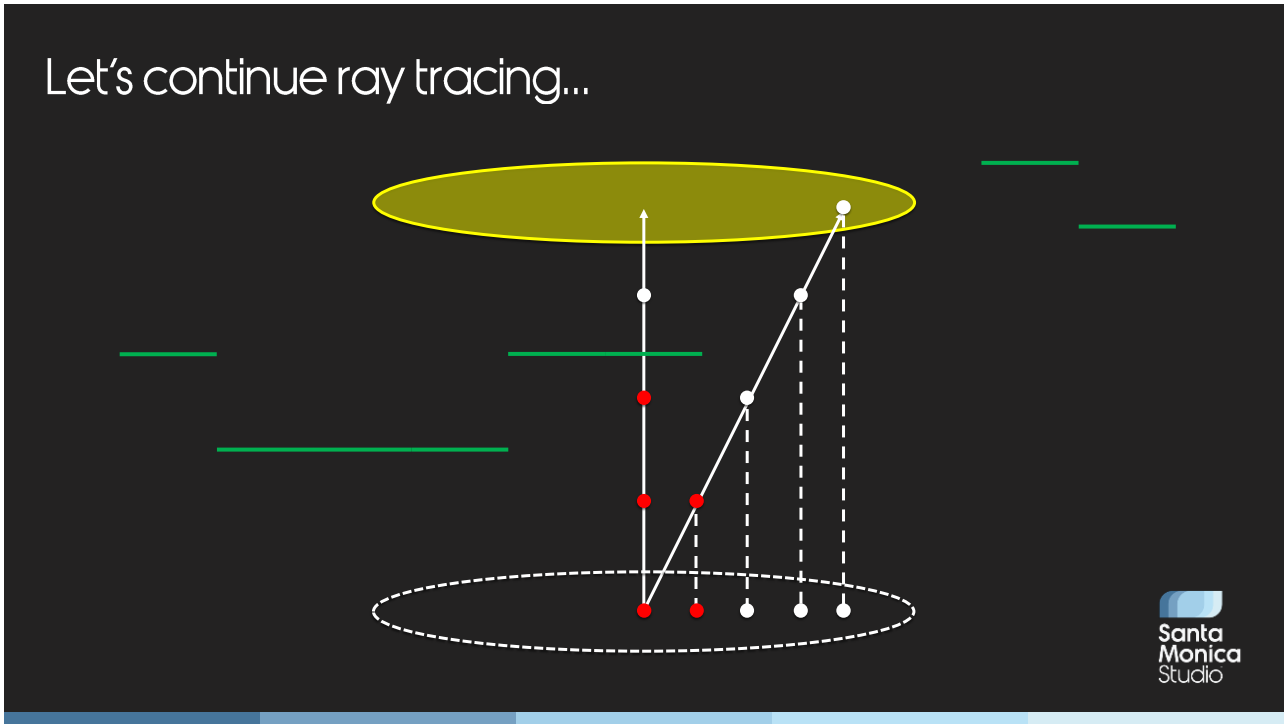
This all seems rather naïve though – we might as well have just taken the original shadowed sample. However, other rays that aren't parallel to the direction of the light are more complex.



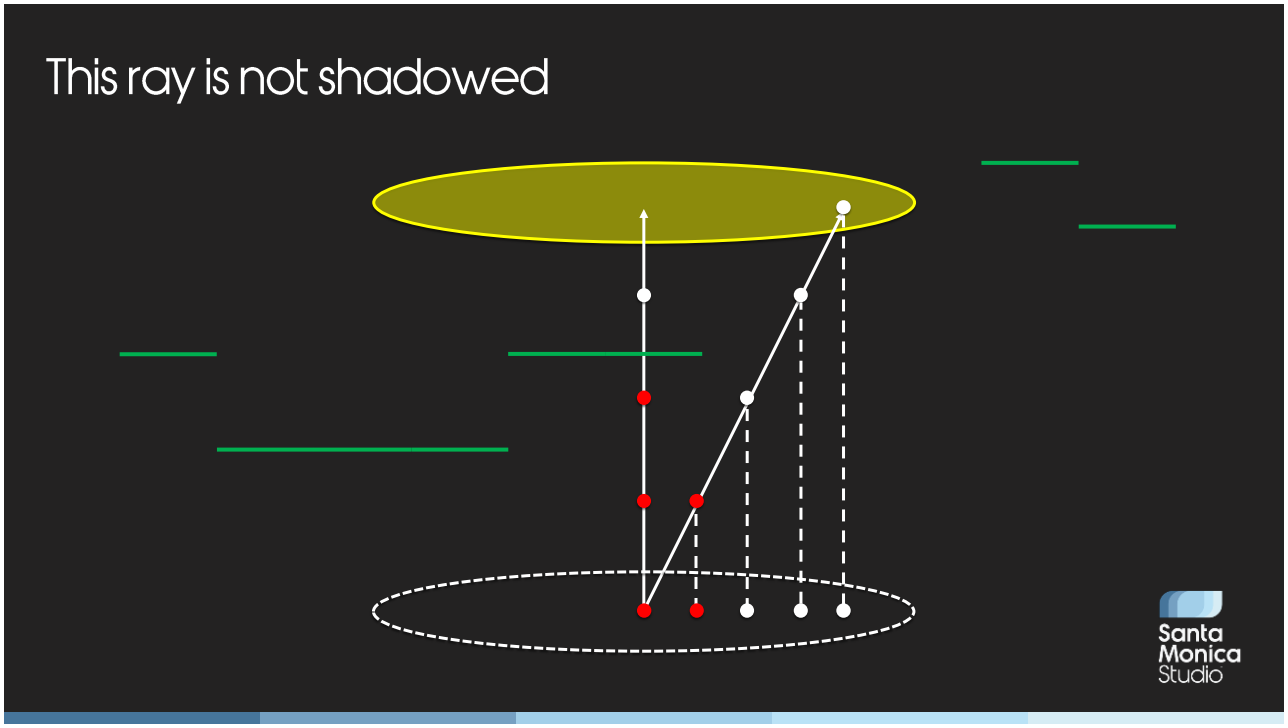
Let's trace another ray. After a few steps we notice we are no longer shadowed. Does this mean that we've intersected an object? Well, clearly it doesn't in this case.



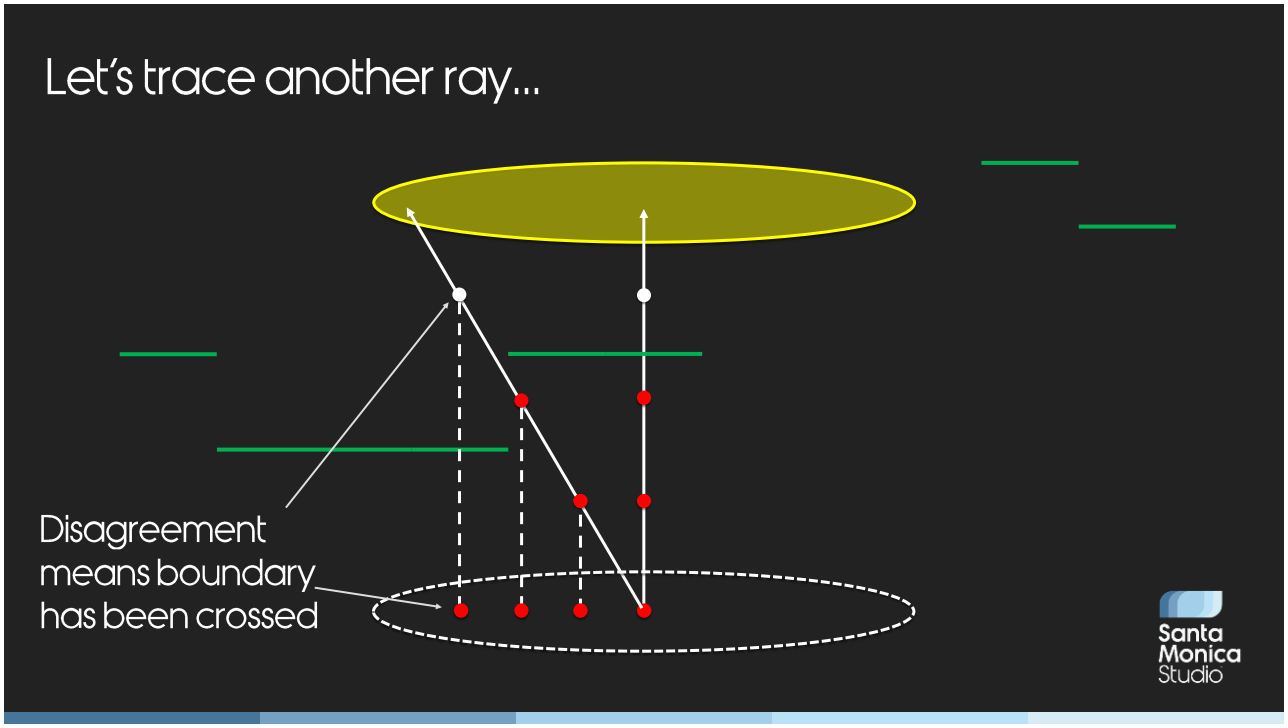
We also sample on the disk at the original pixel's depth, and discover that sample isn't shadowed either. The fact that the two samples are in agreement (they're both non-shadowed) shows that no boundary has been crossed.



So we continue ray tracing until we hit the light…

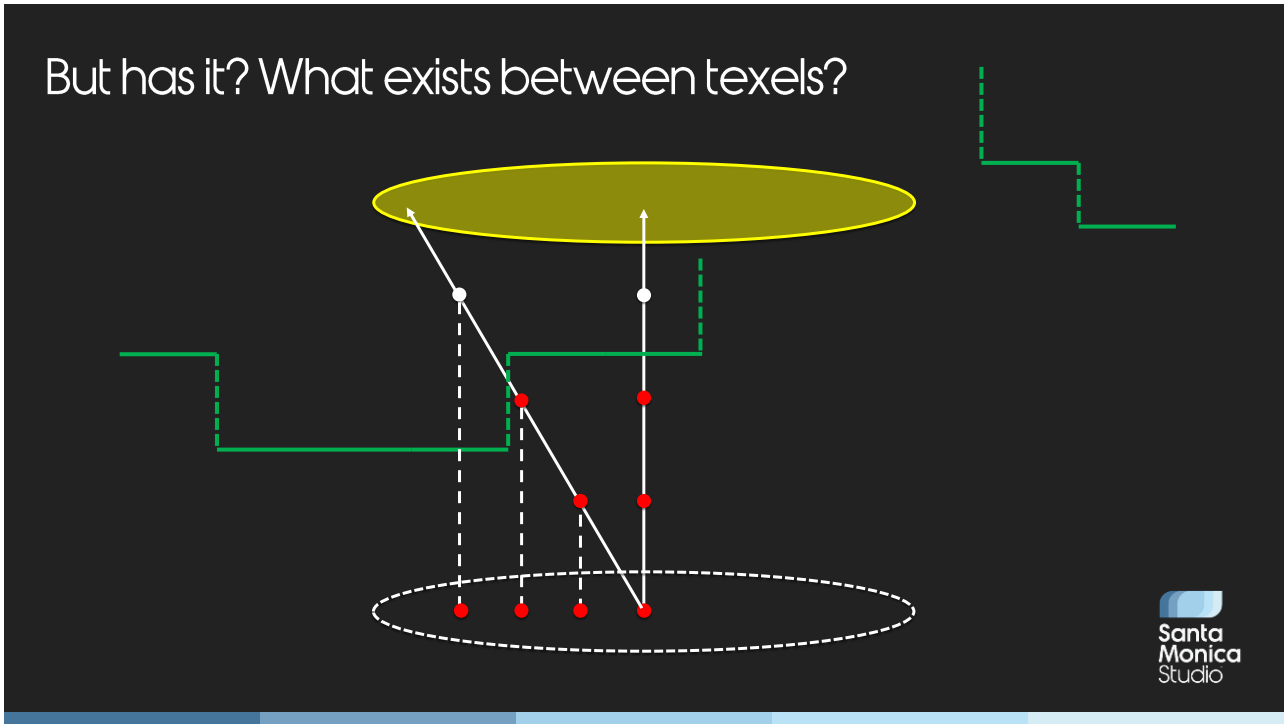


The samples on the disk vs the samples on the ray are always in agreement, showing we never crossed a boundary. This means we take the last result and say that the ray isn't shadowed.

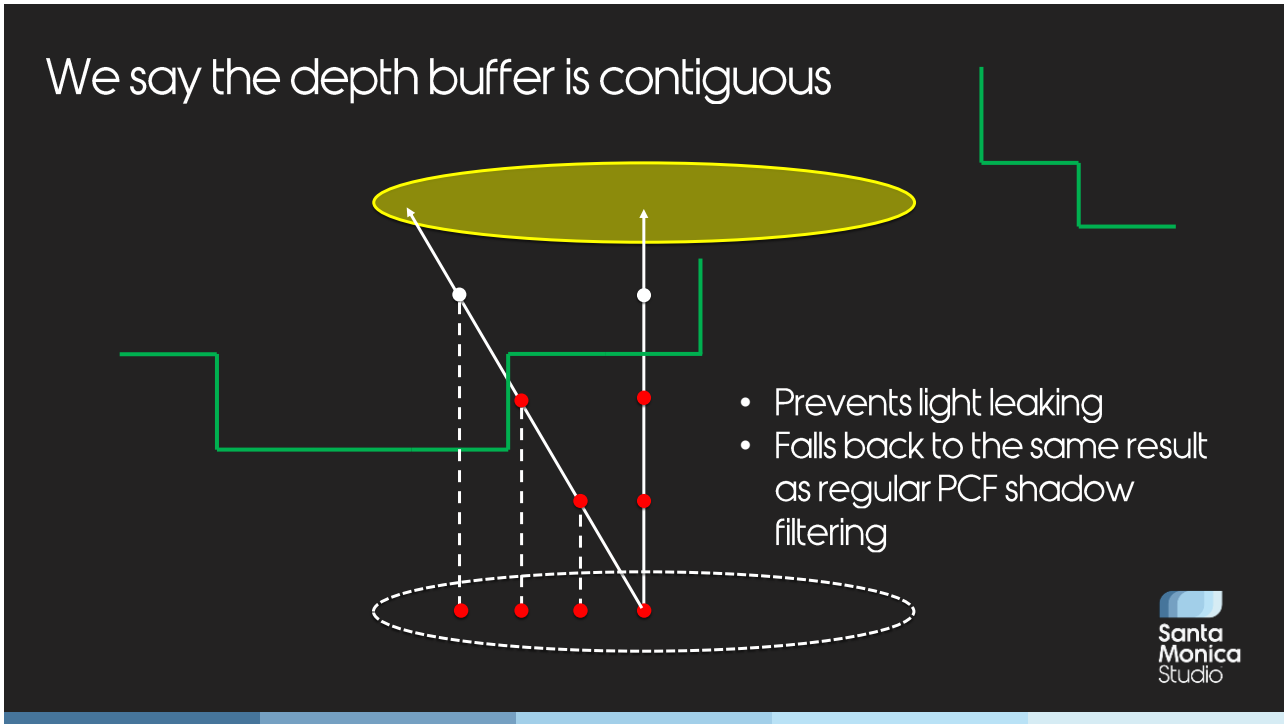


Let's trace yet another ray, a more complex one this time.

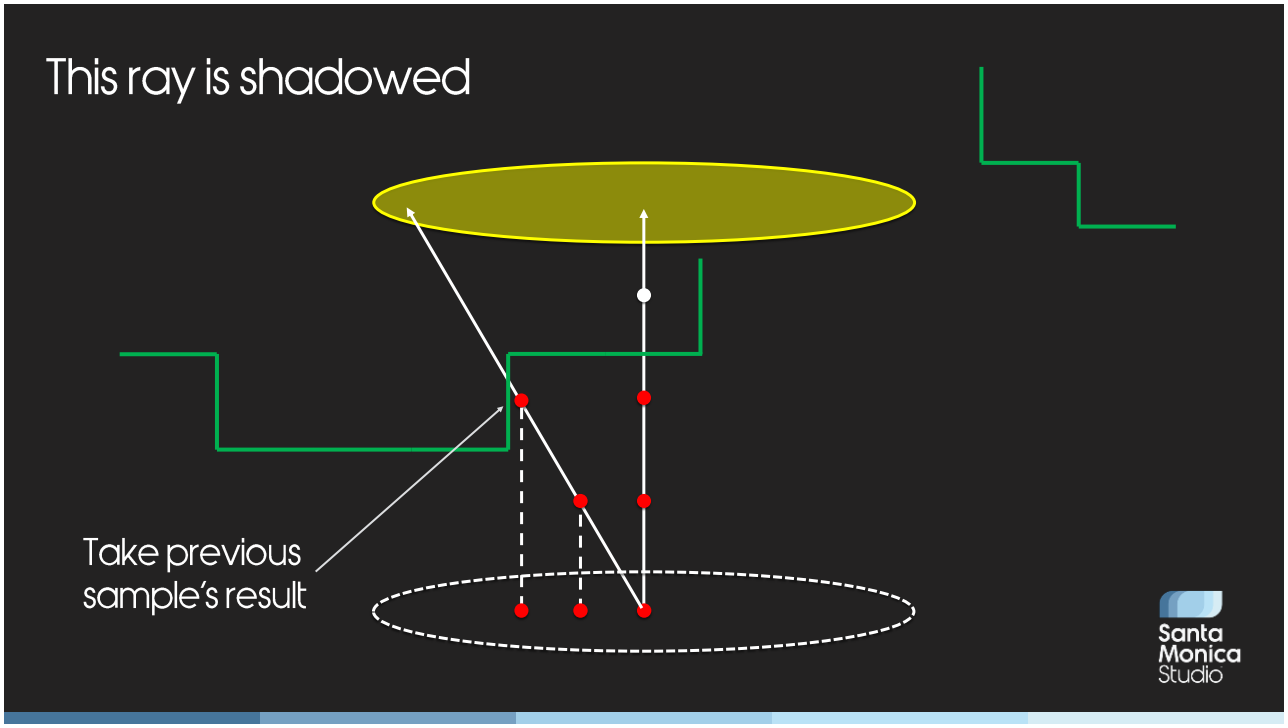
We trace until we find a disagreement between the disk and the ray samples, which means a boundary has been crossed… but has it?



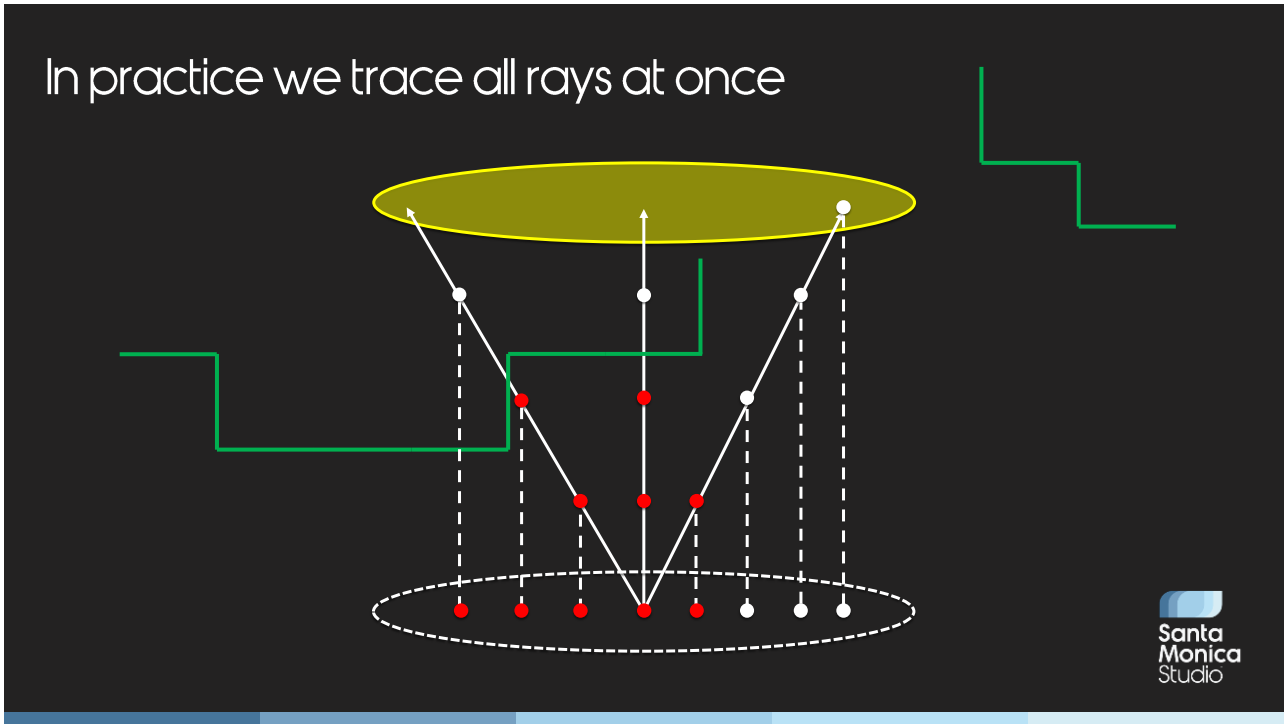
This is an innate problem of screen space ray tracing techniques. We don’t know if those individual depth texels are connected or not (or if they’re not, how “thick” each individual texel is).



However, for our purposes, we say the depth buffer is contiguous. This prevents light leaking and worst case, will just give us the same result as regular PCF filtering.



So in the case of this ray, we determine we've crossed a boundary, and take the result of the sample before that. This ray is shadowed.



We trace each ray a sample at a time, killing a ray whenever it intersects the depth buffer

```
// We choose to ray march in step order: for each step, iterate over all the rays
for (float percent = minPercent; percent <= maxPercent; percent += stepSize)
{
  uint sampleInd;
  uint latch = rayMask;
  while ((sampleInd = FirstSetBit_Lo(latch)) != 0xFFFFFFFF)
  {
    latch &= ~(1 << sampleInd);
    float2 rayDelta = PoissonPCFSampleOffset(sampleInd, 1.0f, sc);
    // Ensure the filter size is never smaller than a texel
    float filterSize = max(1.0f, texelFilterRadius * percent);
    float rayZ = testZ + percent * maximumRayMarchDistance;
    float rayVisible = ShadowTap(float3(shadowmapUV.xy + filterSize * rayDelta, rayZ), int2(0, 0));
    float discVisible = ShadowTap(float3(shadowmapUV.xy + filterSize * rayDelta, testZ), int2(0, 0));
    // We compare ray vs disc to detect the cases where the ray "escapes" between texels
    if (0.99f < abs(rayVisible - discVisible))
    {
      rayMask &= ~(1 << sampleInd);
    }
    else
    {
      // Doing this here prevents the first sample from being overoccluded, when the disc self-shadows
      rayResults[sampleInd] = discVisible;
    }
  }
}
```

Contact Hardening Shadows

- We improve quality by:
  - Jittering sample positions
  - Using filtering when sampling shadow maps
    - Not a binary in/out of shadow per sample
    - Means fewer rays are needed

Santa Monica Studio

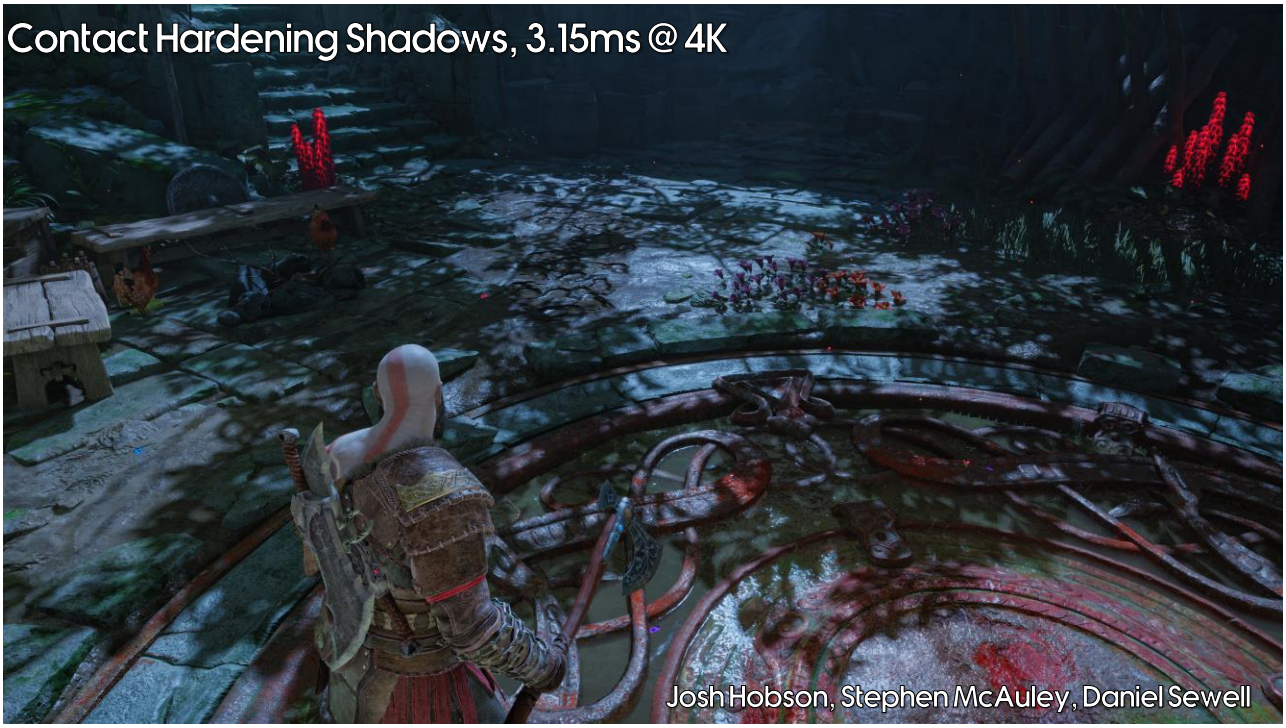
A note here – you might think, why are we taking shadow taps on both the ray and the disk? Theoretically you need to just sample the depth of the shadow map then compare against the ray and the disk's depths. That would eliminate a lot of texture samples. However, we found for quality's sake we needed to take filtered samples when sampling shadow maps, and using hardware filtering meant that we needed to do those two shadow taps.

I think with more work we could eliminate that requirement and optimize that some more, but we ran out of time.


PCF Shadows, 1.30ms @ 4K
Josh Hobson, Stephen McAuley, Daniel Sewell


Contact Hardening Shadows, 3.22ms @ 4K
Josh Hobson, Stephen McAuley, Daniel Sewell


PCF Shadows, 0.89ms @ 4K
Josh Hobson, Stephen McAuley, Daniel Sewell


Contact Hardening Shadows, 3.15ms @ 4K
Josh Hobson, Stephen McAuley, Daniel Sewell

Contact Hardening Shadows

- Ray tracing is slow
  - Early out if possible
- Create a downsampled min/max shadow map
  - Dilate according to shadow filter radius
  - Similar to depth of field techniques (Abadie2018)
- Use dilated min/max depth to:
  - Early out if fully in or out of shadow
  - Tighten range of ray trace to relevant area

Dilated min/max shadow map

Santa Monica Studio

Contact Hardening Shadows

- Our contact hardening shadows are not "physical":
  - Feature came in late
  - Lighting artists had tuned shadow filter radii manually
  - Desired to keep their look as much as possible
- "Area light" for ray tracing is assumed to be:
  - Shadow filter radius in size
  - 2m from the pixel

Santa Monica Studio

This means that we go from "sharp" to "most blurry" shadows over the course of two metres.

Actually, for the directional light, we allowed the contact hardening shadows to have a shadow filter radius twice as big as authored previously, as they ideally wanted blurrier shadows.

Light Leaking

- In general, we found removing light leaking to give the biggest visual upgrade
  - This is when light "leaks" through an object giving light where there should be none
  - Both contact hardening shadows and raytraced cubemaps help with light leaking
- However, upgrading from SSAO (screen space ambient occlusion) to SSDO (screen space directional occlusion) had the biggest impact
  - Adding SSGI also helped

Santa Monica Studio


SSDO and SSGI Disabled
Bruce Woodard

When enabling SSDO, you see a big difference. Spots on Freya, Kratos and Atreus that were abnormally lit suddenly go into shadow.

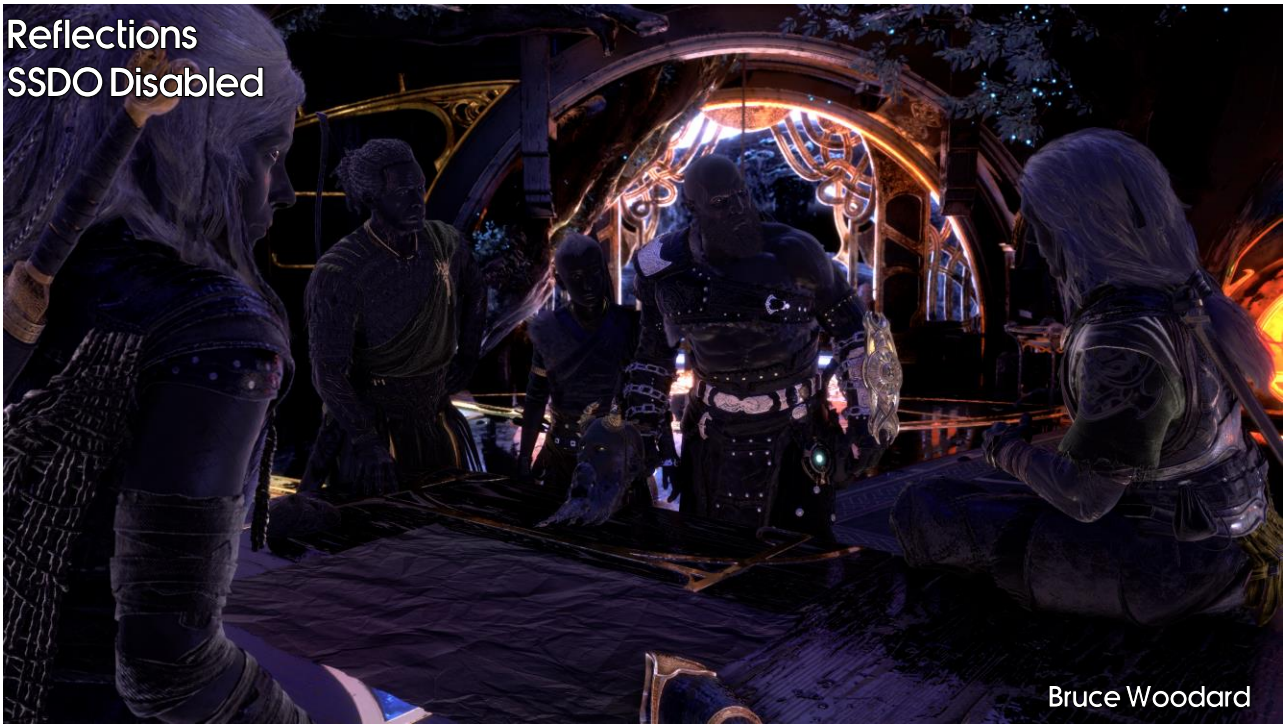
SSDO also helps a lot when you have unshadowed lights (one of the biggest causes of light leaking). We use it to fake shadows on those lights.

Let’s turn it off again, and focus on a few specific areas.

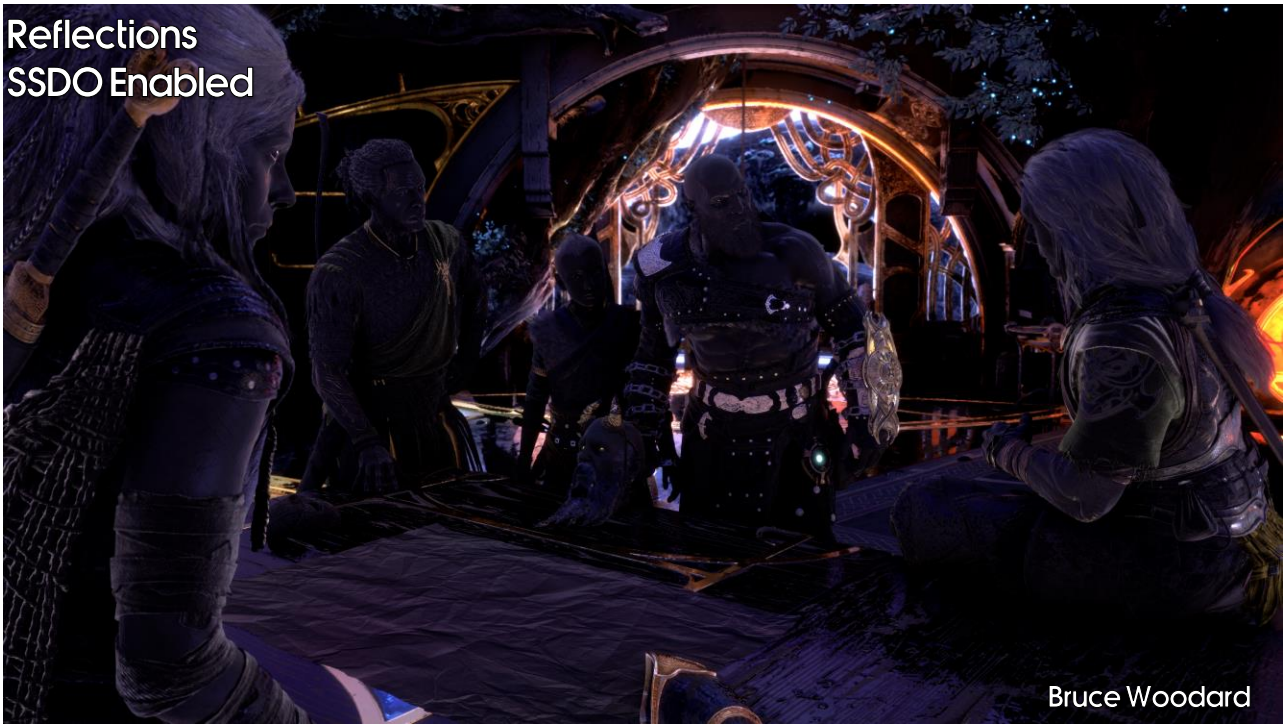
You can see better shadowing behind Freya's sword hilt, plus where Kratos' arm shadows Atreus.

Now let's look at SSGI, I'll highlight a few areas in advance since this is much harder to see. You can see Kratos' groin getting a little brighter from extra bounce light, as well as some spots in the background (see far right), plus the object near Hildisvini's right hand.

When enabling SSGI, you do get an improved image but it’s very subtle.



SSDO doesn’t just help with diffuse light, it helps with specular light too.



You can see a lot of rogue reflection goes away. Just look at the side of Freyr's leg, where it hits the table.

Next let's look at our sky and indirect lighting.

There is a huge change here. Even look at the colour changes on Atreus.

This again is subtle. But look at the object on the desk, near Hildisvini's right hand. It picks up some bounce from the table and is better grounded.

Finally let's look at direct lighting. We use SSDO to shadow direct lights, in particular when they are unshadowed in themselves.

The most obvious gain here is on Freya's upper back, in the upper left. Unnatural light just goes away.

Let’s see it all put together again. This is with SSDO and SSGI off.

And this is with SSDO and SSGI on. A much better image.

Conclusion

- We found reducing light leaking to give the biggest visual gain:
  - SSDO
  - Contact hardening shadows
  - Raytraced cubemaps
- SSDO had the biggest impact out of all of these.
  - It was also the cheapest

Santa Monica Studio

I first talked about these pillars of Narrative Features, Visual Improvements and Art Workflow and PS5 Enhancements during an internal presentation, telling the rest of SMS what the rendering team were up to. In which case, it's understandable I didn't talk about this fourth pillar of Helping Ourselves. But I have a confession… at the time, I didn't even think of this pillar as existing. It was only when coming to write this GDC presentation that I realized I'd missed out a chunk of work that my team did, and it should be classified under something new: a fourth pillar, "helping ourselves".

srtbindings.py

```python
"Creates bindings for an SRT shader."

from __future__ import absolute_import
from __future__ import print_function

import os
import sys
import fnmatch
import argparse
import collections
import copy

from cStringIO import StringIO

from smpub.os.atomicfile import open_atomic
from smpub.pipeline.codegen import msbuild

from . import preprocess
from . import lex
from . import parse
from . import defs

MODULE_DEPENDENCIES = (
    __file__,
    preprocess.__file__,
    lex.__file__,
    parse.__file__,
    defs.__file__,
    msbuild.__file__,
)

TLOG_FILE_NAME = 'srtbindings'

ParentType = collections.namedtuple('ParentType', ['name', 'arraysize'])
OffsetsType = collections.namedtuple('OffsetsType', ['next_texture_offset', 'texture_count', 'next_sampler_offset', 'sample
```

Relatively early in my time on the project, I hit a frustrating workflow blocker. We maintain a fairly rough PC build of the game, that is still on DirectX 11, and we have some Python code (shown here) to help us generate shader bindings on all platforms: PC, PS4 and PS5. However, DirectX 11 bindings come with some limitations, such as the number of resources that can be bound, and I ran into this limit as I added a new feature. Now, you'd think that we could just use some #defines to ignore those resources and that code on PC as a work around, but our shader binding code had no knowledge of the preprocessor… so I was stuck.

I had a conversation with our technical director, Josh Hobson, about this, and to my surprise he just said, "Why don't we get someone to fix it next sprint?" I felt so refreshed and empowered – I've been used to living with these problems and believing them unfixable, with work for content teams being prioritized at almost all cost. Now I knew I would be allowed to fix them, and actually my opinions and those of my team mattered – if something was affecting our work, we should certainly fix it.

I even interviewed someone recently who asked a bit about how we determine and prioritize work, and his rather excited reply at the end of

my answer was, “So you’re saying that you get to decide what work you do?” It’s not like this at all companies! But this is why it’s so important to talk about – this fundamentally is about empowering your teams to make the best decisions for the project. Sometimes that decision is to cut corners; sometimes that decision is to prioritize work for another team; and other times it’s about helping your own team.


Closing Pillar:
Optimization
Santa
Monica
Studio

Optimization

- God of War: Ragnarök's Tech Lock was December 17th 2021
  - Features were (theoretically) delivered
  - Began to shift focus to optimization
    - With hindsight, this was too late
    - The optimization effort was large

- Note: this presentation focuses on the GPU optimization efforts.
  - Budget and telemetry process were very similar on the CPU side.

Optimization: Budgets

- The first step of the optimization process was to lay out budgets
- Goals were:
  1. Create clear budget categories with as little as possible overlap between teams
  2. Maintain art budgets from God of War 2018
  3. All budgets with async compute off
  4. Budgets total 33.25ms:
     - Assume async will get us more headroom
       - Saved around 1.0-1.5ms on average
  5. All budgets set for PS4
     - Our slowest platform; if we optimize for that other platforms *should* follow

Santa Monica Studio

Optimization: Budgets

Category	Teams	Budget
Art	Environment Art, Character Art	8.5ms
Lighting	Lighting, Tech	8.5ms
Shadows	Environment Art, Character Art, Lighting	4.0ms
System	Tech	7.5ms
UI	UI	0.75ms
VFX	VFX	4.0ms

Santa Monica Studio

We created the following budget categories with each category having the subsequent general owners.

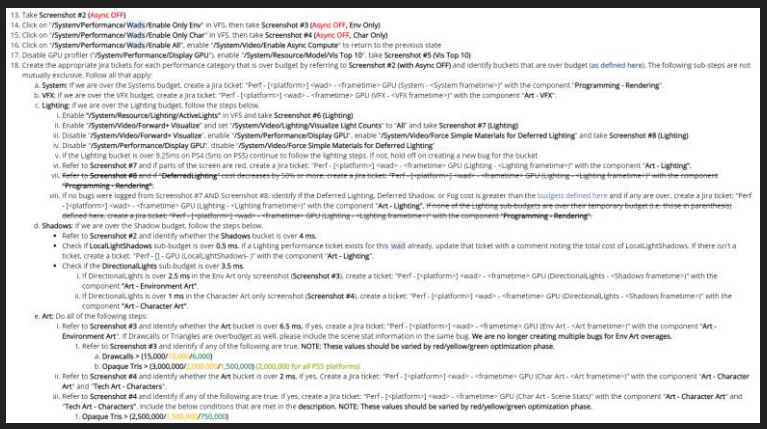
Optimization: Budgets

- Next we needed a way to display them in game
- A "tree" view was important:
  - So we can easily group items into categories
- ImGui helps make this easy!

Santa Monica Studio

Optimization: Triage Process

- Created a decision tree for QA when encountering a performance issue:



13. Take **Screenshot #2** (Async OFF)
14. Click on "**/System/Performance/Wads/Enable Only Env**" in VFS, then take **Screenshot #3** (Async OFF, Env Only)
15. Click on "**/System/Performance/Wads/Enable Only Char**" in VFS, then take **Screenshot #4** (Async OFF, Char Only)
16. Click on "**/System/Performance/Wads/Enable All**", enable "**/System/Video/Enable Async Compute**" to return to the previous state
17. Disable GPU profiler ("**/System/Performance/Display GPU**"), enable "**/System/Resource/Model/Vis Top 10**", take **Screenshot #5 (Vis Top 10)**
18. Create the appropriate Jira tickets for each performance category that is over budget by referring to **Screenshot #2 (with Async OFF)** and identify buckets that are over budget (as defined here). The following sub-steps are not mutually exclusive. Follow all that apply:
    a. **System:** If we are over the Systems budget, create a jira ticket: "Perf - [<platform>] <wad> - <frametime> GPU (System - <System frametime>)" with the component "**Programming - Rendering**".
    b. **VFX:** If we are over the VFX budget, create a jira ticket: "Perf - [<platform>] <wad> - <frametime> GPU (VFX - <VFX frametime>)" with the component "**Art - VFX**".
    c. **Lighting:** If we are over the Lighting budget, follow the steps below.
        i. Enable "**/System/Resource/Lighting/ActiveLights**" in VFS and take **Screenshot #6 (Lighting)**
        ii. Enable "**/System/Video/Forward+ Visualize**" and set "**/System/Video/Lighting/Visualize Light Counts**" to "**All**" and take **Screenshot #7 (Lighting)**
        iii. Disable "**/System/Video/Forward+ Visualize**", enable "**/System/Performance/Display GPU**", enable "**/System/Video/Force Simple Materials for Deferred Lighting**" and take **Screenshot #8 (Lighting)**
        iv. Disable "**/System/Performance/Display GPU**", disable "**/System/Video/Force Simple Materials for Deferred Lighting**"
        v. If the Lighting bucket is over 9.25ms on PS4 (5ms on PS5) continue to follow the lighting steps. If not, hold off on creating a new bug for the bucket
        vi. Refer to **Screenshot #7** and if parts of the screen are red, create a jira ticket: "Perf - [<platform>] <wad> - <frametime> GPU (Lighting - <Lighting frametime>)" with the component "**Art - Lighting**".
        vii. ~~Refer to **Screenshot #8** and if "**DeferredLighting**" cost decreases by 50% or more, create a jira ticket: "Perf - [<platform>] <wad> - <frametime> GPU (Lighting - <Lighting frametime>)" with the component "**Programming - Rendering**".~~
        viii. If no bugs were logged from Screenshot #7 AND Screenshot #8, identify if the Deferred Lighting, Deferred Shadow, or Fog cost is greater than the budgets defined here and if any are over, create a Jira ticket: "Perf - [<platform>] <wad> - <frametime> GPU (Lighting - <Lighting frametime>)" with the component "**Art - Lighting**". ~~If none of the Lighting sub-budgets are over their temporary budget (i.e. those in parenthesis) defined here, create a jira ticket: "Perf - [<platform>] <wad> - <frametime> GPU (Lighting - <Lighting frametime>)" with the component "**Programming - Rendering**".~~
    d. **Shadows:** If we are over the Shadow budget, follow the steps below.
        - Refer to **Screenshot #2** and identify whether the **Shadows** bucket is over **4 ms.**
        - Check if **LocalLightShadows** sub-budget is over **0.5 ms.** If a Lighting performance ticket exists for this wad already, update that ticket with a comment noting the total cost of LocalLightShadows. If there isn't a ticket, create a ticket: "Perf - [] - GPU (LocalLightShadows-)" with the component "**Art - Lighting**".
        - Check if the **DirectionalLights** sub-budget is over **3.5 ms**.
            i. If DirectionalLights is over **2.5 ms** in the Env Art only screenshot (**Screenshot #3**), create a ticket: "Perf - [<platform>] <wad> - <frametime> GPU (DirectionalLights - <Shadows frametime>)" with the component "**Art - Environment Art**".
            ii. If DirectionalLights is over **1 ms** in the Character Art only screenshot (**Screenshot #4**), create a ticket: "Perf - [<platform>] <wad> - <frametime> GPU (DirectionalLights - <Shadows frametime>)" with the component "**Art - Character Art**".
    e. **Art:** Do all of the following steps:
        i. Refer to **Screenshot #3** and identify whether the **Art** bucket is over **6.5 ms.** If yes, create a jira ticket: "Perf - [<platform>] <wad> - <frametime> GPU (Env Art - <Art frametime>)" with the component "**Art - Environment Art**". If Drawcalls or Triangles are overbudget as well, please include the scene stat information in the same bug. **We are no longer creating multiple bugs for Env Art overages.**
            1. Refer to **Screenshot #3** and identify if any of the following are true. **NOTE: These values should be varied by red/yellow/green optimization phase.**
                a. **Drawcalls > (15,000/**~~10,000~~**/6,000)**
                b. **Opaque Tris > (3,000,000/**~~2,000,000~~**/1,500,000)** (2,000,000 for all PS5 platforms)
        ii. Refer to **Screenshot #4** and identify whether the **Art** bucket is over **2 ms.** If yes, Create a jira ticket: "Perf - [<platform>] <wad> - <frametime> GPU (Char Art - <Art frametime>)" with the component "**Art - Character Art**" and "**Tech Art - Characters**".
        iii. Refer to **Screenshot #4** and identify if any of the following are true. If yes, create a jira ticket: "Perf - [<platform>] <wad> - <frametime> GPU (Char Art - Scene Stats)" with the component "**Art - Character Art**" and "**Tech Art - Characters**". Include the below conditions that are met in the **description. NOTE: These values should be varied by red/yellow/green optimization phase.**
            1. **Opaque Tris > (2,500,000/**~~1,500,000~~**/750,000)**

Santa Monica Studio

Next, we actually needed to find the performance issues in game. That was the job of our amazing QA team.

However, we needed to tell them what to do when they found a performance issue. We created this decision tree to follow, which first involves pulling up the GPU and CPU profilers.

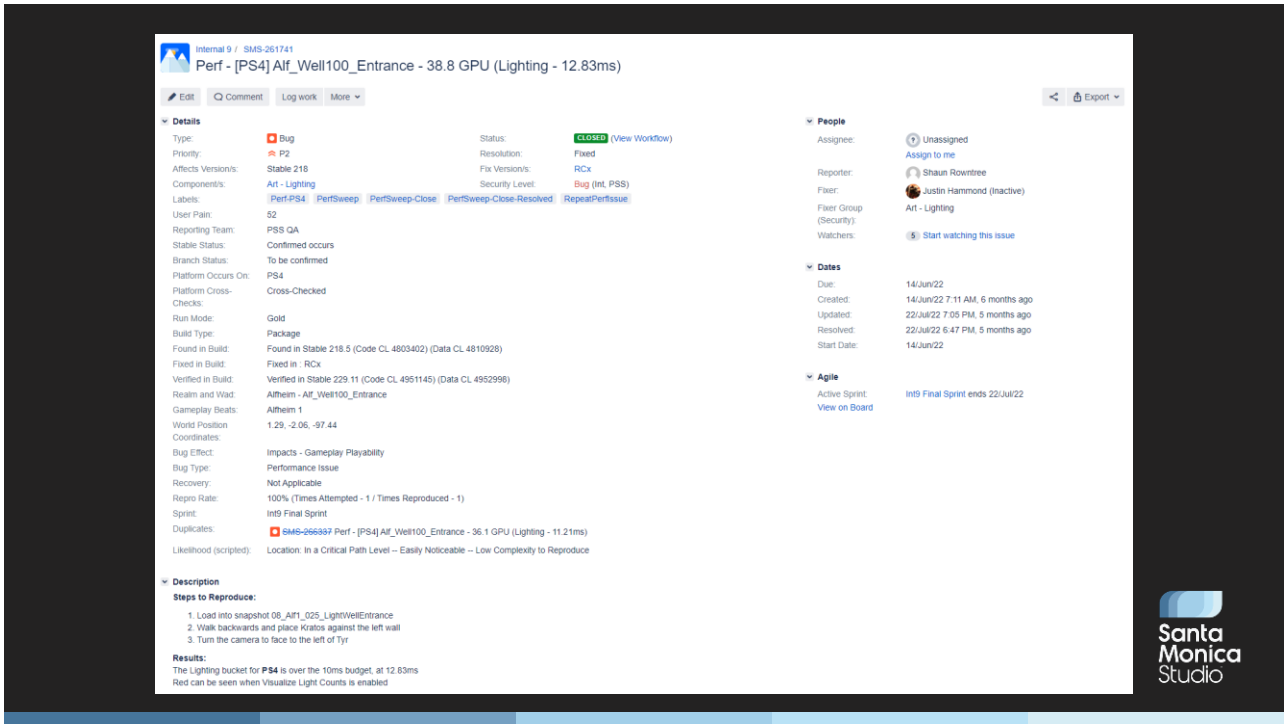
Optimization: Triage Process

- Created a decision tree for QA when encountering a performance issue:

c. **Lighting:** If we are over the Lighting budget, follow the steps below.
   i. Enable **"/System/Resource/Lighting/ActiveLights"** in VFS and take **Screenshot #6 (Lighting)**
   ii. Enable "**/System/Video/Forward+ Visualize**" and set "**/System/Video/Lighting/Visualize Light Counts**" to "**All**" and take **Screenshot #7 (Lighting)**
   iii. Disable "**/System/Video/Forward+ Visualize**", enable "**/System/Performance/Display GPU**", enable "**/System/Video/Force Simple Materials for Deferred Lighting**" and take **Screenshot #8 (Lighting)**
   iv. Disable "**/System/Performance/Display GPU**", disable "**/System/Video/Force Simple Materials for Deferred Lighting**"
   v. If the Lighting bucket is over 9.25ms on PS4 (5ms on PS5) continue to follow the lighting steps. If not, hold off on creating a new bug for the bucket
   vi. Refer to **Screenshot #7** and if parts of the screen are red, create a Jira ticket: "Perf - [<platform>] <wad> - <frametime> GPU (Lighting - <Lighting frametime>)" with the component "**Art - Lighting**".

Santa Monica Studio

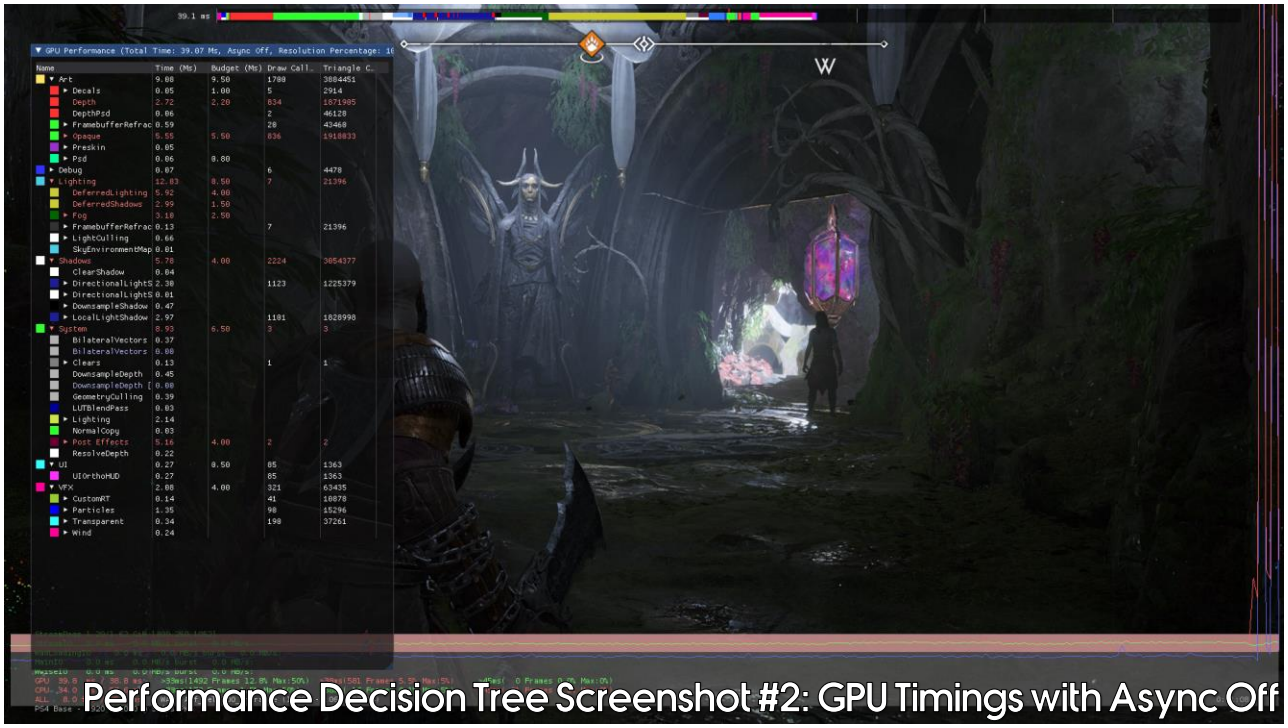
If we zoom in a bit more, we can see the steps taken when the lighting category is overbudget. It involves enabling various options in our debug menus, bringing up various debug displays, and taking various screenshots to attach to the JIRA. It also helps determine which team to send the JIRA to at the end of the process.



That would result in a JIRA looking something like this (a real world example!). It details where and how the performance issue occurred, and what areas of the game were over budget. In this case it's lighting, and the decision tree determines that the JIRA should go to the lighting team (as opposed to the rendering team, which would be the other option). If multiple budget categories were over budget in the same location, we'd create JIRAs for each one.

Santa Monica Studio

Each JIRA would have many screenshots attached. Which screenshots were taken would depend on the steps determined in the decision tree. For example, if lighting was overbudget as in this case, then we would take screenshots of debug views to help figure out what the problem was.



So, for example, this is screenshot #2 in the hierarchy, which is the GPU budgets with async compute turned off. We first turn async compute off as that allows us to get accurate budgets – we then know the shadow budget contains just shadows, not shadows plus whatever async work was happening at the same time.

As we know lighting is one area that's over budget, we capture a screenshot with the list of active lights.

And we also capture a screenshot with the visible light counts per tile. As you can see, there's a lot of "red" here meaning a lot of lights, so this is likely the cause of us being over budget here.

We capture other screenshots too, but hopefully this helps you understand what's happening.

Optimization: Brainstorming

- We have:
  - Created budgets
  - Designed a triage process

- Now what do we optimize on the Rendering team?
  - Held team brainstorming sessions
  - All held remotely over Teams

Optimization: Brainstorming

- First attempt:
  - Unstructured, open brainstorming session
  - Team told in advance the topic was performance

- Result:
  - None of us were prepared
  - Lots of silence

Santa Monica Studio

Optimization: Brainstorming

- Next step:
  - Brainstorm on how to brainstorm better

- I talked with my manager, Jeremy Wood, and producer, Tre Fitzgerald, about how to improve the process
  - They had some fantastic ideas

Santa Monica Studio

Optimization: Brainstorming

- Second attempt:
  - Sign up sheet for areas to investigate
    - Tried both areas of the game and areas of the code
  - Booked calendar time
  - Asked the team to write up notes in advance on a Wiki page

- Result:
  - Everyone was prepared
  - Many good ideas
  - Engaging conversation

Santa Monica Studio

Even better, in some circumstances, team members started commenting in advance on others' thoughts, so you have some pre-debate!

2022-02-04 - Rendering Team Meeting - Performance Brainstorm

Created by Stephen Mcauley, last modified just a moment ago

Sign Up Sheet

Part of the Rendering Loop	Programmer(s)
G-Buffer (Depth, Opaque etc.)	Bryan, Mat
System Passes (Downsample Depth, Bilateral Vectors, etc.)	Valerio, Sam
Shadows	Matt, Xuanyi, Paolo
Fog	
Lighting	Jon
Post Effects	Steve

Notes

Jonathan DiGiacomo

1. It looks like there's one batch that takes extremely long for DeferredLightingCompute, even when all of the lights in the scene are disabled. It seems to use about 156 vGPRs which is probably why it takes so long. The ForwardPlusDeferredCompute batches seem to be a little fickle about whether they show up in the ForwardPlusDeferredCompute subcategory or not. When there's more work to pair them out, occupancy still seems to be an issue.
2. Light culling seems to have some gaps in between the work as well as a pause after it finishes it's workload. The pause appears to be clearing buffers, but it's not obvious where that work is occurring unless the pauses after are it flushing out to memory. The pauses aren't particularly big however

Valerio Guagliumi

System Passes:

1. Normal downsampling was moved from DownsampleDepth to DownsamplePlugs, but that means that now we need to write the "downsample offset" texture and read it back, so I imagine that this introduces some overhead. This had to be done to support GI#0 from decals. Is there some alternative?
2. Is there a way that we could move GPU culling to async and kick it right after Depth? If the problem are Opaques that export depth, probably there is little benefit from taking those into account. Alternatively we could have split Opaque into two steps, first one with depth write, second without: this would also allow to potentially async DownsampleDepth as well (as long as normal is not needed, see 1).
3. BilateralVectors is pretty expensive. I didn't see anything obviously bad in it, but maybe an optimization pass could squeeze out something

Optimization: Brainstorming

- Having team members' thoughts in advance has helped productive and inclusive conversation:
  - "Bob, you had a fantastic idea about shadow performance, could you share it with everyone?"
  - Everyone can have a turn at speaking
  - Easy opportunities to affirm and support team members
- Text chat on the side can be very helpful too
  - Some people prefer that method of communication
  - Easier sometimes than trying to join the conversation
  - Feel free to invite those people to share their thoughts verbally too

Santa Monica Studio

Another fun benefit is that if you don't get through everybody's pre-written ideas in one meeting, then you absolutely know you have content for another brainstorming meeting on the same topic.

We've carried on with this method of brainstorming since then, for example, with brainstorming rendering improvements for our next project.

Optimization: Shaders

- Complex deferred lighting shaders were one of our biggest bottlenecks
- They had low occupancy, i.e. high VGPR count
- Reducing VGPRs is key, but how?
- Hard to relate VGPRs in the final shader ISA to the actual HLSL source code

Santa Monica Studio

Optimization: VGPR Usage Tool

- Wrote a tool to parse the output shader ISA
- Catalogues every time each VGPR is written to
  - e.g. v39 is written to on lines 4368 and 5680
- Returns this information in an output text file

Santa Monica Studio

VGPR Usage Tool Output

vgprusages.txt

```
// 4713:     fetches[2] = baseSurface[coord + uint2(1, 0)].rg;
// 4734:     float reconstructedChroma = fetches[0].y * weights.x + fetches[1].y * weights.y + fetches[2].y * weights.z
+ fetches[3].y * weights.w;
// 4734:     float reconstructedChroma = fetches[0].y * weights.x + fetches[1].y * weights.y + fetches[2].y * weights.z
+ fetches[3].y * weights.w;
// 4734:     float reconstructedChroma = fetches[0].y * weights.x + fetches[1].y * weights.y + fetches[2].y * weights.z
+ fetches[3].y * weights.w;
// 4734:     float reconstructedChroma = fetches[0].y * weights.x + fetches[1].y * weights.y + fetches[2].y * weights.z
+ fetches[3].y * weights.w;
v26:
// 4587:     if (enableRO)
// 4605:         lightInput.textureSO = lerp(lightInput.textureSO, 1, pow(1 - abs(NDotE), 3));
// 4605:         lightInput.textureSO = lerp(lightInput.textureSO, 1, pow(1 - abs(NDotE), 3));
// 4713:     fetches[2] = baseSurface[coord + uint2(1, 0)].rg;
// 4716:     float4 neighborLumas = float4(fetches[0].x, fetches[1].x, fetches[2].x, fetches[3].x);
// 4726:     float weightSum = weights.x + weights.y + weights.z + weights.w;
// 4726:     float weightSum = weights.x + weights.y + weights.z + weights.w;
v27:
// 4713:     fetches[2] = baseSurface[coord + uint2(1, 0)].rg;
// 5930:     lvi.NoV = dot(lightInput.normalWS, LightInput_EyeDirWS(lightInput));
// 5930:     lvi.NoV = dot(lightInput.normalWS, LightInput_EyeDirWS(lightInput));
v28:
// 4764:     float3 normalWS = normalize(gbuffers.normalTexture.Load(coord).xyz*2.0f-1.0f);
v29:
// 4764:     float3 normalWS = normalize(gbuffers.normalTexture.Load(coord).xyz*2.0f-1.0f);
v30:
// 4684:     else if (pixelFlags.shadingModel == kShadingModelOpaqueRefraction)
// 4697:     else if (DoesShadingModelHaveEmissive(pixelFlags.shadingModel))
// 4706: }
v31:
// 4684:     else if (pixelFlags.shadingModel == kShadingModelOpaqueRefraction)
// 4697:     else if (DoesShadingModelHaveEmissive(pixelFlags.shadingModel))
// 4706: }
v32:
// 4764:     float3 normalWS = normalize(gbuffers.normalTexture.Load(coord).xyz*2.0f-1.0f);
v33:
```

Santa Monica Studio

"float3 normalWS" is stored in v28, v29, v32

```
v28:
// 4764:     float3 normalWS = normalize(gbuffers.nc
v29:
// 4764:     float3 normalWS = normalize(gbuffers.nc
v30:
// 4684:     else if (pixelFlags.shadingModel == kSh
// 4697:     else if (DoesShadingModelHaveEmissive(p
// 4706: }
v31:
// 4684:     else if (pixelFlags.shadingModel == kSh
// 4697:     else if (DoesShadingModelHaveEmissive(p
// 4706: }
v32:
// 4764:     float3 normalWS = normalize(gbuffers.nc
v33:
```

Santa Monica Studio

Optimization: VGPR Usage Tool

- Very easy to see what variables in HLSL are mapping to which VGPRs
- Gives insight into what variables have long lifetimes, causing inflated VGPR use
- Invaluable to helping optimize complex shaders

Optimization: Geometry Passes

- We gained a surprising amount of time back in our geometry passes by revisiting initial assumptions:
  - Eliminating shader prefetches
  - Better sorting
    - Sort by anything that causes context rolls:
      - Culling
      - Winding order
      - Vertex shader to pixel shader binding

Santa Monica Studio

Let’s take look at this scene in Midgard. This area saw a surprisingly large benefit from improving sorting. Let’s bring up the profiler…


GPU Performance (Total Time: 27.11 Ms, Async Off, Resolution Percentage:
Name Time (Ms) Budget (Ms)
Art 9.11 8.50
Decals 0.10 1.00
Depth 2.20 2.25
FramebufferRefraction 0.00
HeightField 0.74
Opaque 5.70 5.25
Preskin 0.28
Sky 0.09
Transparent 0.00
Debug 0.06
Lighting 7.57 8.50
DeferredLighting 3.74 4.50
DeferredShadows 1.75 1.75
Fog 1.64 1.75
FramebufferRefraction 0.04
Lensflare 0.00
LightCulling 0.32
SkyEnvironmentMapOpaque 0.07
SkyEnvironmentMapTransparent 0.00
Shadows 1.24 4.00
ClearShadow 0.01
DilateShadow 0.03
DirectionalLightShadow 1.11
DirectionalLightShadowTransmittance 0.01
DownsampleShadow 0.05
HTILEResummarizeAndResolvePass 0.00
LightShadowTransmittance 0.03
Shadow 0.00
System 7.45 7.50
BilateralVectors 0.38
Clears 0.05
DownsampleDepth 0.50
GeometryCulling 0.21
LUTBlendPass 0.03
Lighting 1.81
NormalCopy 0.03
Post Effects 4.10 4.50
Rescale 0.00
ResolveClearAuxBuffers 0.06
ResolveDepth 0.26
ResolvePropertiesClear 0.02
UI 0.05 0.50
UIOrthoHUD 0.05
VFX 1.61 4.00
CustomRT 0.05
Particles 1.04
Transparent 0.44
Wind 0.07
Art: 9.11ms
Shader prefetch enabled
Depth pass sorted by depth only
Opaque pass sorted by material ID only

Around 0.7ms is a good saving – we saw less in some areas and even more in others!

By disabling the shader prefetch we save even more. Perhaps this doesn’t work for us as we have so many shaders in our game?

Optimization: Shadows

- Shadows were expensive in a number of ways:
  - Rendering geometry to shadow maps
  - A large fixed cost spent processing those shadow maps
- Shadow proxies significantly reduced our geometry cost:
  - Static shadow proxy meshes per light
  - Lower poly shadow proxies for meshes
    - Particularly helpful for characters, auto-generated by our tech art team
- However, let's dive deeper into the fixed cost…

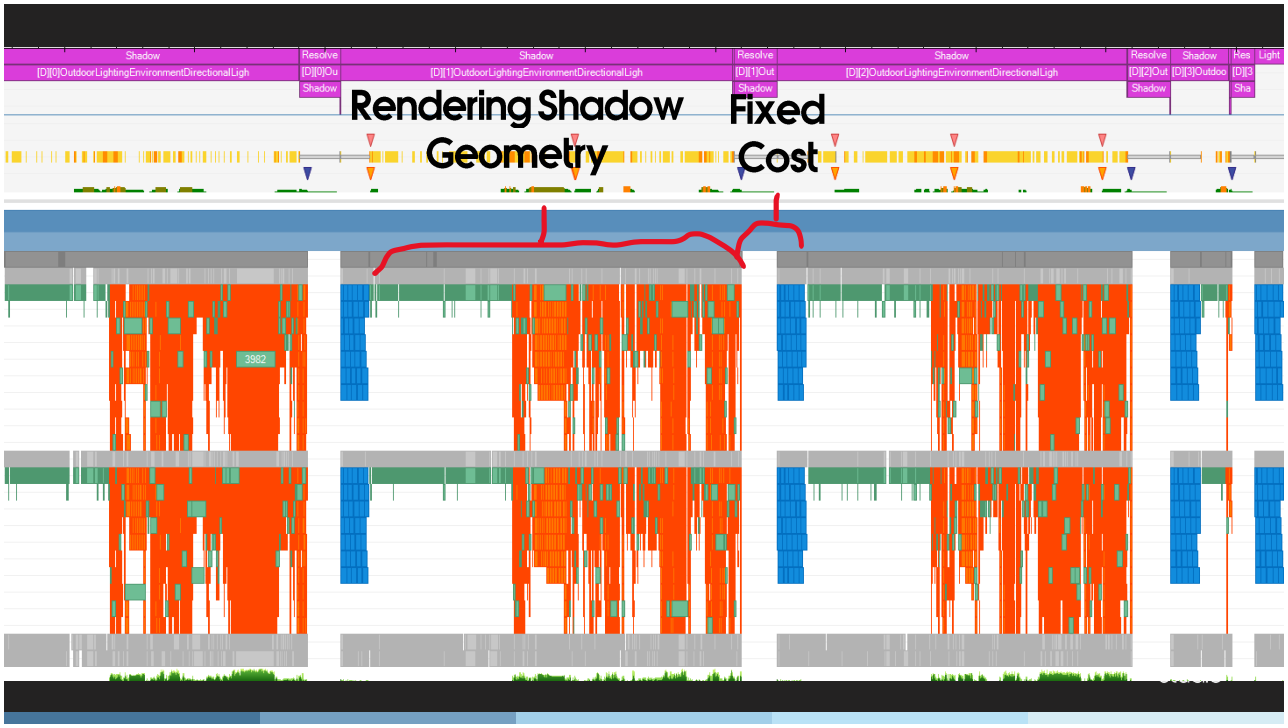
Santa Monica Studio

I had never used shadow proxies before, perhaps just biasing the LODs used in the shadow, but this made a huge difference.

Let’s take a look at this scene on the PS4…

We'll bring up the profiler and see the shadows are 4.95ms…

And if we look in more detail, we'll see that there are ShadowDecompressDepth and ShadowResolveESM calls for every shadow map, that are taking up time.



Let's take a look at a Razor GPU capture and see what's happening…

You can see the blocks of green and orange, rendering the shadow geometry. And then a gap (the depth decompression) and then blue (the resolve to ESM). For the most part these fixed costs aren't that high compared to rendering the shadow map itself… but look at the end of the image! The fixed cost looks higher than actually rendering the shadow map.

You might think that the resolve to ESM should be 'free', in that it should be running in async with the next frame's shadow map. I have async compute disabled here and that's definitely the intention of this system. However, the depth decompression can't run async… and nothing in this life is free. Having this work run async with the shadows not only slows down the shadows (so we'd want it to be as fast as possible), but also blocks other possible async compute work from running in this space.)

Optimization: Shadows

- Four cascades at 1640x1640
- 4.95ms total shadow cost
  - ~0.5ms for depth decompress
  - ~0.45ms for ESM resolve

- Fixed cost per shadow map for depth decompress and ESM resolve

- Can we reduce the fixed cost?

Santa Monica Studio

Optimization: Shadows

- The depth decompression is needed
  - Faster than clearing an uncompressed depth surface

- What about ESMs?
  - We only use ESMs for volumetric fog
    - Regular shadow maps cause aliasing in fog
    - Sample lighting and shadowing per froxel
    - Each froxel encompasses many regular shadow texels
  - Can we use another cheaper technique?

Santa Monica Studio

Optimization: Shadows

- Idea: Replace ESMs with downsampled min/max shadow map
- Result: Downsampling the shadow map by 8x8 is no faster than calculating the ESMs

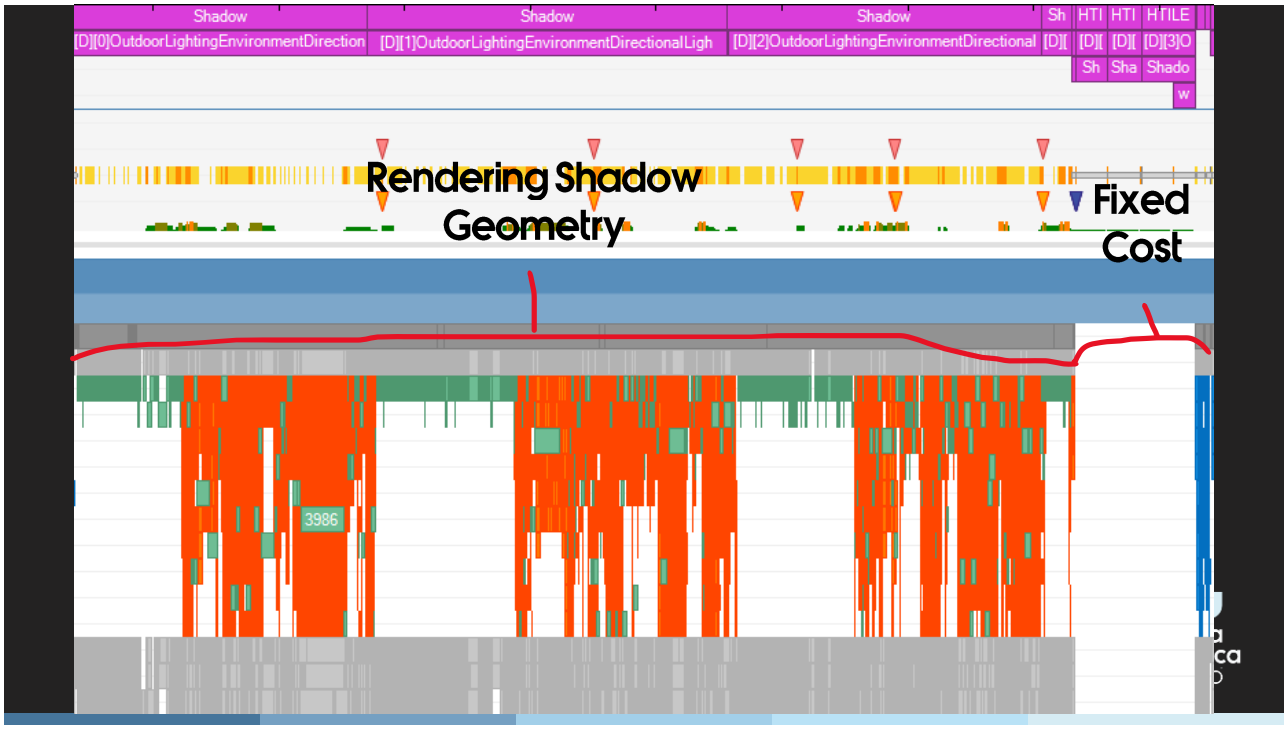
Optimization: Shadows

- Idea: Use the HTile instead of a downsampled shadow map
  - Resolve the HTile Buffer into depth min/max

- Result: 0.04ms for the HTile resolve
  - Instead of 0.45ms for the ESM conversion

Santa Monica Studio


Shadows 4.52
DilateShadow 0.02
DirectionalLightShadow 4.42
[D][0]OutdoorLightingEnvironment[1.30
ShadowHTileResumDecompressDept 0.12
[D][1]OutdoorLightingEnvironment[1.56
ShadowHTileResumDecompressDept 0.13
[D][2]OutdoorLightingEnvironment[1.35
ShadowHTileResumDecompressDept 0.13
[D][3]OutdoorLightingEnvironment[0.21
ShadowHTileResumDecompressDept 0.09
Shadows: 4.52ms



And let's look at a Razor capture. You can see that we've changed the position of the fixed cost passes to happen at the end of shadow rendering. It turned out that grouping these passes together was faster.

Optimization: Shadows

- One more thing...
- Remember how we optimized contact hardening shadows?
  - A dilated, downsampled min/max shadow map
- Now we always have a downsampled shadow map
  - Dilating is incredibly cheap (~0.02ms)
- Could this optimize shadow maps in general?

Santa Monica Studio

Optimization: Shadows

- Just sample dilated min/max buffer before PCF shadow kernel:

```
float FilterShadowPPCF(float3 shadowmapUV, float texelFilterRadius, in float rotation)
{
  float2 minMaxZ =
    ShadowDilatedMinMaxBy8Texture.SampleLevel(gLightSamplers.smpPointClamp, shadowmapUV.xy, 0);
  if (minMaxZ.x > shadowmapUV.z)
    return 0.0f;
  else if (minMaxZ.y < shadowmapUV.z)
    return 1.0f;

  ...
}
```

- The "fully lit" case is rarely, if ever, taken:
  - Self-shadowing "artefacts".

Santa Monica Studio


GPU Performance (Total Time: 31.17 Ms, Async Off, Resolution Percentage:
Name Time (Ms) Budget (Ms)
Art 10.22 8.50
Decals 0.16 1.00
Depth 3.96 2.25
FramebufferRefraction 0.28
HeightField 0.00
Opaque 5.44 5.25
Preskin 0.22
Sky 0.16
Transparent 0.00
Debug 0.05
Lighting 7.81 8.50
DeferredLighting 3.55 4.50
DeferredShadows 1.30 1.75
Fog 2.37 1.75
FramebufferRefraction 0.07
LightCulling 0.32
SkyEnvironmentMapOpaque 0.20
SkyEnvironmentMapTransparent 0.00
Shadows 4.52 4.00
ClearShadow 0.00
DilateShadow 0.02
DirectionalLightShadow 4.41
[0]OutdoorLightingEnvironment 1.31
ShadowMaskTileResumDecompressDept 0.12
[1]OutdoorLightingEnvironment 1.55
ShadowMaskTileResumDecompressDept 0.13
[2]OutdoorLightingEnvironment 1.35
ShadowMaskTileResumDecompressDept 0.12
[3]OutdoorLightingEnvironment 0.20
ShadowMaskTileResumDecompressDept 0.09
DirectionalLightShadowTransmittance 0.04
DownsampleShadow 0.04
HTILEResummarizeAndResolvePass 0.00
LightShadowTransmittance 0.00
System 7.34 7.50
BilateralVectors 0.42
Clears 0.03
DownsampleDepth 0.55
GeometryCulling 0.27
LUTBlendPass 0.03
Lighting 1.77
NormalCopy 0.05
Post Effects 3.89 4.50
Rescale 0.00
ResolveClearAuxBuffers 0.07
ResolveDepth 0.23
ResolveLightBufferClears 0.00
ResolvePropertiesClear 0.02
VFX 1.22 4.00
CustomRT 0.33
Particles 0.46
Transparent 0.31
Optimization off
Deferred shadows: 1.30ms

So in this scene it was actually a 0.01ms loss (a 0.01ms gain in sampling shadows but 0.02ms loss in calculating the dilated shadow map), but overall across the game it was a net win. (It has a bigger effect in areas with more overlapping shadow maps, this just has one directional light per lighting tile).

Conclusion

Conclusion

- We shipped on November 9th, 2022!
- This truly was a team effort:
  - This presentation has shown links between the rendering team and tech art, art, QA, production, management and game direction
- My role as a lead was to help coordinate the efforts between all these groups:
  - On behalf of the rendering team
  - To achieve the game vision

Santa Monica Studio

Thanks

Rendering:
- Josh Hobson
- Mathew Hendry
- Valerio Guagliumi
- Sam Willis
- Paolo Surricchio
- Jonathan DiGiacomo
- Bryan McPhail
- Xuanyi Zhou
- Matt Yan
- Daniel Sewell
- Jendrik Illner
- Bruce Woodard
- Vitor Menezes

Technical Art:
- Nathan Kennedy
- Sean Feeley
- Leonard Cremer
- Blair Pierpont
- Amy Chen

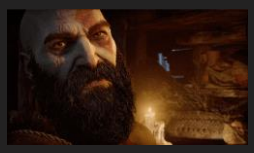
Other:
- Rafael Grassetti
- Dan McKim
- Tre Fitzgerald
- Jeremy Wood
- Vadim Slyusarev

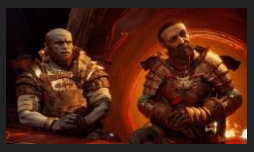
GDC:
- Julien Merceron

Santa Monica Studio

Thank You!

stephen.mcauley@sony.com





STUDIOS. Santa Monica Studio

JOIN US AT GDC 2023

GOD OF WAR RAGNARÖK

BUILD YOUR GOD OF WAR GDC AT:
SCHEDULE.GDCONF.COM

BRUNO VELAZQUEZ ♦ ANIMATION DIRECTOR
DAVID GIBSON ♦ ANIMATION DIRECTOR
ERICA PINTO ♦ LEAD NARRATIVE ANIMATOR
MEHDI YSSEF ♦ LEAD GAMEPLAY ANIMATOR
Animation in 'God of War Ragnarök' ♦ Animation Summit
MONDAY, MARCH 20 ♦ 9:30AM – 10:30AM ♦ ROOM 303, SOUTH HALL

SUE PACETE ♦ SR USER RESEARCHER
Playtesting God of War Ragnarök Accessibility Options ♦ UX Summit
MONDAY, MARCH 20 ♦ 1:20PM – 1:50PM ♦ ROOM 2001, WEST HALL

PAOLO SURRICCHIO ♦ SR STAFF PROGRAMMER
Reinventing the Wheel for Snow Rendering ♦ Advanced Graphics Summit
MONDAY, MARCH 20 ♦ 1:20PM – 2:20PM ♦ ROOM 301, SOUTH HALL

BEN NINES ♦ SR STAFF DEVOPS ENGINEER
Automated Testing at Santa Monica Studio ♦ Tools Summit
MONDAY, MARCH 20 ♦ 4:40PM – 5:10PM ♦ ROOM 3004, WEST HALL

XUANYI ZHOU ♦ PROGRAMMER
Real-time Neural Texture Upsampling in 'God of War Ragnarök' ♦ Machine Learning Summit
TUESDAY, MARCH 21 ♦ 2:10PM – 2:40PM ♦ ROOM 2010, WEST HALL

ETHAN AYER ♦ SR ENVIRONMENT ARTIST
The Art of Making Vistas ♦ Art Summit
TUESDAY, MARCH 21 ♦ 3:00PM – 3:30PM ♦ ROOM 3007, WEST HALL

GÖKSU UĞUR ♦ AI LEAD
Preparing AI Systems For God of War Ragnarök ♦ Programming
WEDNESDAY, MARCH 22 ♦ 9:00AM – 10:00AM ♦ ROOM 303, SOUTH HALL

VICKI SMITH ♦ SR STAFF LEVEL DESIGNER
The Final Battle of 'God of War Ragnarök': Techniques For Delivering High-stakes Sequences ♦ Design
WEDNESDAY, MARCH 22 ♦ 10:30AM – 11:00AM ♦ ROOM 2002, WEST HALL

STEPHEN McAULEY ♦ LEAD RENDERING PROGRAMMER
Rendering 'God of War Ragnarök' ♦ Programming
WEDNESDAY, MARCH 22 ♦ 2:00PM – 3:00PM ♦ ROOM 303, SOUTH HALL

ERIC GOTTESMAN ♦ SR STAFF DEVOPS ENGINEER
Modernizing multiplayer services for "God of War: Ascension" (PS3) ♦ Production & Team Leadership ♦ Presented by Amazon Web Services
WEDNESDAY, MARCH 22 ♦ 2:00PM – 3:00PM ♦ GDC PARTNER STAGE, EXPO FLOOR, NORTH HALL

SAM STERNKLAR ♦ SR PROGRAMMER
'God Of War Ragnarök's' Visual Scripting Solution ♦ Programming
WEDNESDAY, MARCH 22 ♦ 10:00AM – 11:00AM ♦ ROOM 2006, WEST HALL

ADAM OLIVER ♦ SR COMBAT DESIGNER
Breaking Barriers: Combat Accessibility in 'God of War Ragnarök' ♦ Design
THURSDAY, MARCH 23 ♦ 2:00PM – 2:30PM ♦ ROOM 2002, WEST HALL

GÖKSU UĞUR ♦ AI LEAD
Practical Tools for Transitioning Into Leadership Roles ♦ Leadership
THURSDAY, MARCH 23 ♦ 2:00PM – 2:30PM ♦ ROOM 303, SOUTH HALL

ZACH BONN ♦ SR STAFF TECHNICAL UI DESIGNER
'God of War Ragnarök': Building The UI For a AAA Title ♦ Design
THURSDAY, MARCH 23 ♦ 4:00PM – 5:00PM ♦ ROOM 303, SOUTH HALL

SALAAR KOHARI ♦ PROGRAMMER
Companion Traversal in 'God of War Ragnarök' ♦ Programming
FRIDAY, MARCH 24 ♦ 10:00AM – 11:00AM ♦ ROOM 2002, WEST HALL

TENGHAO WANG ♦ SR PROGRAMMER
Joint-based Skin Deformation in 'God of War Ragnarök' ♦ Programming
FRIDAY, MARCH 24 ♦ 1:30PM – 2:30PM ♦ ROOM 2006, WEST HALL

NARLEIGH AWNER ♦ TECHNICAL NARRATIVE DESIGNER
How to Build a Home: Designing Narrative For Sindri's House in 'God of War Ragnarök' ♦ Design
FRIDAY, MARCH 24 ♦ 3:00PM – 3:30PM ♦ ROOM 2001, WEST HALL

Santa Monica Studio GDC


Santa Monica Studio
Our journey
Your story
We're hiring for what's next!
We're expanding our family across disciplines and would love to meet you. Please visit sms.playstation.com/careers for all openings or drop us a line at sms.recruiting@sony.com
@ santamonicastudio
@ SonySantaMonica
@ santamonicastudio

References

(Abadie2018) A Life of a Bokeh, Guillaume Abadie, SIGGRAPH 2018
(Grujic2018) Water Rendering in Far Cry 5, Cristian Cutocheras & Branislav Grujic, GDC 2018
(Hobson2019) The Indirect Lighting Pipeline of 'God of War', Josh Hobson, GDC 2019
(Lagarde2012) Local Image-Based Lighting with Parallax-Corrected Cubemaps, Sébastien Lagarde & Antoine Zanuttini, SIGGRAPH 2012
(McAuley2018) The Challenges of Rendering an Open World in Far Cry 5, Stephen McAuley, SIGGRAPH 2018
(Pines2010) From Scene to Screen, Joshua Pines, SIGGRAPH 2010
(Stachowiak2015) Stochastic Screen-Space Reflections, Tomasz Stachowiak, SIGGRAPH 2015
(Surricchio2023) ..., Paolo Surricchio, GDC 2023
(Zhou23) ..., Xuanyi Zhou, GDC 2023

Santa Monica Studio