



Smolder

Real-Time Volumetric Effect Rendering
in Glacier and 007 First Light

Hello! Thank you so much for the introduction, Natasha, and thank you all for joining me here today.

Introduction

SIGGRAPH



Alexander Mueller

Senior Render Programmer

IO Interactive

Bluesky: @lexdev.net

Discord: lex410

LinkedIn: <https://www.linkedin.com/in/lexmueller/>



© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

Here's some of my contact details. Please feel free to reach out anytime.



© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

Let's start with the game.

[CLICK]

We shipped 007 First Light roughly 2 months ago, and we shipped it with a lot of volumetric effects. You can see some of them in the launch trailer here.

Smolder

SIGGRAPH

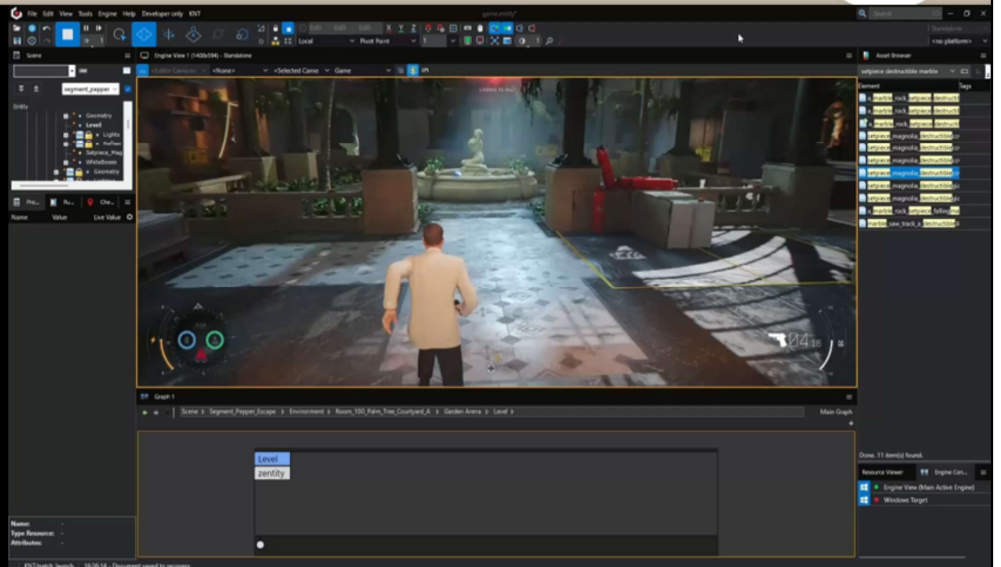


© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

Those effects are powered by a render system we call Smolder. Our artists used Smolder effects everywhere in the game, for set dressing, for gameplay gadgets, in cutscenes and in intense action sequences.

Glacier Engine

SIGGRAPH



© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

The game is built on top of our in-house engine Glacier. It is the engine that powers all of IOIs projects and it is built around systemic gameplay and quick iterations. If you have worked with an in-house engine before, you know that it's not always working as seamlessly as we want it to be. But when it does, it's truly magical and artists & designers can get into a flow that allows them to build truly amazing things.



- Use volumetrics for dynamic gameplay VFX everywhere
- Enable artists to work freely with volumetric effects as if they are any other kind of system
 - Make volumetrics a first-class citizen in Glacier
 - Avoid any constraints that artists have to usually work around
 - Scalable across platforms, no fallback required

That brings us to the dream behind Smolder. A while back I was approached by one of our Principal Tech Artists, Martin Nielsen (who has two very Danish middle names I can't pronounce, so I will skip those 😊).

He had a couple of dynamic gameplay VFX for one of our projects that isn't 007 First Light, and those required a volumetric renderer that works everywhere. Martin is a very technically skilled artist and he was primarily interested in the results, but for me the workflow was the important part.

[CLICK]

I wanted to enable artists to work freely with volumetrics in Glacier, just as if they are any other kind of system.

[CLICK]

I wanted to make volumetrics a first class citizen in Glacier. It ties back into the power of great workflows, and how those can enable artists to build beautiful worlds.

For me, that is the single most important part of any render system and it is something I am trying to push internally constantly.

We can build the most amazing tech, but if the artists can't work with it or don't enjoy working with it they will not create equally amazing results. So ultimately, it does not matter.

[CLICK]

And that means that we really want a system that "just works", where artists can place volumes in the world without thinking about any limitations and can focus on the artistic vision they have.

[CLICK]

That also means we don't want them to think about the target hardware it has to run on. I've seen quite a lot of games that pursue render tech that produces great results on higher end platforms like PS5 or XBOX Series X, but then artists have to go in and build content for a fallback system once they port to lower end hardware that can't run that original tech.

These days especially it's also more important than ever to keep the minimum specs on PC low, as it becomes less and less affordable for players to upgrade their hardware.



- Existing volumetric tech
 - Purpose built, not generic enough for our use case
 - Limited scene integration (especially with transparency)
 - Limited lighting integration (only selected lights, fixed number of shadows, ...)

[Wronski14] Volumetric Fog: Unified compute shader based solution to atmospheric scattering - <https://advances.realtimerendering.com/s2014/index.html>

[Valent14] Reflections and Volumetrics of Killzone Shadow Fall - <https://advances.realtimerendering.com/s2014/index.html>

[Hillaire15] Towards Unified and Physically-Based Volumetric Lighting in Frostbite - <https://advances.realtimerendering.com/s2015/index.html>

[Schneider17] Nubis: Authoring Real-Time Volumetric Cloudscapes with the Decima Engine - <https://advances.realtimerendering.com/s2017/index.html>

[Bauer19] Creating the Atmospheric World of Red Dead Redemption 2: A Complete and Integrated Solution - <https://advances.realtimerendering.com/s2019/index.html>

[Schneider22] Nubis, Evolved: Real-Time Volumetric Clouds for Skies, Environments, and VFX - <https://advances.realtimerendering.com/s2022/index.html#Nubis>

[Schneider23] Nubis3: Methods (and madness) to model and render immersive real-time voxel-based clouds - <https://advances.realtimerendering.com/s2023/index.html#Nubis3>

The question now was how to approach this, we started by looking at what others already shipped. There's a lot out there and even if we just look at the volumetrics focused presentations at advances from previous years, the list is pretty long.

We have systems like froxel based volumetric fog, which we already had in Glacier at the time, but the visuals we could get out of it were not good enough. Volumetric clouds have also become extremely popular due to the work of Andrew Schneider on the Horizon games, and there have been a bunch of other systems like the smoke grenades in Counter Strike 2 for example. All those systems had some limitations that made them not a good fit for our use case though.

[CLICK]

They are typically very purpose built.

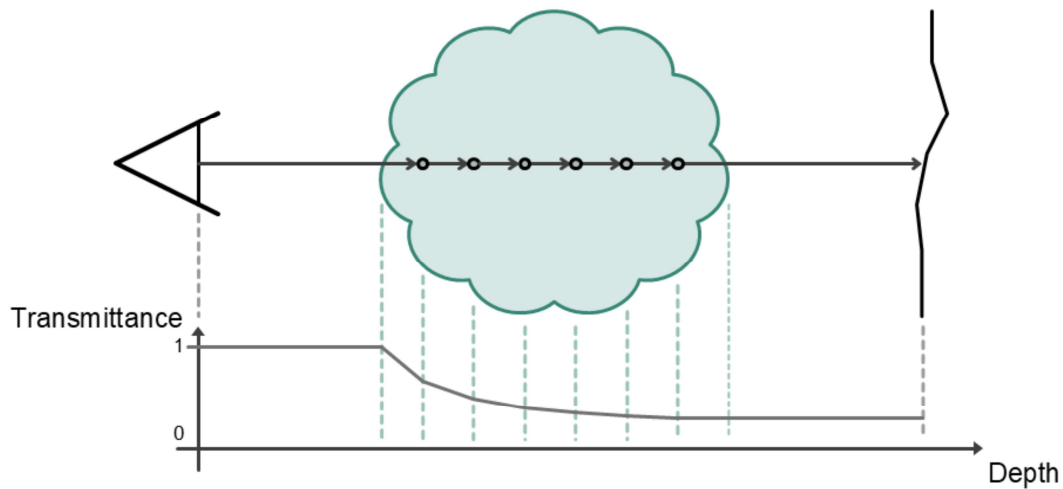
[CLICK]

A lot of them come with limited scene integration, especially when it comes to transparency.

[CLICK]

And a lot of them have a very limited lighting integration, usually they only support a few lights, very few shadow casting lights etc.

It was clear to us we had to find a new way to build what we were looking for – We wanted a generic, fully integrated and performant volumetric effects renderer.



<https://scratchapixel.com/lessons/3d-basic-rendering/volume-rendering-for-developers/intro-volume-rendering.html>

© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

Before we dive into the details, here's a quick overview of how raymarching works for those who have never done it.

For an in depth look at it I recommend the scratchapixel series on it, it's really well made. When rendering volumetrics, we have to calculate two things: How much of the light behind the volume is transmitted through it and how much of the incoming light is scattered towards the camera.

To render those volumes, we usually use raymarching.

[CLICK]

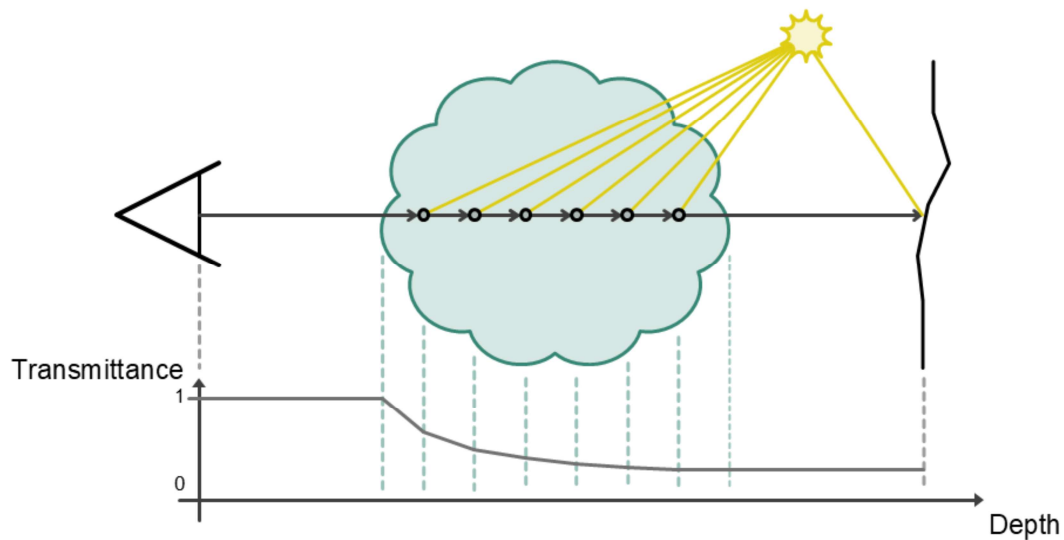
For each step along the primary ray, we sample the density of the volume.

[CLICK]

That allows us to calculate the optical depth, which is a measure of transparency, and to generate the transmittance function piece by piece.

[CLICK]

This assumes that the density within each step is uniform, so if it isn't there's some error introduced here and a smaller step size is needed to not under sample the volume.

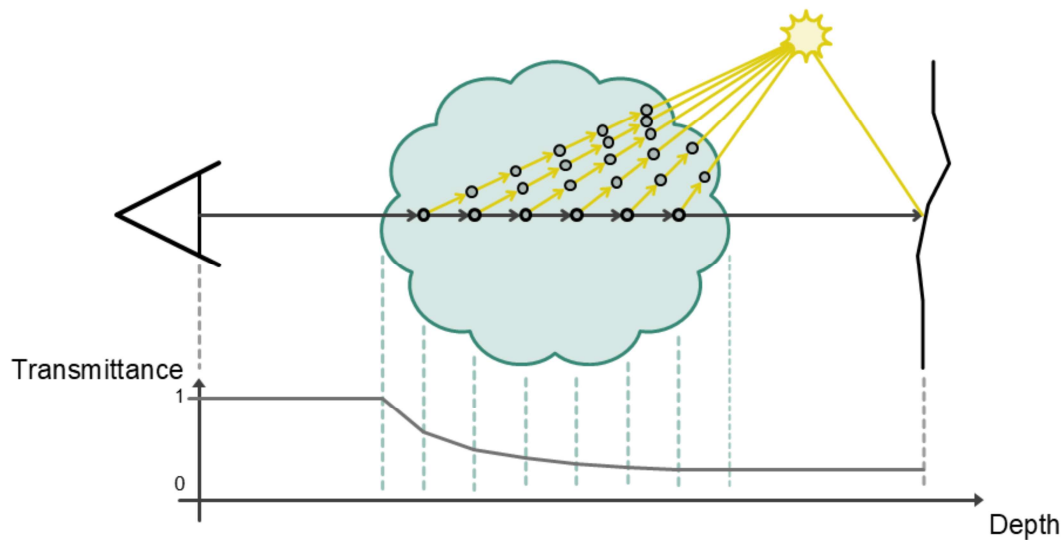


© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

To integrate the scattering, we must determine the amount of incoming light for each of our samples.

[CLICK]

Since that light passes through the volume to reach our sample point, we need to calculate transmittance for the oncoming light as well.



In the full algorithm we take a step,

[CLICK]

we sample the density and we update the transmittance up to this point.

[CLICK]

We then do a secondary raymarch to calculate the transmittance for the light ray and we can calculate the scattering for the sample. That scattering is attenuated by the main ray's transmittance and added to the result.

[CLICK]

The primary Raymarch continues to the next sample, and we step through the volume until we reach the end or the transmittance is 0, at which point we can stop because the additional samples will not contribute to the result.

Requirements 1: Full Lighting Integration - Sun



- Sun
 - Cascaded Shadow Maps
 - 3 cascades
 - 25m in front of camera
 - Scrolling Cascade
 - 200m around camera
 - Static shadows only
 - Only updates dirty regions during movement
 - Global Shadow Map
 - Rest of the map
 - Up to 4km x 4km
 - Time sliced updates



© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

With that out of the way, let's look at the list of requirements for our volumetric renderer, and the various problems we have to solve to achieve our goal.

We start with the full lighting integration.

For direct lighting, we first have the sun, which in our case comes with three different types of shadows.

We have very traditional cascaded shadow maps, we use 3 cascades in 007 First Light, and they cover 25 meters.

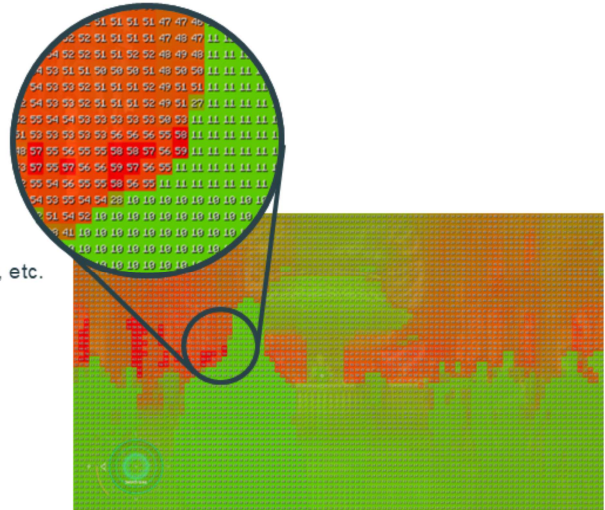
Beyond that, we have a scrolling cascade around the camera that only contains static shadows. That one covers 200 meters.

For the rest of the map, we have a very low-res global shadow map that covers it. Our biggest map in 007 First Light is 4 by 4 kms.

Requirements 1: Full Lighting Integration – Local Lights



- Sun
- Local Lights
 - All lights in the scene
 - 150-200 lights in heavy views
 - Tile based light culling
 - All light types
 - Static & dynamic shadows, projector maps, penumbra color, etc.
 - Heavy register pressure



© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

For local lights, the main issue is the number of local lights we have in our scenes.

Our lighting artists love placing lots of lights everywhere.

In some of our heaviest views, we have between 150 to 200 lights.

We also still had a very old school tile-based light culling system at the time, which was originally built for Hitman 2016.

In this debug view we have up to 60 lights per tile for example.

Beyond that we have to support all of our different light types, and they all come with a lot of different features. That comes with high register pressure.

Requirements 1: Full Lighting Integration - GI



- Sun
- Local Lights
- Global Illumination
 - Real-time
 - DDGI style probes
 - Irradiance, Radiance, Depth
 - 8x8 Octahedral textures
 - 3 cascade levels, 64x64x16 probes, 0.6m, 1.2m, 2.4m+
 - High sampling cost
 - Trilinear probe interpolation
 - Volumetric samples require spherical integration of radiance



[Majercik, Guertin, Nowrouzezahrai, McGuire19] **Dynamic Diffuse Global Illumination with Ray-Traced Irradiance Fields** - <http://jcgt.org/published/0008/02/01/>
[Sousa25] **Fast as Hell: idTech8 Global Illumination** - <https://advances.realtimerendering.com/s2025/index.html#doom>

© 2026 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

Lastly, we also have global illumination to take into account.

With 007 First Light we added a fully real-time DDGI style global illumination system. The probes are pretty standard, in our case they store irradiance, radiance and depth in 8x8 octahedral textures. We have 3 cascades, each with 64x64x16 probes, and the probes are 60 centimeters apart on the smallest grid level. It's a pretty dense grid.

Sampling those probes comes with a high sampling cost. We have to sample 8 probes and trilinearly interpolate between them.

For regular surfaces, that means taking one irradiance sample per probe to get the pre-integrated result for a hemisphere around the given normal.

That is not the case for volumetrics.

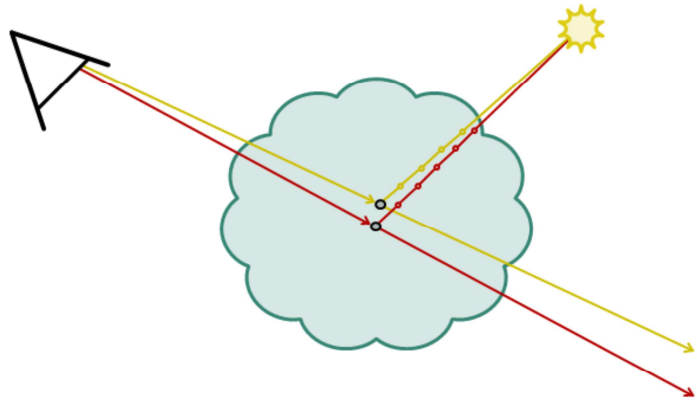
Volumetrics require a full integration of the incoming radiance over a sphere.

If we don't use an isotropic phase function, we have to do that at runtime.

Requirements 2: Volumetric Shadows



- Self-shadowing
 - Costly shadow rays for each sample
 - Divergence & high cache inefficiency
 - Duplicated samples



[Hillaire15] Towards Unified and Physically-Based Volumetric Lighting in Frostbite - <https://advances.realtimerendering.com/s2015/index.html>

© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

That brings us to the second set of lighting requirements – the volumetric shadows.

[CLICK]

I talked about part of the cost already, the costly shadow rays. They don't just scale badly; they are also not great for wave coherency and cache hit rate. We need to sample large parts of the volume for each individual ray during integration.

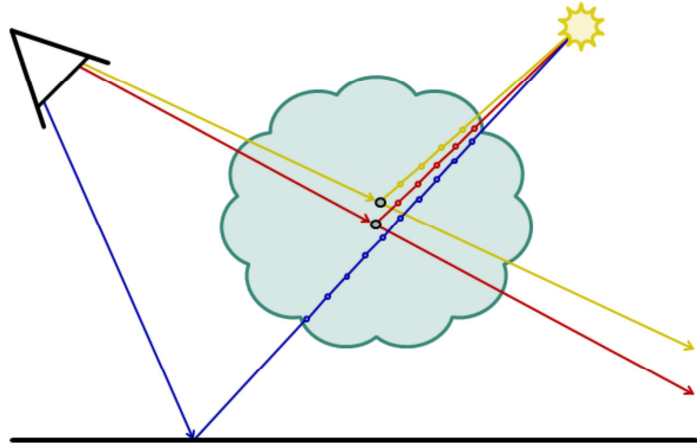
[CLICK]

If we add a second ray next to it, we can see another problem – massive duplication of samples. We constantly have to sample the same area of the volume over and over again.

Requirements 2: Volumetric Shadows



- Self-shadowing
 - Costly shadow rays for each sample
 - High divergence & cache inefficiency
 - Duplicated samples
- On the environment
 - Per pixel shadow rays

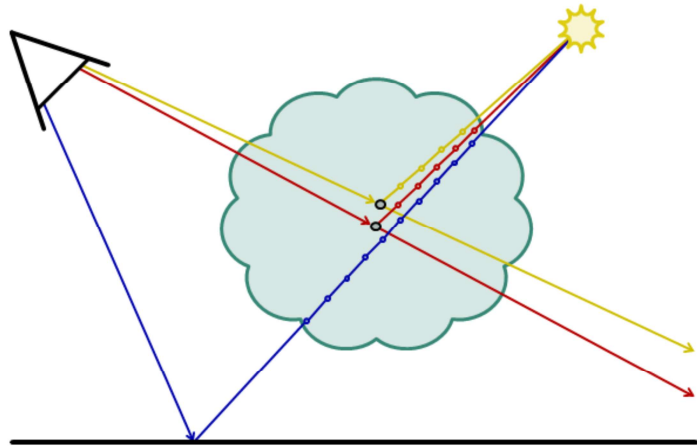


If we also want to see the shadows on the environment, we need to shoot an additional shadow ray from each surface to the light source, for each pixel in our scene.

Requirements 2: Volumetric Shadows



- Self-shadowing
 - Costly shadow rays for each sample
 - High divergence & cache inefficiency
 - Duplicated samples
- On the environment
 - Per pixel shadow rays
- For all lights
- No baked shadows



© 2026 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

And remember that we have to do that for each light if we want all of them to cast volumetric shadows.

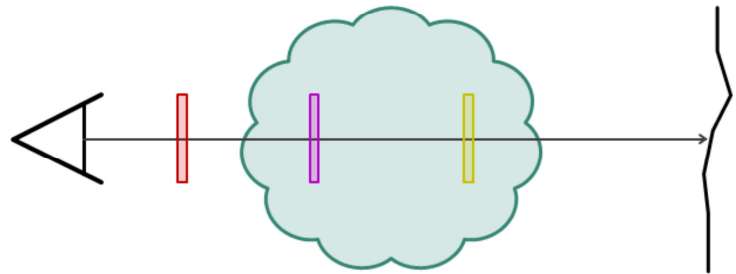
Clearly that amount of shadow rays isn't happening.

I also wanted all shadows to be real-time, having baked shadows is yet another limitation for artist workflows.

Requirements 3: Scene Integration



- Opaque
 - Per pixel accuracy
- Transparency
 - Circular dependency
 - Passes can't overlap



[Salvi13] Pixel Synchronization: Solving Old Graphics Problems With New Data Structures - <https://advances.realtimerendering.com/s2013/index.html>

© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

The next issue is scene integration.

[CLICK]

For opaque that is simple, we just stop the raymarch when we reach the pixel's depth. The only requirement here is that we have per pixel accuracy to avoid artefacts around depth discontinuities.

[CLICK]

Integration with transparency is a lot more complicated. It is easy to solve when the transparent surface doesn't overlap with the volume.

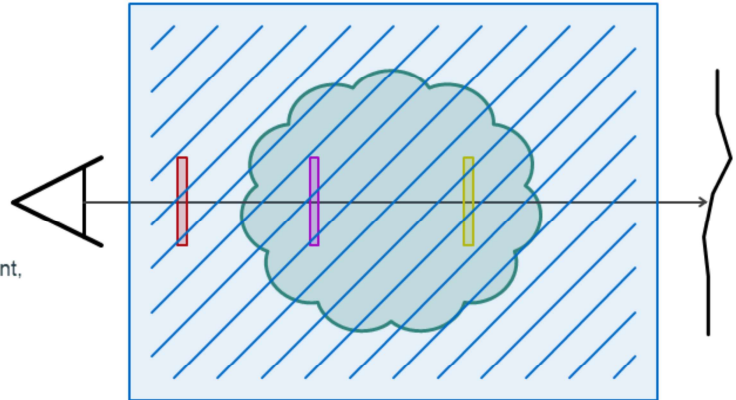
[CLICK]

But when they do, we need the volumetric effects to occlude the transparent surfaces, but also the other way around. We have a circular dependency here and we can't render both in the same pass in our rasterization-based pipeline.

Requirements 3: Scene Integration



- Opaque
 - Per pixel accuracy
- Transparency
 - Circular dependency
 - Passes can't overlap
- Volumetric Fog
 - Same issues as transparency
 - So far fog transmittance is applied to transparent, not the other way around



[Salvi13] Pixel Synchronization: Solving Old Graphics Problems With New Data Structures - <https://advances.realtimerendering.com/s2013/index.html>

If we take volumetric fog into account on top, we have a third system in that dependency chain.

Requirements 4: Volume Placement

SIGGRAPH



- Support overlapping volumes
 - Required for free volume placement
- Increased sampling cost from bounds check
 - View space/World space to object bounding box matrix multiplication
 - No precise bounds from OBB check, volume density rarely fills up the entire OBB
- Skip samples in front and behind of volumes
- Early out pixels with no overlap

© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

Overlapping volume support is another piece of the puzzle to allow artists to freely place volumes.

[CLICK]

In the cheapest of options we transform the sample position into object space and check if we are outside of the volume bounds. But even just those matrix multiplications add up quickly if we have to do them for each volume for each of our samples. It's also not very precise, the volumes are rarely boxes and we will get a lot of false positives.

[CLICK]

Ideally we would just like to skip the empty space around volumes entirely.

[CLICK]

For pixels that have no overlap, we don't even want to start the raymarch to begin with.

Requirements 5: Image Quality & Performance



- Current gen fidelity
 - Image clarity
 - Realistic art direction
- Great performance & quality scaling
 - RTX 5090 on the high end
 - Switch 2, Steam Deck, XBOX Series S, GTX 1660, RX 5700 on the low end
 - One system, no duplicated content authoring
- No temporal filtering
 - Fast moving effects, potentially close to camera
 - No motion vectors, would result in heavy ghosting

© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

The last set of requirements is around visuals and performance. On the visual side the system must meet current gen fidelity requirements. If the effects don't fit into the environment and can't support the more realistic art direction we have in some of our games, it's basically unusable for them.

[CLICK]

On the performance side, we need to scale all the way from a 5090 on the high end, to the lower end devices we want to support.

Switch 2, Steam Deck, Series S and on PC a GTX 1660 and a RX 5700.

The one thing we want to avoid at all costs is that we need a fallback system for lower end platforms and have to author content twice.

[CLICK]

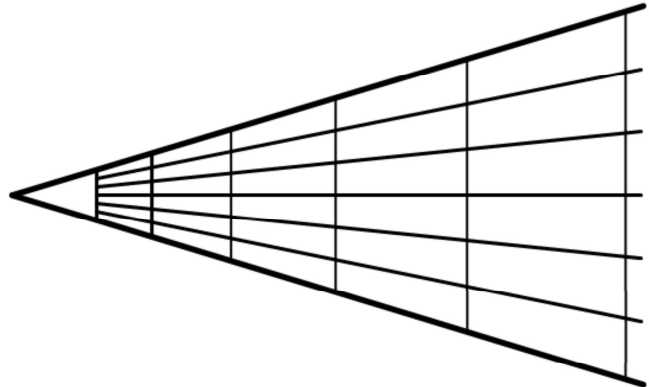
A lot of existing volumetric systems rely on temporal filtering to make that balance between quality and performance happen.

Unfortunately, when effects move quickly or are close to the camera this tends to result in heavy ghosting due to the lack of motion vectors.

Quite a long list of issues to solve we ended up with. Clearly, we had a bit of a challenge ahead of us.



- Frustum aligned voxels (froxiels)
- Volumetric integration that stores intermediate results
- Froxel grid
 - 8x8, 12x12, 16x16 pixels / froxel
 - 64, 96, 128 froxiels in depth
 - R16G16B16A16 Scattering + Transmittance
- Cubic filter for lookup

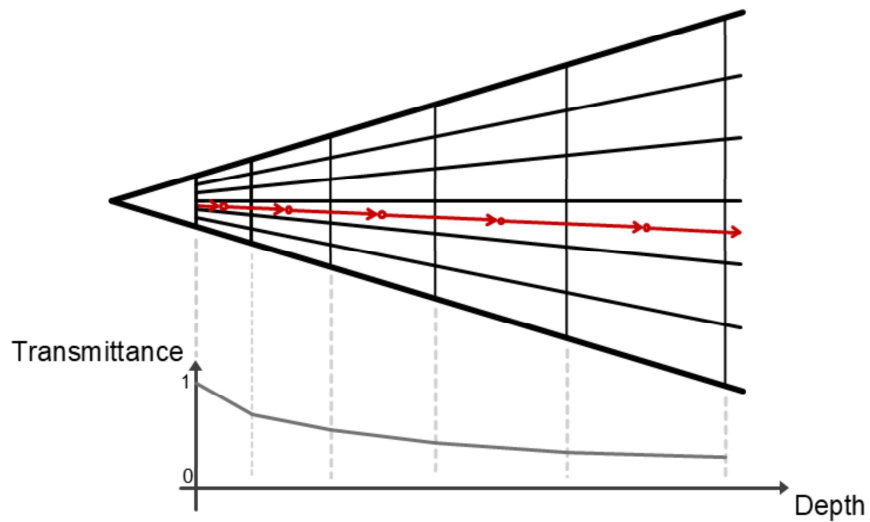


[Wronski14] Volumetric Fog: Unified compute shader based solution to atmospheric scattering - https://bartwronski.com/wp-content/uploads/2014/08/bwronski_volumetric_fog_siggraph2014.pdf

[Hillaire15] Physically-based & Unified Volumetric Rendering in Frostbite - <https://www.ea.com/news/physically-based-unified-volumetric-rendering-in-frostbite>

[Bauer19] Creating the Atmospheric World of *Red Dead Redemption 2*: A Complete and Integrated Solution - <https://advances.realtimerendering.com/s2019/index.htm>

There's two more systems I have to introduce before we go into the implementation. For volumetric fog we use the pretty much standard froxel grid based approach. I left some references here for details, but the essential part is simply that it is a volumetric front to back integration that stores intermediate results in a frustum aligned grid.



© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

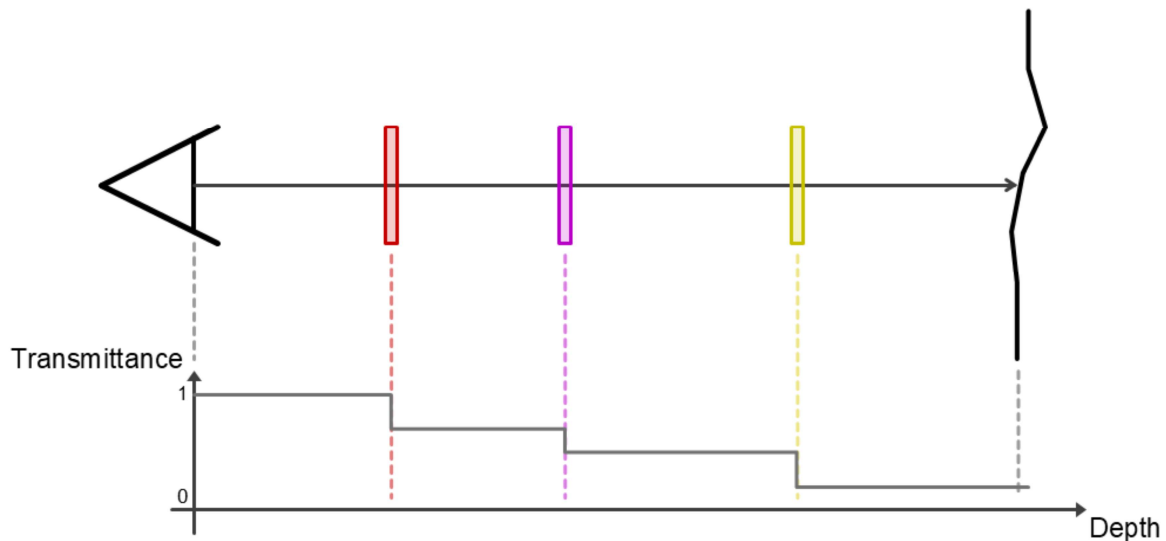
We can think of that grid as a very low resolution Raymarch, the underlying code is the same.

[CLICK]

We can map that to a transmittance function just as before

[CLICK]

, and we assemble that function piecewise as we move through the slices of the grid.



© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

For transparency, we traditionally render the surfaces back to front in order to integrate them properly within a single pass.

[CLICK]

Each surface's opacity is represented by an alpha value.

In this case too we can draw a transmittance function, similar to before.

[CLICK]

The sorting is per draw and not per pixel and that usually comes with very noticeable flickering artefacts.

That's where order-independent transparency, or OIT comes in.



- Order-Independent Transparency
 - Remove dependency on CPU side sorting
 - Create per pixel sorting on GPU side
 - Decouple transmittance integration from rendering

[Drobot25] Adaptive Voxel-Based Order-Independent Transparency - <https://advances.realtimerendering.com/s2025/index.html#avoit>

© 2026 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

All OIT algorithms in some form try to avoid this dependency on CPU side sorting. They usually either do that by creating a per pixel sorting on GPU side, or they do it by decoupling the transmittance integration from the actual surface rendering. If we have this transmittance function ahead of time, we can render the individual transparent surfaces in any order we like, we just have to sample and apply the right transmittance at that depth.

Check out Michal Drobot's presentation from last year for an overview of different techniques

Moment-Based Order-Independent Transparency



- Two pass rendering
 - Generate moments
 - Resolve transparency
- Transmittance function stored as series of power moments of the optical depth
- 0th moment with precise final optical depth stored for normalization and opaque composition
- Moments stored in logarithmic space
 - Can just be added up instead of multiplied
 - Fully differentiable, e.g. can be filtered
- Very storage intensive
 - 6 moments + 0th moment
 - 7 * 32b = 224b / pixel

Moment-Based Order-Independent Transparency

CEDRICK MUNSTERMANN, University of Bonn, Germany
 STEFAN KRUMPH, University of Bonn, Germany
 REINHARD KLEIN, University of Bonn, Germany
 CHRISTOPH PETERS, Karlsruhe Institute of Technology, Germany and University of Bonn, Germany

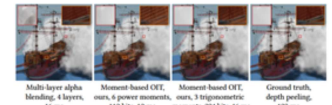


Fig. 1. Our novel techniques for order-independent transparency (OIT) use moment-based reconstructions of the transmittance. They can handle complicated participating media and transparent surfaces such as the ship simultaneously. We use the same approach to render shadows for the transparent shadow casters. The timings are full frame times for a resolution of 1024 on an NVIDIA GeForce GTX 1080 Ti rendering 40 million transparent fragments.

Compositing transparent surfaces rendered in an arbitrary order requires techniques for order-independent transparency. Each surface color needs to be multiplied by the appropriate transmittance to the eye to incorporate occlusion. Building upon moment shadow mapping, we present a moment-based method for compact storage and fast reconstruction of this depth-dependent function per pixel. We work with the logarithm of the transmittance such that the function may be accumulated additively rather than multiplicatively. Then an additive rendering pass for all transparent surfaces yields moments. Moment-based reconstruction algorithms provide approximations to the original function, which are used for compositing in a second additive pass. We utilize existing algorithms with four or six power moments and develop new algorithms using eight power moments or up to four trigonometric moments. The resulting techniques are completely order-independent, work well for participating media as well as transparent surfaces and come in many variants providing different tradeoffs. We also utilize the same approach for the closely related problem of compositing shadows for transparent surfaces.

Authors' addresses: Cedrick Munstermann, munstermann@bonn.de, University of Bonn, Endenicher Allee 14b, 53115, Bonn, Germany; Stefan Krump, krump@bonn.de, University of Bonn, Germany; Reinhard Klein, klein@bonn.de, University of Bonn, Germany; Christoph Peters, christoph.peters@kit.edu, Karlsruhe Institute of Technology, Am Fassberg 1, 76131, Karlsruhe, Germany; University of Bonn, Germany.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permission from permissions@acm.org.

© 2018 Copyright held by the owner(s). Publication rights licensed to ACM.
 2018-05-05, ACM, 10.1145/3203206
<https://doi.org/10.1145/3203206>

Proc. ACM Comput. Graph. Interact. Tech., Vol. 5, No. 1, Article 7. Publication date: May 2018.

[Munstermann, Krump, Klein, Peters18] Moment-Based Order-Independent Transparency - <https://dl.acm.org/doi/10.1145/3203206>

In our case, we use moment-based order-independent transparency, or MBOIT for short.

It stores the transmittance function as a series of power moments of the optical depth, as well as one precise 0th moment that contains the optical depth at the end of the ray.

Those moments are in logarithmic space. That turns the moment blending from a multiplicative problem into an additive one, which makes it extremely easy to generate this function on the GPU. Another nice property of the moments is that they are fully differentiable, which for example allows us to filter them freely.

The downside is that they are very storage intensive. We use 6 moments, with 32 bits each.

We tried running with 16 bit moments, but the artefacts we got were too noticeable.

We need a total of 224 bits per pixel for those moments, it's very heavy and it puts a lot of strain on memory bandwidth and caches.

Attempt 1 - Volumetric Particles

SIGGRAPH



- What if we handle volumetrics the same way we handle transparency?
 - Generate moments for a transmittance function first
 - Combine with transparent transmittance
- Decouple shadow rendering from main view
 - Generate transmittance function for shadows (volumetric shadow)
 - Resolve during main view lighting calculations



© 2026 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

That finally brings us to our first attempt.

The core idea was pretty simple, what if we apply the same OIT algorithm to volumetrics? If we generate the transmittance moments first, we can just add them to the transparent moments before resolving either of the systems, which would make the integration trivial

[CLICK]

, and you can see that in this very early test we did here.

[CLICK]

For the shadows, we knew that we needed some representation of a transmittance function there too so we can sample them at any depth for self shadowing and shadowing of the various surfaces.

That way we remove all of the duplication cost, we only render the shadow once in a very wave coherent way and sample that shadow where needed.

Attempt 1 - Volumetric Particles

SIGGRAPH



- Utilize GPU particle system as the base
 - Powerful system with lots of existing capabilities
 - Some tooling already exists and can be expanded upon
 - Existing fluid sim integration
- Needs a lot of particles to fill up the volume of individual effects
 - Very heavy to raymarch

© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

For density modelling, we decided to utilize our GPU particles.

One of our programmers, Asger Meldgaard, started building a real-time fluid simulation system for it and using that as a base would give us a lot of capabilities for free, so we decided to go with that.

But raymarching a large number of particles is not cheap, especially if we have to do it twice for the generate and resolve passes.

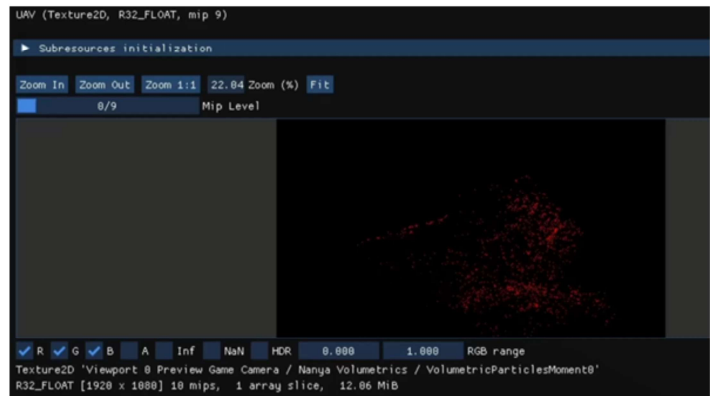
Some early tests with acceleration structures were not very encouraging, and it was clear to us that we need a different approach.

Attempt 1 - Volumetric Particles

SIGGRAPH



- Software rasterize them in compute shaders
 - 1 pixel / thread
 - Compute overlapping particles
 - Analytical computation of transmittance and scattering
 - Simple spheres with density falloff
- Moderately sized particles cover 100s of pixels
- Multi scale rendering to handle particle sizes
 - Each particle draws to 1 specific mip
 - Each particle draws roughly in the same amount of pixels



© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

We decided to try and software rasterize the particles in compute shaders.

Each thread would render a single pixel.

We start by computing which particles overlap that pixel, and we then analytically compute the transmittance for each of them and accumulate the moments.

We then could render the scattering in a second pass using the same logic, similar to OIT.

We just treated all particles as simple spheres with a density falloff.

[CLICK]

This didn't scale well enough.

Even moderately sized particles quickly covered 100s, if not 1000s of pixels, and we had hundreds of thousands of particles in a large effect.

[CLICK]

To solve this, I tried to render this system at multiple scales and draw each particle only to 1 specific mip depending on its size.

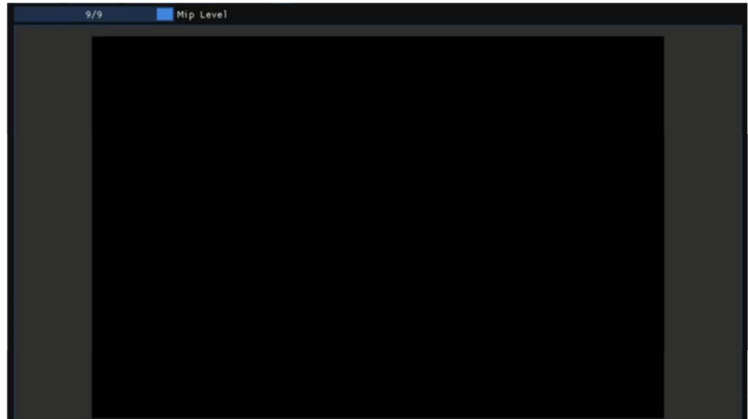
This way we roughly rendered the same number of pixels for each particle.

Attempt 1 - Volumetric Particles

SIGGRAPH



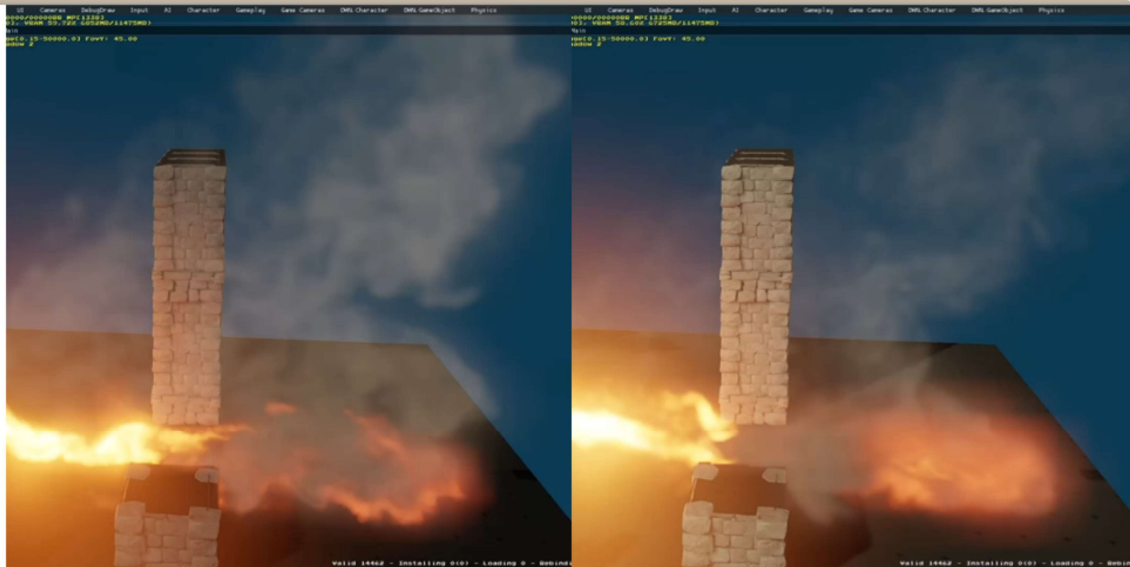
- Flatten mips
 - Sum up moments
 - Cubic filter to avoid box artefacts



After all the mips are rendered, we can flatten them by simply adding up the moments from each of them.

We had to use a cubic filter since linear filters left us with very noticeable artefacts.

Attempt 1 - Volumetric Particle Results



© 2026 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

These are the results of that tech at this stage. It's Glacier footage that isn't from 007 First Light, so I had to crop the video quite a lot. For the purpose of this presentation, I need you all to just pretend that this is footage from 007 and that Bond is standing offscreen with a flamethrower.

[CLICK]

You can see the realtime fluid sim at work and there was a lot of promise in this footage, utilizing our GPU particles allowed us to achieve complex effects early in testing, and we got features like collision with the ground for free.

But you can also see a couple of issues. The difference between these videos is the scale of the particles we used.

In the left one, we have quite a bit of sharpness, but you can see the dots of the individual particles.

We were already pushing the number of particles very high here, but since the effect is very large it was not enough.

In the right video we scaled the particles up, that way we do not see those dots but the effect is noticeably blurry.

Too blurry for what art direction was looking for.

Attempt 1 - Volumetric Particle Results



© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

Here's a close up of the same effect.

Also notice the depth testing artefacts around the edges of the foreground in the right video due to the multi scale rendering, it was an issue I never found a good solution for.

[CLICK]

And this was unfortunately a pattern we had throughout a lot of the development process of this system.

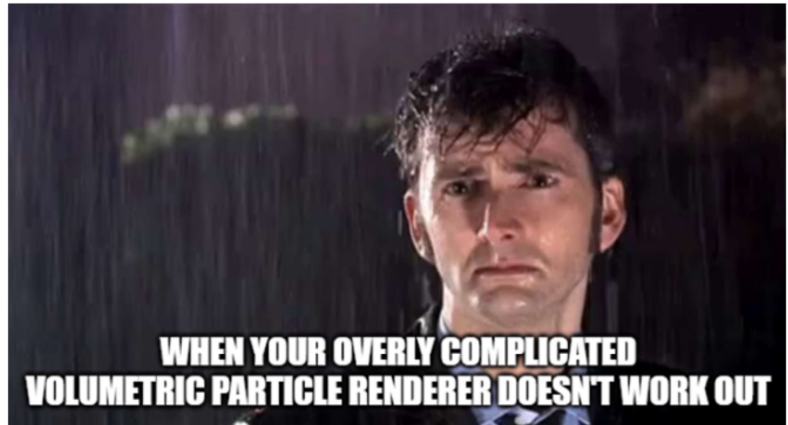
There was so much promise in it, but there was always a bunch of issues that were increasingly hard to fix.

Small issues required complex changes to the algorithm and that did not help with the performance.

Attempt 1 - Volumetric Particle Results



- Too complicated
- Questionable performance scaling
 - Moment accumulation too costly
 - Shadow rendering cost too high
- Visual quality not good enough
 - Artefacts
 - Blurriness <=> Dotted look
- Extremely hard to author
- Growing sense that tech doesn't live up to expectations



© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

We knew going into this that it might not work out, and the reality for this system was that we kept talking about the same issues without seeing a solution.

It was too complicated.

[CLICK]

The performance didn't scale the way we wanted, and especially the shadow rendering cost was way too high.

[CLICK]

The visual quality also wasn't where we wanted it to be with the various artefacts and tradeoffs.

[CLICK]

Most importantly it was just almost impossible to author effects.

[CLICK]

It also just plain sucked to see all the cool work Martin and Asger were doing and not being able to deliver on the render side of it, and I could feel that they were increasingly disappointed with the results we got.

Those last two things especially made it clear to me that I had to reconsider the approach, they broke the one golden rule I had. The workflow was miserable.

[CLICK]

Around that time I started to consider if we needed a fundamentally different approach.



- Not having a sampling based renderer is too limiting for artists
- Back to raymarching?
 - Having two raymarch passes is too expensive
 - Overlapping volumes need to be handled
 - Secondary rays are too expensive
 - Performance scaling & Quality
 - Can the volumetric shadows deliver the quality we want
 - Does the performance scale the way we want
- The largest issue was doubt
 - Already spend a lot of time on the first attempt and it didn't work
 - Don't want to throw that away just to fail attempt 2 as well

I went back to the drawing board with the learnings from the volumetric particles. The most important one was that not having a sampling-based renderer was too limiting for artists.

[CLICK]

So, I took another look at raymarching and why I discarded it originally.

[CLICK]

The first reason was that having two raymarch passes for the moment generation and scattering resolve was too expensive, but without those the transparent and fog integration still needed to be solved.

[CLICK]

The second one was that overlapping volumes had to be handled in a way that was efficient.

[CLICK]

The third issue we had originally was already solved more or less, it's the cost of the secondary rays for the shadows. In theory we had a solution for it, volumetric shadow maps.

[CLICK]

I had ideas to solve the other problems, but the core question was if we can get the quality to where we needed it to be, and if we could do so at the performance level we were aiming for.

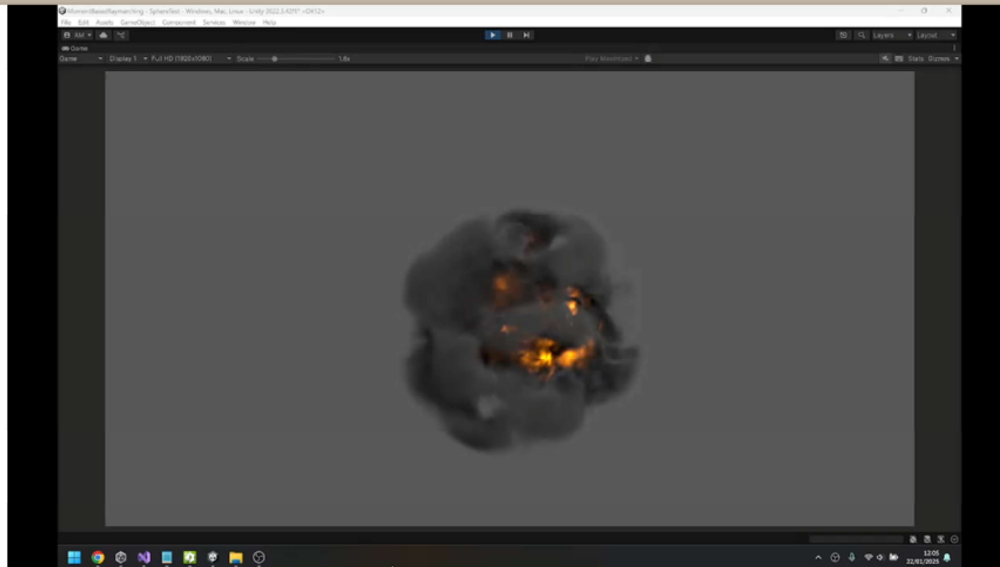
[CLICK]

But by far the largest issue I had at the time was that I had a lot of doubt. I already spent 2 ½ months on the first attempt, and it didn't work out, but maybe I could still solve the issues I had with it. What I knew is that I definitely didn't want to be the guy that throws

that work away just to fail a second attempt as well.
I needed some certainty.

Attempt 2 - Smolder

SIGGRAPH

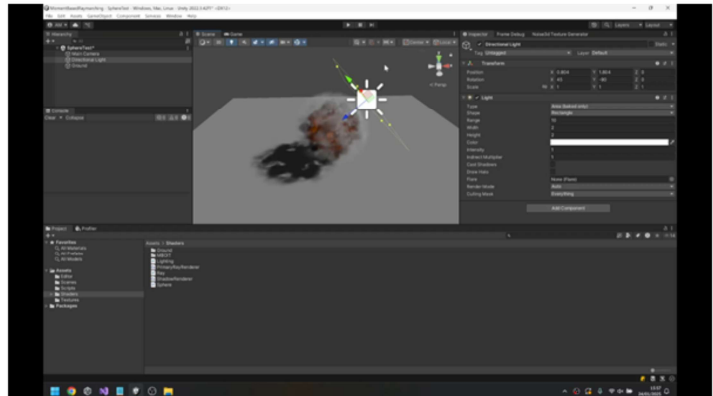


© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

Luckily for me, I went on a skiing trip with my family the week after. And while my parents and sister went to bed at 10pm, I grabbed my laptop and coded a prototype every evening for a few hours. I only had Unity on that, but all I needed to code was a couple of compute shaders, and Unity did just fine.



- [Hillaire15] Volumetric Shadow Cascades
 - 3D grid based
 - Limited precision $\leftrightarrow$ High memory footprint
- [Salvi10] Adaptive Volumetric Shadow Maps (AVSM)
 - High memory footprint
 - Costly to compute (requires sorting)
- [Hillaire20] Beer Shadow Maps (BSM)
 - Very cheap
 - Low memory footprint
 - Assumed uniform density



[Hillaire15] Towards Unified and Physically-Based Volumetric Lighting in Frostbite - <https://advances.realtimerendering.com/s2015/index.html>

[Salvi10] Adaptive Volumetric Shadow Maps - <https://www.intel.com/content/dam/develop/external/us/en/documents/salvi-avsm-egsr2010-564567.pdf>

[Hillaire20] Physically Based and Scalable Atmospheres in Unreal Engine - https://blog.selfshadow.com/publications/s2020-shading-course/hillaire/s2020_pbs_hillaire_slides.pdf

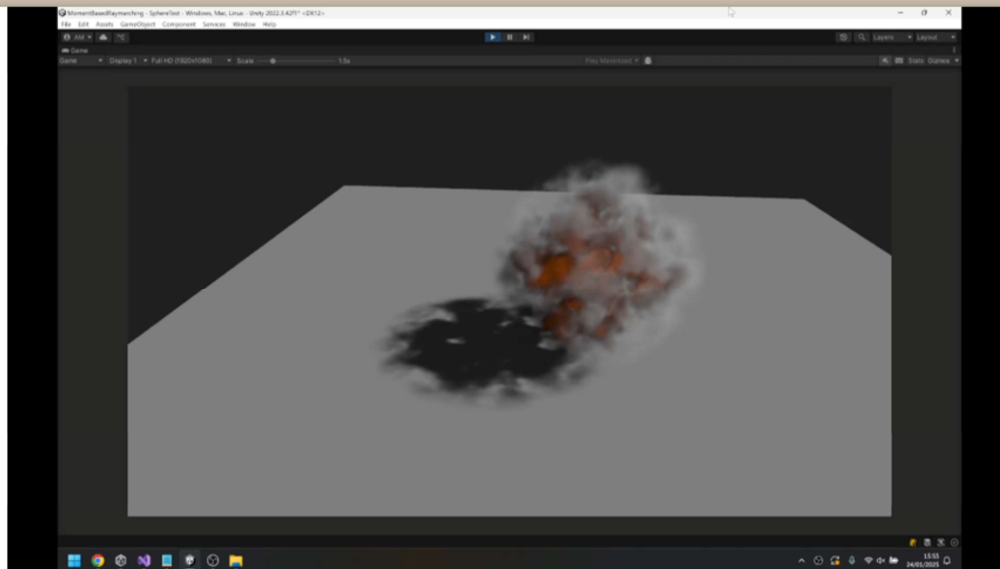
I coded a simple raymarcher through a procedural sphere and wanted to add a single shadow to it so I can judge the quality and performance scaling. I looked at some solutions that were out there, but I discarded a lot of them early on. The volumetric shadow cascades because they wouldn't give me the quality I wanted, the adaptive volumetric shadow maps because they sounded way too costly in terms of memory and performance.

The most promising ones I found were Hillaire's Beer Shadow Maps. They are very cheap and don't need a lot of memory. I experimented with them, but they assume uniform density of the volume, and the resulting quality of the self shadowing wasn't good enough.

Ultimately I decided to stick with MBOIT style moments for the shadows. They gave me a good mix of quality and cost, and they came with the great properties I mentioned before. The quality and performance issues I had with them in the volumetric particle case were because of the multi-scale rendering, but the tech itself actually worked really well.

Attempt 2 - Smolder

SIGGRAPH



© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

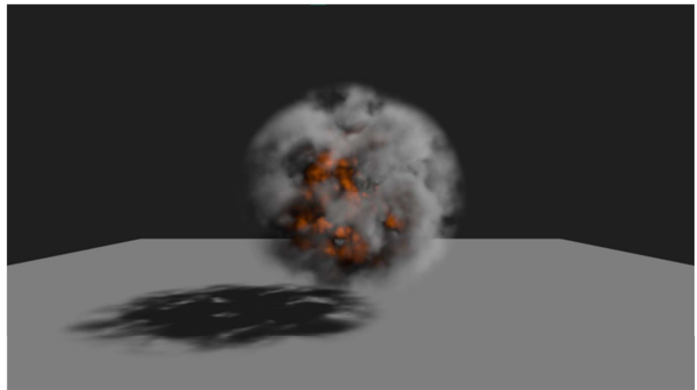
I also realized you can get away with a fairly low resolution and large step sizes when rendering those shadows due to the already blurry nature of them. The results I got from this test were great.

Smolder Early Prototype Results

SIGGRAPH



- Per-pixel visual quality
- Better lighting/shadow quality
- Way better performance scaling
 - 20-30 μ s per shadow texture (3050Ti Laptop)
 - 0.7ms @ 1080p (3050Ti Laptop)
 - 10 μ s per shadow texture (7900 XTX)
 - 90 μ s @ 1080p (7900 XTX)



© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

Not only did I get the visual quality I was looking for, the performance news was just as good.

Even on my 3050Ti powered laptop rendering the shadows was dirt cheap, and the main view performance was also great.

Especially since I knew that there's a lot of optimization opportunities down the line.

This little prototype already was a massive leap over the previous tech, and it gave me the confidence to come back from vacation and make the call to leave our existing tech behind the Monday after that.

Smolder Early Prototype Results

SIGGRAPH



But most importantly - Tech Art approved!

Martin Bjarke Kragh Nielsen 1/24/2025 3:53 PM



I will never be able to unsee the shadow

Martin Bjarke Kragh Nielsen 1/24/2025 3:56 PM



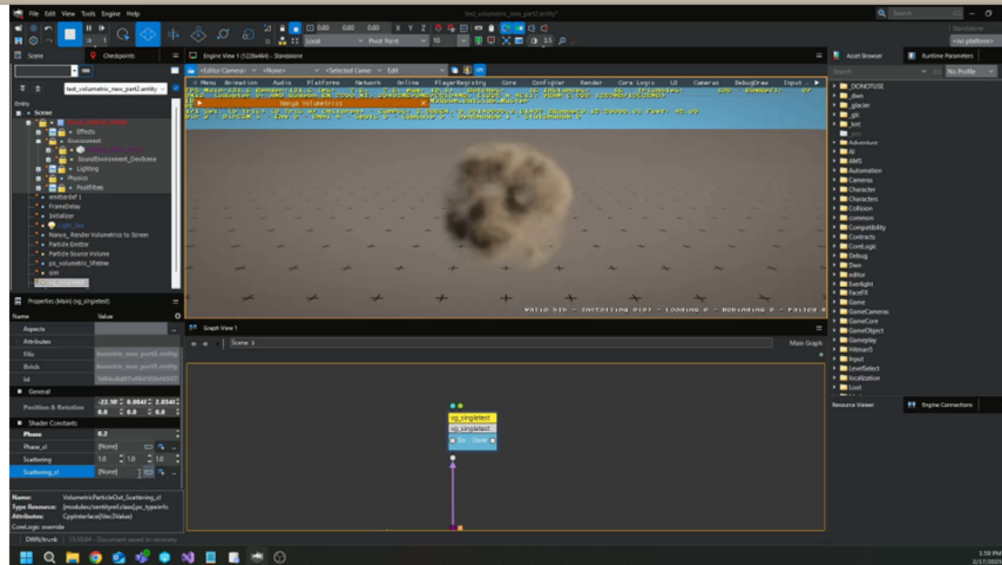
but where are the dots?

© 2026 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

Most importantly, it got Martin hyped. It was some energy we desperately needed after being a bit stuck on the previous attempt.

Smolder Early Glacier Prototype

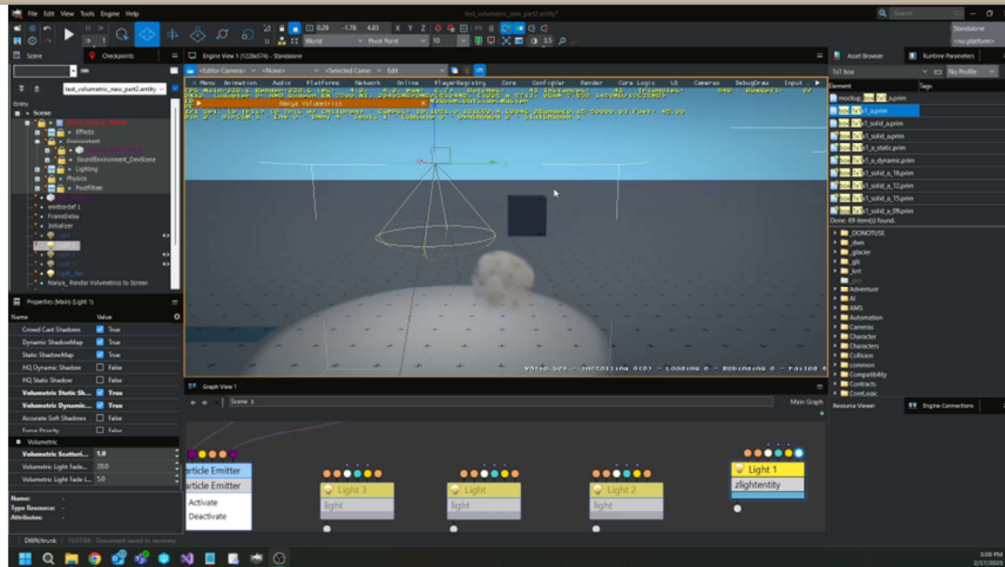
SIGGRAPH



© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

With that decided, Smolder started taking shape in Glacier. I started by bringing over the sphere and got the basic setup up and running...

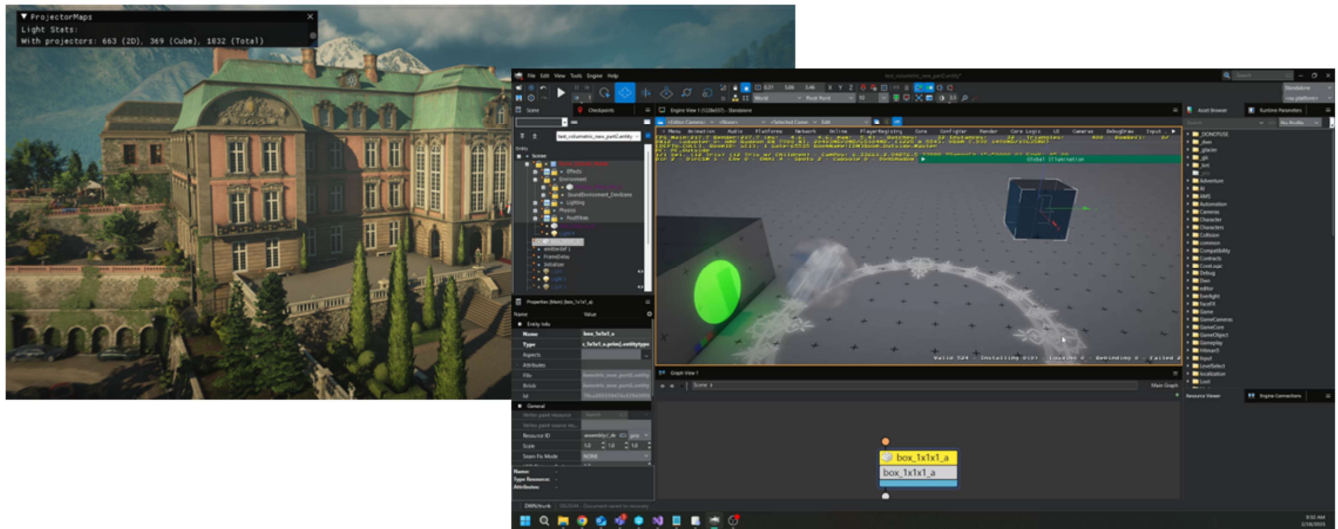
Smolder Early Glacier Prototype - Dynamic Shadows



© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

...followed by a basic light loop and shadow sampling.

Smolder Early Glacier Prototype - Projectors



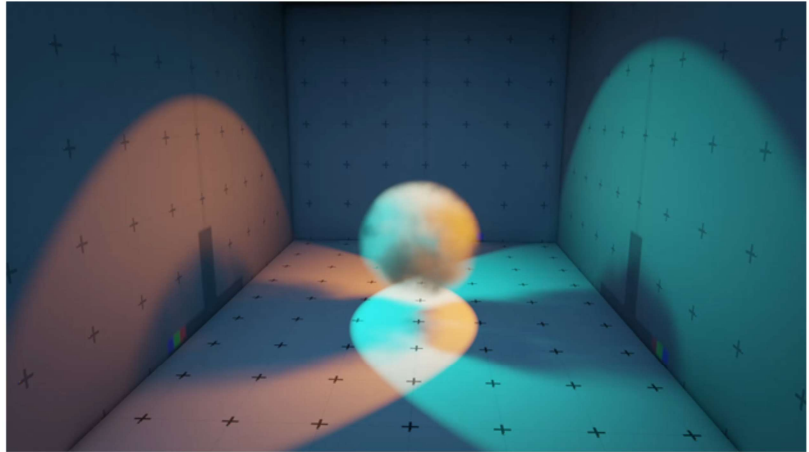
© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

I also hooked up a bunch of other lighting features, like projector maps. Our artists on 007 First Light love using projector maps, and this turned out to be a crucial feature to support. For example in this specific hotel area in the game we have over 1500 lights total, and over 1000 of them use projector textures.

Smolder Early Glacier Prototype - Spot Lights



- Moment Shadows
 - 0th Moment + 4 Power Moments
 - 16 bits/moment
 - 1 R16 + 1 R16G16B16A16 Texture
 - 80 bits/texel
 - 64x64 – 1024x1024 resolution



© 2026 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

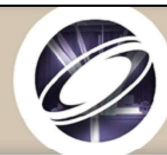
Here's an early test for volumetric spot light shadows.

We ended up sticking with 4 moments and 16 bit quantization. That results in a memory footprint of 80 bits/texel.

Not crazy cheap, but we can get away with a lot when it comes to the resolutions of the shadows to make up for it.

These ones here are 128x128, but we support resolutions from 64x64 on the low end all the way up to 1k.

Smolder Early Glacier Prototype – Transparent Shadow Sampling



- Sampling Cost
 - Two step sampling
 - 0th moment sample first for early out
 - Simple hardware bilinear sample
 - Differentiable moments
 - No costly PCF lookup
 - ~30% of the sampling cost of regular shadows



© 2026 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

Here's an example of sampling those shadows on transparent surfaces and you can see the shadow accumulating over the depth of the volume. Sorting isn't working just yet.

At this point you might be wondering about sampling cost of those shadows. I showed early numbers to render the shadows and the primary rays, but we need to consider the cost we add to the opaque and transparent lighting as well.

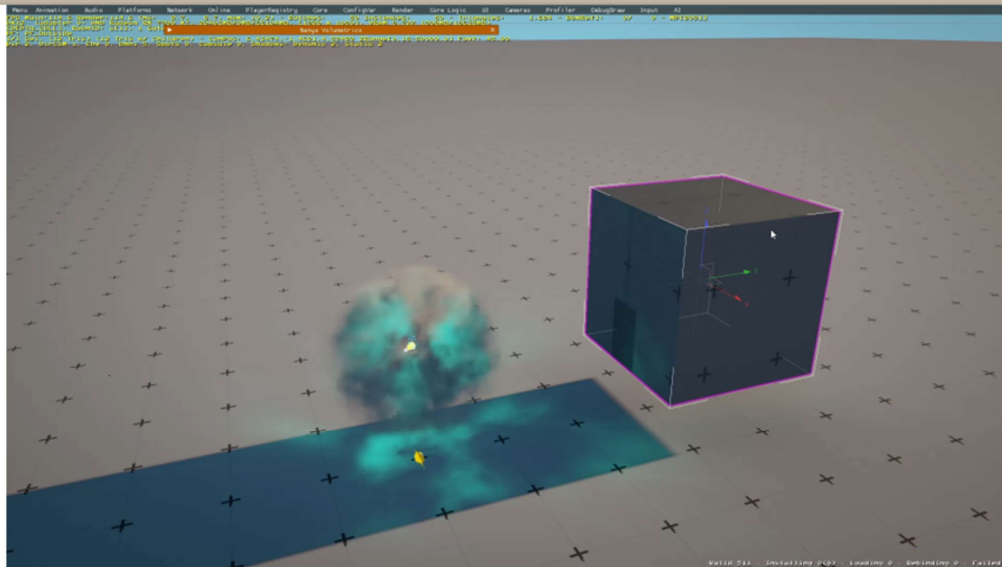
Here's where the moment-based shadows give us another good opportunity. If you remember we need a 0th moment that contains the optical depth for the end of the ray.

We can sample that first and know based on the value if we are inside the shadow at all, which allows us to skip the sample of the other moments and the evaluation of the transmittance function.

Also, we aren't storing depth in the volumetric shadows, and we don't need any costly PCF filters. A single bilinear sample works just fine.

In 007 the sampling cost of the volumetric shadow is roughly 30% of the regular shadow sampling cost for lights that have both types of shadows.

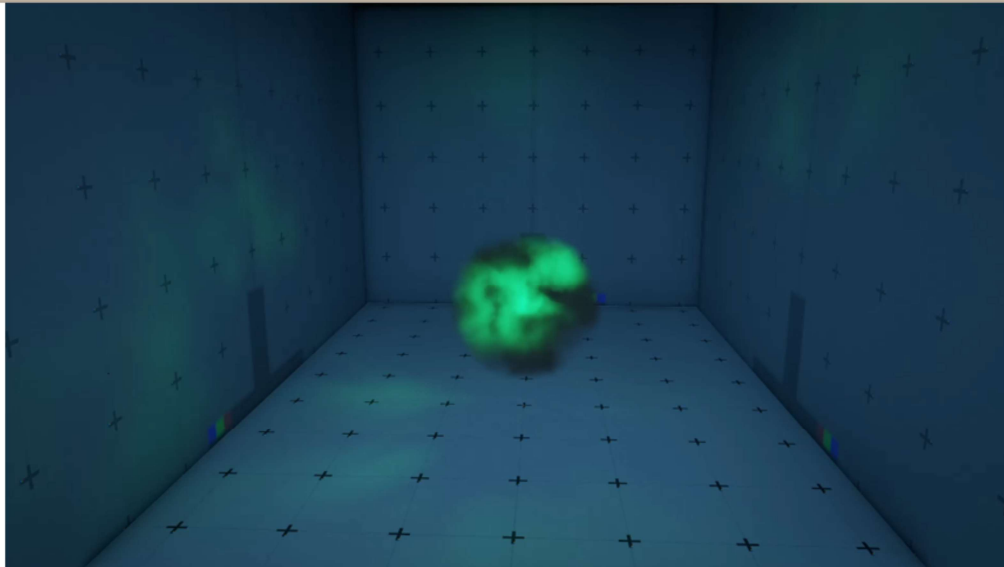
Smolder Early Glacier Prototype - Depth Testing



© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

Since we no longer had the multi-scale approach, depth testing was also working as expected.

Smolder Early Glacier Prototype - Omni Lights

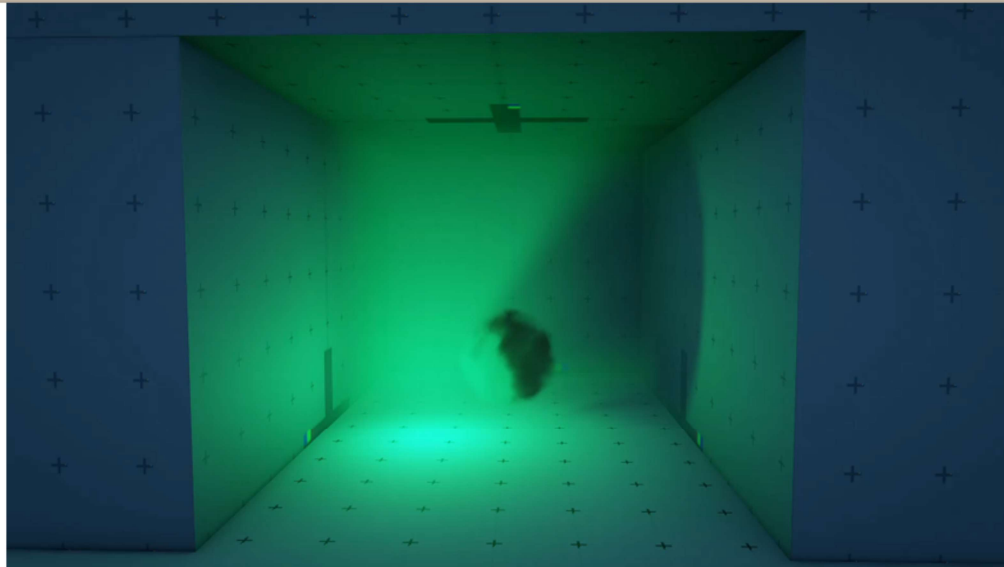


© 2026 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

Omni light shadows are usually a massive pain in rasterized pipelines, but in the raymarched shadow case we are not limited by a fixed projection. We can just render them into an octahedral texture in a single pass.

This is where the visuals started to inspire a little bit, where we could create some visual effects that you usually don't see in games, even though we still only had the procedural sphere.

Smolder Early Glacier Prototype - Volumetric Shadows



© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

I also added sampling support of the shadows to the volumetric fog.

This is also where we started to show the tech to more people in the engine team, and there was a lot of excitement building up around the tech. It also helped a lot that it took only a few hours to add new features at this point, and progress was great.

Smolder Early Glacier Prototype - Volumetric Shadows



© 2026 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

Here's a stress test of the system a bit with some higher frequency shadow detail.

The 007 First Light Smoke Gadget

SIGGRAPH



- Existing solution had issues with
 - Visuals
 - Performance
 - Gameplay readability



© 2026 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

This early Smolder work happened in February of 2025, and at this point we did not plan to use this tech in 007 at all.
But then came the smoke gadget.

The version we had had issues with visuals and performance. Most importantly, our gameplay director did not like that the shape of the volume was hard to read. We use that shape for AI line-of-sight calculations, so having a clear visual representation was extremely important for the stealth gameplay.

With the early results we saw from Smolder, the discussion quickly came up if we can use that for the smoke gadget instead.



- Timeline
 - March 2025: Request for better volumetrics tech
 - June 2025: Reveal trailer
 - September 2025: Full State of Play gameplay reveal
 - March 2026: Release (moved to May 2026 later)
- Smolder
 - Just a floating volumetric sphere
 - Still a bunch of unknowns
 - Massive scope and risk in developing real-time fluid simulation
 - Can't afford to make it my full-time project :(
- Limit scope, utilize prebaked volumetric simulations and only use the tech for the smoke gadget

At this point it was early March, and 007 First Light was in full production. Our reveal trailer was scheduled for 3 months later, and the full gameplay reveal at Sony's State of Play was supposed to happen another 3 months after that. The release was a year out.

[CLICK]

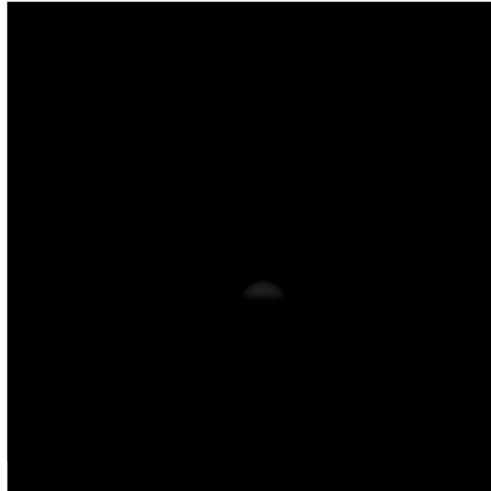
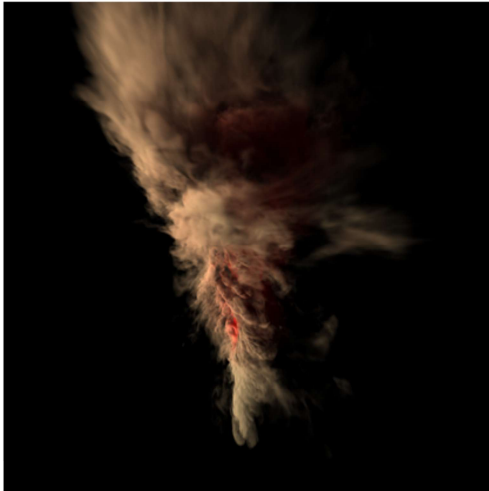
And Smolder, while making good progress, was far from ready. It was still just a floating sphere, there were a lot of unknowns, and getting the real-time fluid sim ready and shippable was another massive issue.

We also just couldn't afford to make Smolder my full time project. There was a lot of other work on my todo list.

[CLICK]

Ultimately I had to make a call, and that was to commit to building Smolder for that smoke gadget, but not to build it on top of the fluid simulation tech.

Instead, we would run the simulation offline, and use prebaked density data.



For the density data, it was pretty clear that our best option is to utilize the OpenVDB format. I experimented with it a while back in a small offline renderer project, and I'm a pretty big fan of it. It comes with a very mature c++ library, and is pretty battle tested at this point.



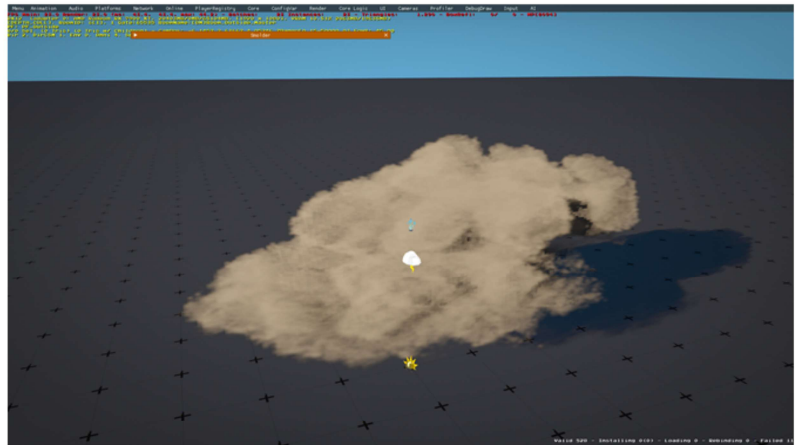
<https://www.openvdb.org/>

But as usual most importantly – it supports a great workflow. It is widely supported, and artists can use whatever tool they want. A lot of the modelling and procedural tools they are familiar with support it.

Most of our volumes are built in EmberGen and Houdini.



- Very easy to integrate (~half a day)
- Very wide feature set
- Small memory footprint (fp4 compression)
- High register usage
- Couldn't get performance to work out



<https://developer.nvidia.com/nanovdb>

https://media.disneyanimation.com/uploads/production/data_set_asset/1/asset/Cloud_Readme.pdf

OpenVDB isn't really usable on GPU side though, we needed a runtime format. The most obvious one is NanoVDB – I didn't have any experience with it but seemed easy enough to set up to give it a try.

Also shoutout to Disney for releasing the cloud dataset, and especially for adding a 1/16th scale version of it 😊

NanoVDB is essentially just a flattened version of the sparse voxel tree in OpenVDB. It has some nice runtime features on top, like fp4.

[CLICK]

But unfortunately, I couldn't get the performance to work out, mostly due to high register pressure.

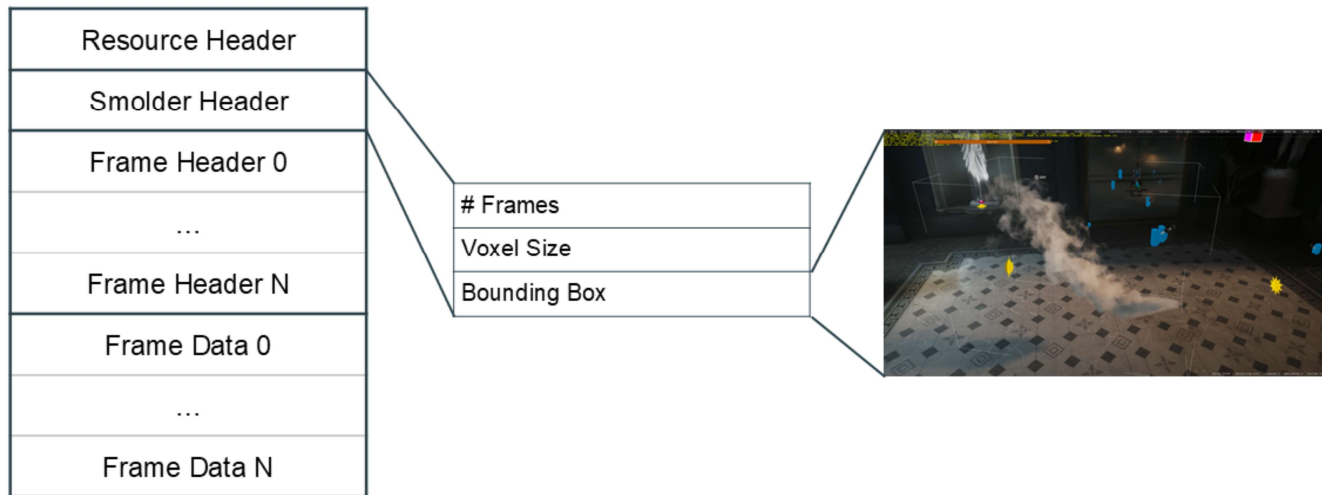
I decided that a custom runtime format would be the more optimal choice. It added development time but ultimately saved us a lot of performance and gave us a lot more flexibility.



Resource Header
Smolder Header
Frame Header 0
...
Frame Header N
Frame Data 0
...
Frame Data N

The format itself is not too complicated. The resource header contains generic information about the resource and is not Smolder specific.

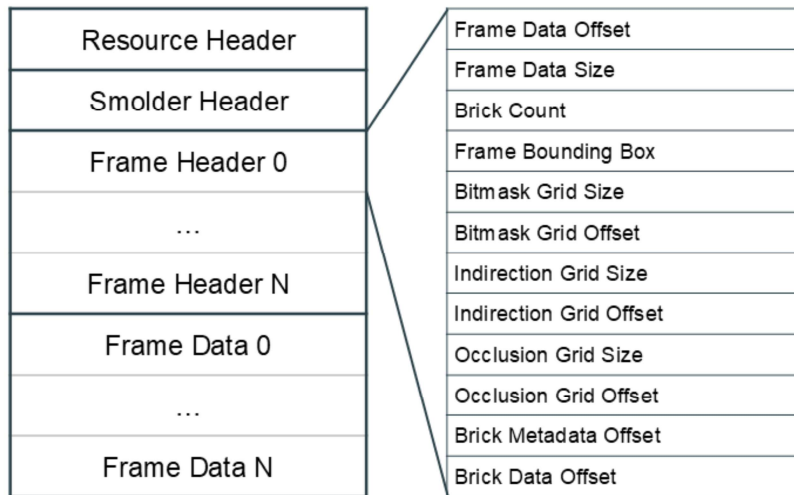
Custom Runtime Format - Smolder Header



© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

Next is the Smolder header. It contains the number of frames in the animation, the size of the voxels and a bounding box for the entire animation, which is only used to show the bounds in the editor, not for the actual rendering.

Custom Runtime Format - Frame Header



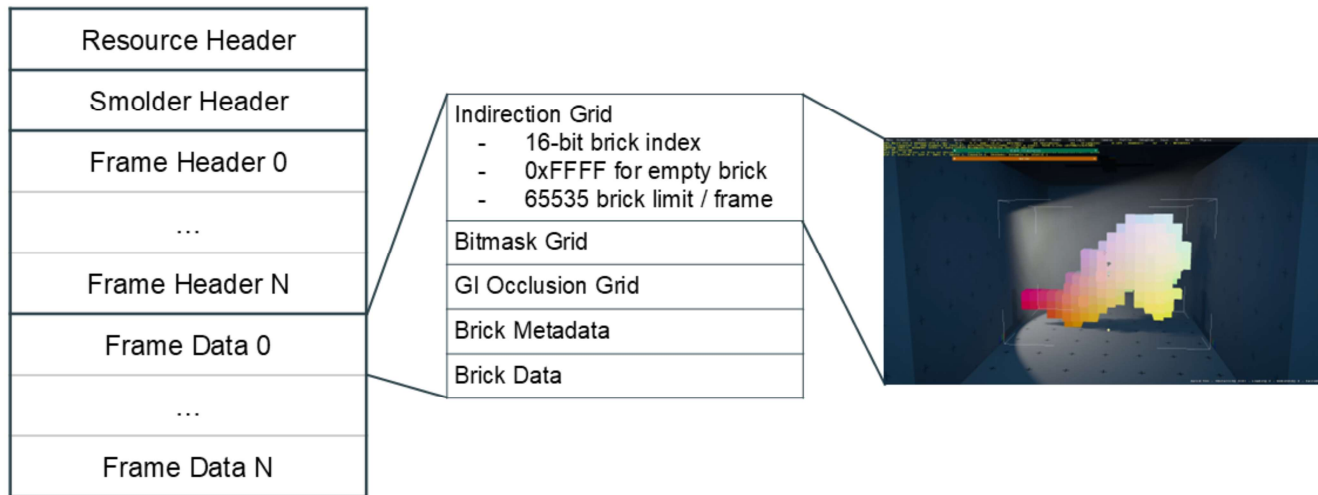
© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

It's followed by a list of frame headers and then a list of frame data.

The header is basically just describing where each piece of data can be found in the data blob.

It also contains a tight bounding box for that specific frame, which is used for the actual bounds checks in the renderer.

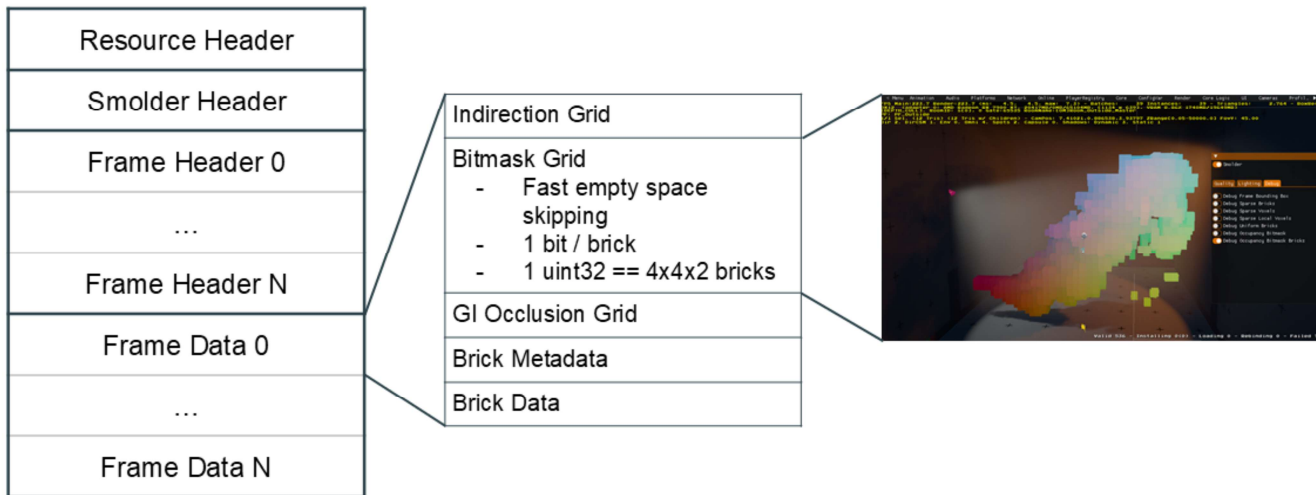
Custom Runtime Format - Indirection Grid



© 2026 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

Let's get to the actual interesting part, the data. The data is stored in a one level, sparse grid of bricks, with each brick being 4x4x4 voxels. The indirection index is 16 bit. I thought that's more than enough, but artists ran into that limit on a few occasions. We might have to increase it to 32 bits going forward.

Custom Runtime Format - Bitmask Grid



[Amanatides, Woo87] A Fast Voxel Traversal Algorithm for Ray Tracing - <http://www.cse.yorku.ca/~amana/research/grid.pdf>

© 2026 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

Next, we have a bitmask for the bricks that we use for empty space skipping. Each brick is a single bit in this grid and we store it in groups of 4x4x2 bricks.

We use the fast voxel traversal algorithm for that, which at this point is almost 40 years old. Still works great though.

We walk through those brick groups and can simply check for 0 in order to skip the entire group.

This bitmask speeds up the culling step noticeably compared to walking through the indirection indices.

Custom Runtime Format - GI Occlusion Grid



Resource Header
Smolder Header
Frame Header 0
...
Frame Header N
Frame Data 0
...
Frame Data N

Bitmask Grid
Indirection Grid
GI Occlusion Grid - 1 byte occlusion value per 8x8x8 bricks - Generated offline from uniform random samples
Brick Metadata
Brick Data



© 2026 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

Next, we have a GI occlusion grid, we added that to the format fairly late. It contains extremely low res prebaked occlusion, and we just generate that data from uniform random samples around the block of bricks. The reason why we need it is that once we added GI sampling, we got noticeably flat volumes, the screenshots are with occlusion on and off.

Custom Runtime Format - GI Occlusion Grid



Resource Header
Smolder Header
Frame Header 0
...
Frame Header N
Frame Data 0
...
Frame Data N

Bitmask Grid
Indirection Grid
GI Occlusion Grid - 1 byte occlusion value per 8x8x8 bricks - Generated offline from uniform random samples
Brick Metadata
Brick Data

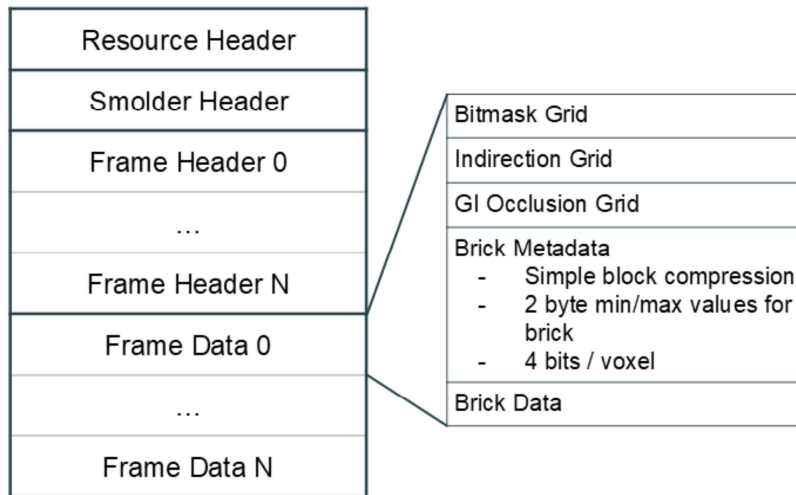


© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

Here's a GI only view.

Even at this low resolution it gives us the darkening inside of the volume that we are looking for, and it costs us very little.

Custom Runtime Format - GI Occlusion Grid



© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

Finally, we have the brick data. Similar to the fp4 compression in NanoVDB, I added a very simple block compression to the bricks in the custom format as well. The metadata contains a min and a max value for each brick.

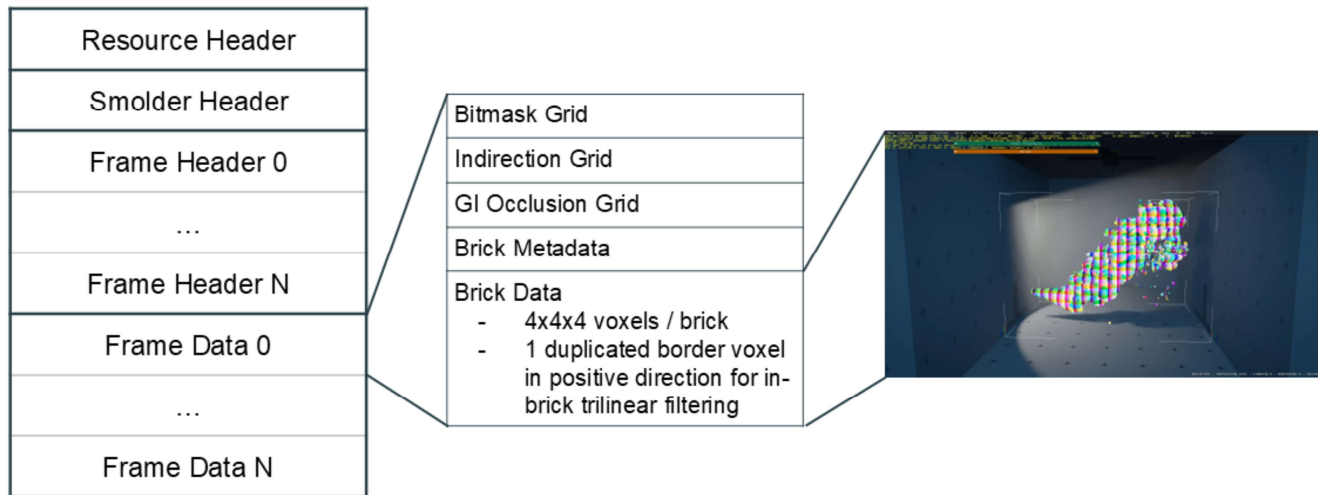


We ended up using 4 bits per voxel, it resulted in basically no quality loss for the density data.

I also tried using 2 bits, but as you can see here, we started to lose some details in the shadows and the quality got noticeably worse.

It's not unusable though, maybe it can be a scaling option in the future.

Custom Runtime Format - GI Occlusion Grid

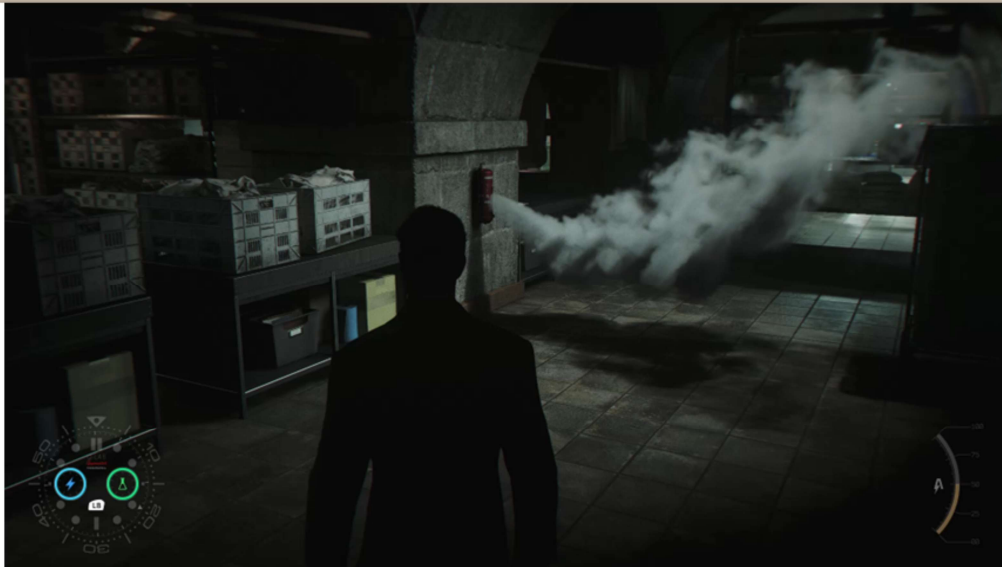


© 2026 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

Lastly we have the brick data. They are actually 5x5x5 voxels in size in the data since we duplicate voxels to allow for trilinear interpolation within the same brick at runtime. In return the bricks need a lot more memory, but the tradeoff is worth it.

Hello Smolder in 007 First Light

SIGGRAPH



© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

With the runtime format sorted, we had what we needed to get the artists started on building content. I also created this first test of Smolder in a 007 First Light environment and posted it in the game's teams chat.

It got a lot of artists very excited about the tech, and it's where Smolder essentially stopped being a smoke gadget tech.

The day after that I was in a meeting with our tech art director who asked me about the feasibility of using Smolder for volumetric clouds in our gameplay trailer.

Early 007 First Light Tech Art Tests

SIGGRAPH



Per Magnusson 6/25/2025 3:45 PM
Gotta show another test, took out a VDB with higher resolution (probably way too high for us to use in the game), but damn how the shadows fall across it and all it looks so damn good



© 2026 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

At this point more tech artists started experimenting with Smolder, with Per specifically doing a lot of testing and getting some great results.

While I was extremely worried about Smolder being used for an increasing number of effects and the work that would cause, it was also really nice to see it being used for cases we never planned for and never tested.

And it was handling a lot of them pretty well.

The very close collaboration and quick turnaround times between me and tech art kept building up a lot of positive momentum in the group.

Early 007 First Light Tech Art Tests

SIGGRAPH



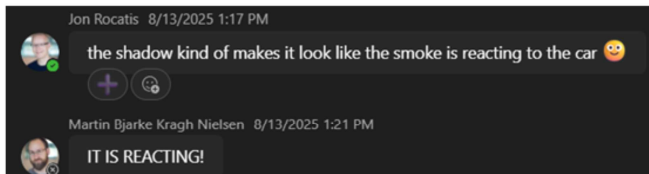
Per Magnusson 7/2/2025 11:06 AM
Drifting smoke testing



© 2026 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

It was also interesting to see how the artists kept pushing the quality of the simulations as they got more used to the tools and workflows.

Early 007 First Light Tech Art Tests



© 2026 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

We also started seeing some cases where people didn't quite expect the tech to produce some of the outputs we got.

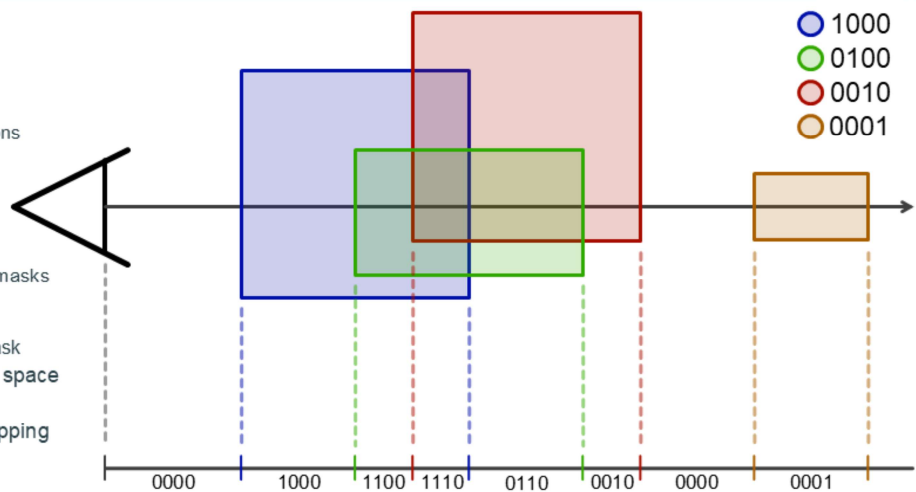
[CLICK]

Please excuse Martin's driving in this video 😊

Overlapping Volumes & Segments



- Wave based
 - 8x8 or 8x4 pixels
 - Dedicated shader permutations
- Segment based culling
 - 2 segments per volume
 - Up to 4 volumes per wave
- Output per wave
 - 1 uint32: 8x4 bit segment bitmasks
 - 8 float16: Segment depths
 - 1 uint32: Wave volume bitmask
- Occupancy bitmask based empty space skipping
- No inner volume empty space skipping



[Malan25] How to Decimate your textures - BCn compression tricks in Horizon Forbidden West - <https://graphicsprogrammingconference.com/archive/2025/#how-to-decimate-your-textures-bcn-compression-tricks-in-horizon-forbidden-west>

© 2026 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

But while all of those experiments were happening, I still had a bunch of problems to figure out. And with more effects getting built, the pressure increased to make the rest of the system work.

The first issue to solve was how to handle overlapping volumes.

To avoid per sample checks, I wanted to calculate volume overlaps ahead of time.

Since Smolder is built around the threadgroup coherency, we can do this calculation on a per group basis. A threadgroup in Smolder is either 8x4 or 8x8 pixels, depending on hardware support for Wave64. Smolder makes heavy use of wave intrinsics, and having two different versions of the shader for the different wave sizes was a great optimization.

I ended up following an idea that can be found in offline rendering and split the ray into segments. Each overlapping volume will add two segments to the total.

[CLICK]

We can skip the last segment, that one will never overlap anything.

For 007 I limited the overall amount of overlapping volumes to 4, but it can be more if needed.

We can then sort all of those intersection distances to get the bounds of our segments, and we can use the knowledge of which volume creates which bound to generate a bitmask for each segment.

[CLICK]

For the first two segments that would give us these two bitmasks for example.

[CLICK]

And we can just do that for all the other segments.

[CLICK]

For the segments, we have to store 8 depth values, and 16 bit was enough for those.

We also need to store a mask that tells us which of the volumes we have around are the 4 volumes for this wave.

[CLICK]

We use the occupancy bitmask data to tighten the bounds, that makes those segments really accurate.

[CLICK]

It doesn't account for any holes inside of the volumes, that might be something that can be improved in the future.

[CLICK]

I'm particularly interested in testing the BCN compressed volume data that Hugh Malan presented at GPC last year, that seems like a very good fit for the prebaked volume data.

Overlapping Volumes & Segments

SIGGRAPH



- How do we sort them?
- Cutesort!
 - WaveActiveBallot based rank sort
 - Track smaller and equal value mask per lane
 - Loop through bits lowest to highest
 - Higher bits override lower ones
 - Lane can compute sorted position based on mask



<https://gist.github.com/dondragmer/0c0b3eed0f7c30f7391deb11121a5aa1>

I left one part of the culling out, how do we sort the intersection distances?

The answer is cutesort! One of my favourite algorithms out there, it's essentially just a rank sort algorithm that utilizes WaveActiveBallot, so it can sort values within a wave for us.

During the Raymarch, all we have to do is to track which segment we are currently in and that always tells us exactly which volumes we have to evaluate for each sample.

Overlapping Volumes & Segments

SIGGRAPH



© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

Here's an in-engine example of how it works. I set up 3 volumes that partially overlap each other.



© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

If we switch to a debug view for the segment count, we can see how the segment count is kept to an absolute minimum, and that we get extremely tight bounds for our volumes.

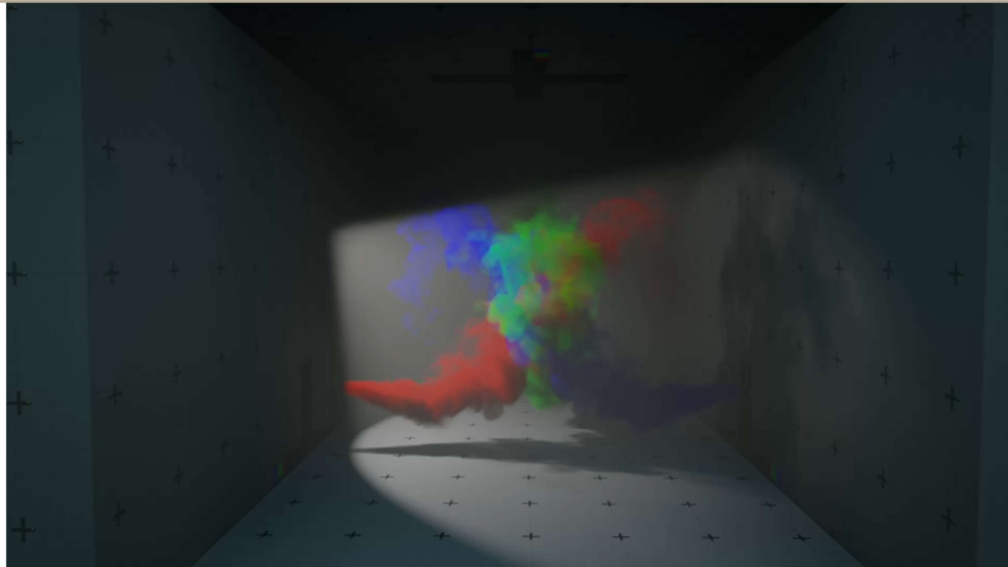
Overlapping Volumes & Segments

SIGGRAPH



© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

We can also step through those segment depths piece by piece to visualize those even better.



© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

Here's an early test of the overlapping volumes in action. That also gives us some nice results like the blending of the colors you can see in the center.

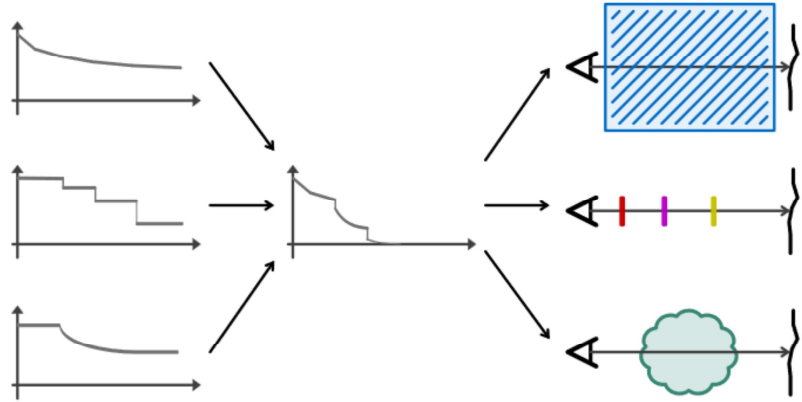
This ticks another box on the way to the fully integrated system we are looking for.

OIT & Volumetric Fog Integration

SIGGRAPH



- Unified transmittance function
 - Enables independent passes
- Passes need to be split into
 - Generate transmittance
 - Resolve scattering
- Already the case for transparent OIT
- Reasonably cheap for volumetric fog
- Pretty expensive for smolder
 - Duplicated raymarch



© 2026 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

When I talked about our raymarching, volumetric fog and transparent rendering at the start of the presentation, I made a point of highlighting that they can all be thought of as transmittance functions. That is because it's key to understanding how we could achieve a perfect integration of those 3 systems in theory.

If we can generate a transmittance function for each of the systems first, we can create a unified function which enables us to have 3 independent render passes for the systems and have them integrate perfectly.

To do this, all passes need to be split into two.

[CLICK]

For OIT that is already the case.

[CLICK]

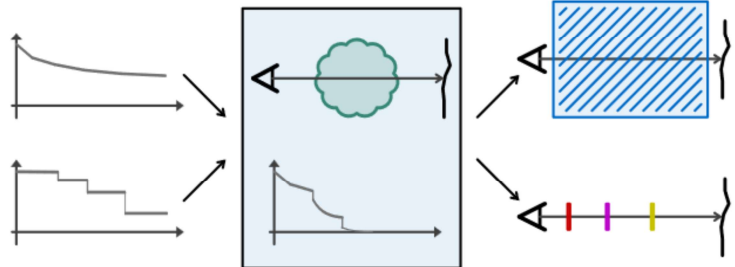
For volumetric fog it is easy to split the current transmittance integration from the scattering one.

[CLICK]

But for Smolder it was not an option, the raymarch cost is too high to do it twice.



- One of the three systems can be single pass
 - Not MBOIT, that one needs to be two passes
 - Smolder is much more expensive to split
- Need to resolve resolution differences
 - MBOIT is per pixel
 - Fog is 8x8 or more pixels/froxel
- Moments can just be mipped
 - REAC 2024 Modernizing geometry rendering in Alan Wake 2



[Aalto24] Modernizing geometry rendering in Alan Wake 2 - <https://www.youtube.com/watch?v=98JgYBQz68>

© 2026 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

The realization I had though is that we do not actually have to split up all 3 systems, one of them can run as a single pass. It just has to run in between the other 2. And since Smolder is more expensive than the volumetric fog, it can just be that single pass system. Sounds easy enough.

[CLICK]

We also have to resolve resolution differences.

OIT has the per pixel moments, and volumetric fog the lower res froxel grid, where a single froxel covers 8x8 pixels or even more.

If we want to apply the transmittance in both directions, we have to mip the moment textures, and sample the mip that matches the froxel resolution.

[CLICK]

I think that is what Remedy already did for Alan Wake 2, which they talked about in their REAC 2024 talk that I linked here.

OIT & Volumetric Fog Integration

SIGGRAPH



So is that how we did it?

So is that how we did it?



So is that how we did it? **Nope :(**

- Rebuilding a large part of the pipeline
 - Time consuming
 - Risky late in production
- Time needed elsewhere
 - Smolder work kept getting pushed for larger optimization work
- Blur & refraction setup overly complicates the issue
- Performance
 - Resolving OIT transmittance for every step of the smolder raymarch is expensive
 - ~20% higher sample cost

Obviously not -- Production realities hit again.

[CLICK]

Changing the setup to what I showed here would have required large changes to the pipeline, which this late in production was just a massive risk.

[CLICK]

It also would take a while and my time was needed elsewhere.

If you have seen our extended gameplay trailer in September, you probably remember that it was extremely choppy despite running on a PS5 Pro.

I was tasked with leading that optimization effort on render side, and that was very much a full time job.

While directors on Bond kept pushing for additional work on Smolder, I had to de-prioritize and delay it continuously based on the work I realized was necessary to get the game into shape.

[CLICK]

We also had another very Glacier specific issue; our blur and refraction pipeline. It supports a lot of very complicated setups, with various layered blurs and particle sorting modes.

That complexity makes it extremely maintenance heavy and basically impossible to integrate with any other system.

[CLICK]

Lastly there is also a performance topic to consider.

Evaluating transmittance based on the moments is not extremely expensive in itself, but doing it for every raymarch step quickly adds up.

In my tests it added roughly 20% to the cost of each sample.



1. Calculate volumetric fog transmittance & integration
2. Apply volumetric fog to opaque scene
3. Most blur & refraction runs at this point
4. Generate transparent transmittance moments
5. Run Smolder
 - a. Keep track of Smolder transmittance moments in local variables
 - b. Store a blue noise jittered integration depth for when Smolder is reasonably opaque
 - c. Remove volumetric fog from scene up to integration depth (subtract partial scattering, divide by partial transmittance)
 - d. Apply Smolder transmittance to background
 - e. Add previously removed partial fog back to result
 - f. Apply fog and transparent transmittance up to integration depth to smolder, add to result
 - g. Add smolder moments to OIT moments
6. Resolve OIT

Okay so let's go through the integration pipeline we ended up with step by step. I kept most of the frame intact and kept most changes constrained to the Smolder pass itself.

We start with our regular volumetric fog integration pass.

[CLICK]

We then apply that volumetric fog to the opaque scene just as before, which is important to not impact the blur and refraction pipeline that runs afterwards.

[CLICK]

The moment generation step in OIT also is unchanged.

[CLICK]

Then we start with the actual Smolder pass.

[CLICK]

During the raymarch, we keep track of the accumulated Smolder primary ray moments in local variables.

[CLICK]

We also store the depth at which Smolder is fairly opaque, and we use the per pixel blue noise to change this offset per pixel and between frames.

[CLICK]

At this point the rest of the pipeline runs after the Smolder raymarch, and it's very hacky.

We start by removing the volumetric fog from the scene up to that integration depth. We can do that by simply removing the scattering up to that point and dividing by the transmittance at that depth afterwards. There is an error we introduce with this, but in reality, it produces good enough results.

[CLICK]

Next, we apply the transmittance from Smolder to that background, and we can add the previously removed partial fog scattering and transmittance back to the result.

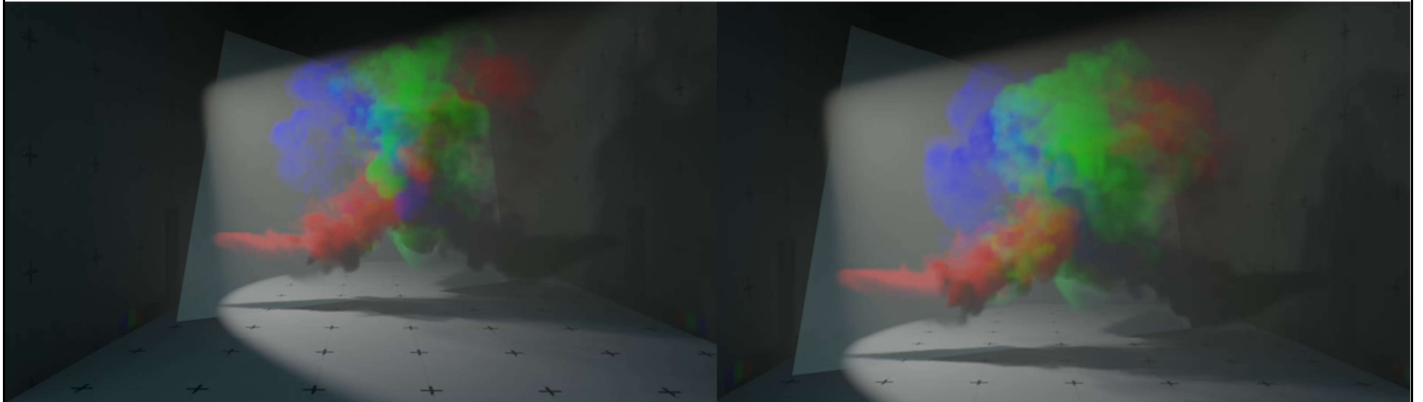
[CLICK]

Before we add the Smolder scattering to the result, we have to apply the fog and transparent transmittance for the integration depth to it.

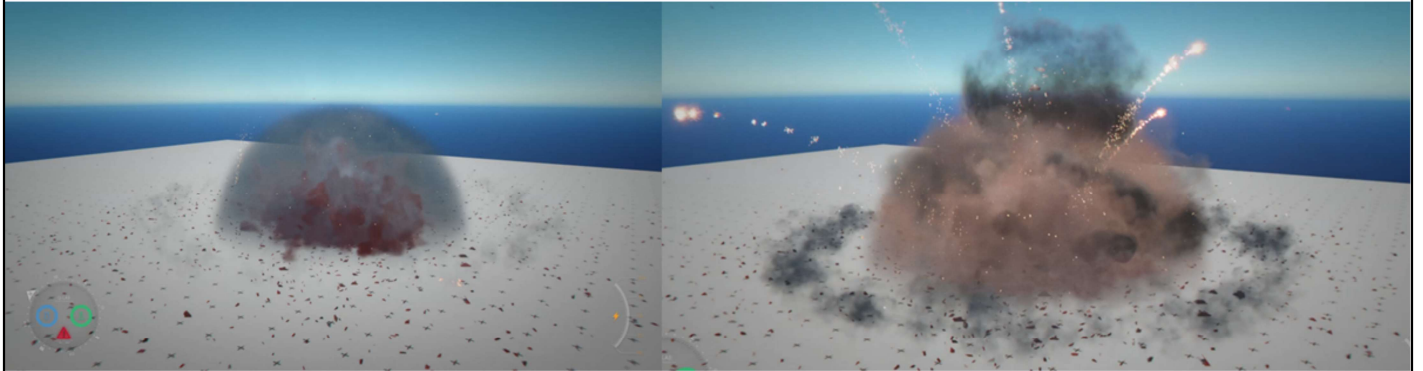
[CLICK]

Lastly, we add the Smolder moments to the transparent moments and run the OIT resolve pass.

This integrates Smolder with all other systems, but it simplifies it with the single integration depth that is used for some parts.



Here are some early results with our imperfect pipeline.
I hate it with a passion, but it was ultimately a practical solution given the timeframe, and it gets the job done.



The tech art team on First Light also built this stress test for me, where they layered all 3 systems and had an explosion go off on top.

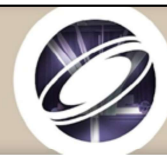


- GI is prefetched ahead of raymarching loop
- Stored in LDS
- One value for the up to 4 visible volumes / wave
- Wave64
 - 4 volumes, 16 threads/volume
 - 8 probes, 2 threads/probe
 - 2 directions sampled
- Wave32
 - 4 volumes, 8 threads/volume
 - 8 probes, 1 thread/probe
 - 2 directions sampled (2 samples/thread)
- Sample weighted locally in thread
- Per volume WaveActiveSum()



For global illumination, I decided to limit sampling it to 1 sample position per volume. The baked occlusion helps to sell it. We can calculate that sample before starting the raymarch, and store those samples in LDS.

I ended up sampling 2 directions per probe, which is not ideal but seemed good enough. We then weigh the sample locally based on the trilinear interpolation weight and can use WaveActiveSum for each of the volumes to accumulate them.



- Built clustered lighting as part of overall optimization work
 - Had smolder optimization in mind
 - Context specific bitmasks
 - Cull as many lights as possible ahead of time (light types, flags, rooms, etc)
 - Can be OR'd if neighbouring pixels land in different clusters in depth



© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

I mentioned that I had to work on optimization a lot during that time, and one thing I did was transition our entire light culling pipeline from the old tile-based system to a modern, clustered lighting pipeline.

I did that with Smolder in mind.

At the core this pipeline is built around context specific bitmasks.

We store a list of lights for each screen area in 2D, and the actual clusters just contain bitmasks for those lists.

Those bitmasks do not need a lot of memory, so we can have specific bitmasks depending on the use case.

For example, we don't need character specific lights or box reflections in Smolder.

We can also just OR the bitmasks together at runtime when neighboring threads share the same cluster X and Y coordinates but have different sample depths and land in different cluster Z coordinates.

Since we have the segments from the culling pass, we can just utilize this information to quickly discard a lot of clusters and lights.

You can see in this example here how we only have lights in clusters that overlap with the volume, and we end up with 7 or 8 lights per cluster.



- Dynamic shadow culling system
 - Starts with all lights in view
 - Cull by type, flags, ...
 - Cull by visible smolder volumes
- All lights in the world cast smolder shadows!
- Last piece of the workflow puzzle



© 2026 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

At this point we had almost all the issues solved that we had to solve for the original dream. The last missing piece was culling of the volumetric shadows.

Up to this point, artists enabled them manually on some key light sources, and as you can imagine that was extremely limiting and came with a lot of performance overhead.

I wanted volumetric shadows enabled on all lights by default, with a toggle for the artists to explicitly disable them if needed.

To achieve that, we needed a dynamic culling system for them.

We start by using the list of frustum culled lights that we already had.

That list can be culled more based on various light properties, similar to the lights we cull from the Smolder light clusters.

But most importantly, we don't have that many Smolder volumes around.

We can easily just cull all the lights that do not overlap with any of those Smolder volumes and therefore don't need volumetric shadows.

That brings the shadow count down to a very manageable number, usually around 10-20 in most areas.

[CLICK]

I recorded the video you can see here at that time; it is an area of the game that was not set up to have any Smolder volumes around.

It shows the power of the culling system, but more importantly it was the first time where I saw the full promise of Smolder finally come to life, where I could drop a volume in a random spot and it just worked. The lighting, the volumetric shadows, the transparent

integration with the bottles on the table.

Shadow Culling

SIGGRAPH

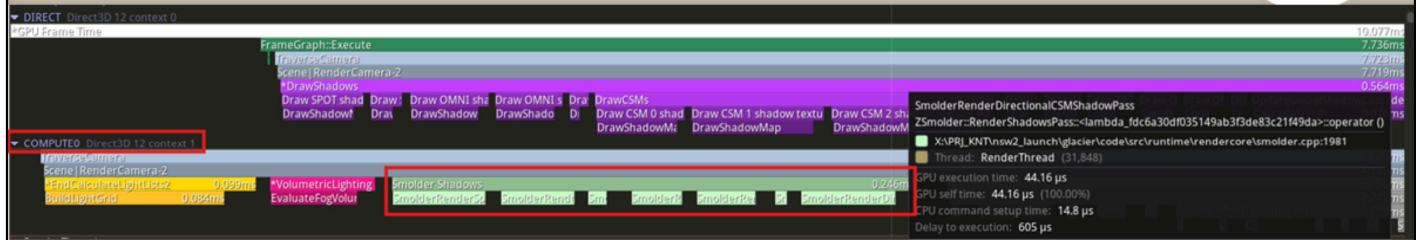


© 2026 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

This also creates one of my favorite Smolder party tricks, where we can just use a volume to darken an entire area in the game. This works in the shipped game with the smoke gadget; the player can just throw that at a light source and you'll get this effect.

Shadow Async Compute Setup

SIGGRAPH



- Fully compute shader based shadow rendering
- No dependencies on any GPU frame data
- Can run in async queue when nothing else can run
- Throttled on consoles to prevent it from stalling the graphics queue
- Basically free™)

© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

Having 10-20 of these volumetric shadows around still sounds like they would be pretty expensive to render, but they actually aren't.

Since they are running in compute shaders, we can just run them in async compute.

And since they have no dependency on any GPU frame data and only need the volumes as input, they can run in places during the frame where almost nothing else can run.

On the consoles, we can throttle them heavily and we do that to make sure that they don't slow down any work happening on the graphics queue.

With all of that the Smolder shadows use GPU resources that would have otherwise been unused, which makes them essentially free to render.



© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

We are at a point now where all the core technical systems are implemented and the challenges we had were solved, so let's take a look at performance and quality.

One of the stated goals for Smolder was to not have any temporal filtering in the system itself. But that being said, there is some filtering for the entire frame in the form of TAA.

That gave us some opportunity to improve the Smolder visual quality.

The most obvious one is to use blue noise to offset the sample depth during the raymarch per pixel.

It cleaned up the structural artefacts nicely that we had up to this point.

In return we had to combat the ghosting we had due to the lack of motion vectors for the volumetrics.



- FSR 3
 - Reactive mask needed to avoid heavy ghosting
 - Lower transmittance = more reactive
- DLSS
 - Okay results for presets E/F (CNN)
 - Ghosting issues with J/K (1st gen transformer)
 - Okay results with presets L (2nd gen transformer)
 - Great results with presets M (2nd gen transformer)
 - No dedicated inputs to improve result
 - Keeping noise per pixel helps a lot
- Upgraded PSSR
 - Great results out of the box

We shipped with 3 different TAA systems, and we had to adjust our approach based on the inputs we had available.

For FSR 3, we had to hook Smolder up to the reactive mask input, otherwise we got very heavy ghosting.

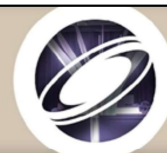
[CLICK]

For DLSS, it's a wide range of presets and models under the hood, and there is no equivalent to the reactive mask input. I primarily struggled with the early transformer model. It was extremely important to keep the noise per pixel for it to be able to produce good results, and I had to remove some test code that ran lighting half res in some situations, it caused most of the issues.

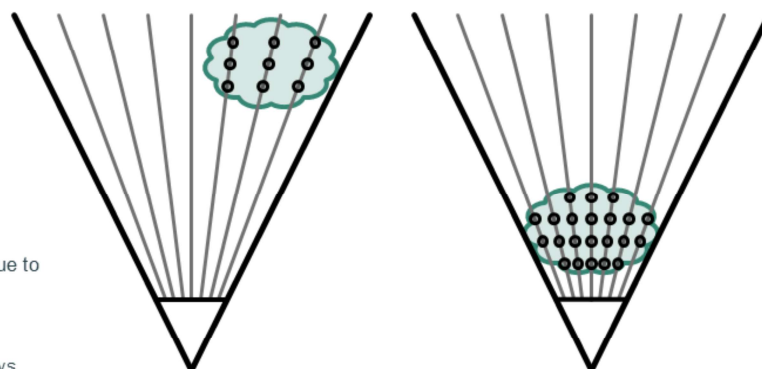
[CLICK]

Lastly, for the upgraded PSSR we got great results out of the box and luckily didn't have to do any specific work for it. So shoutout to our previous speaker Daniel and his team for the work they did on it, they made my life a lot easier.

I also have to thank the teams at the hardware vendors that answered a lot of questions from me and helped me to get the most out of their respective tech.



- Already in a decent spot overall for results
 - Good core architecture
 - Very high wave coherency
 - Segments reduce sample count to a minimum
 - Sample size adaptive based on voxel size
 - Smallest one chosen in overlapping areas
 - Utilize wave intrinsics to vectorize data loads
 - Volume data samplers
 - Clustered lighting data
 - Common sense operations
 - Early out when transmittance close to 0
- Main issue: Very high sample density close to camera due to perspective projection
- Production reality hits again
 - Original 3 week optimization window turned into 2 days
 - Had to focus on highest impact optimizations



For performance, we were overall in a pretty decent spot.

Building a good core architecture for your algorithm is crucial for that, and Smolder's high wave coherency is extremely powerful here.

The segments already reduce the sample count to a minimum, and we adjust the sample size based on the volume's voxel size to not oversample it.

In overlapping areas we chose the smaller step size.

We vectorize data loads with wave intrinsics where possible and we can easily trade between LDS and VGPR usage.

We also do the typical common sense operations, like stopping the raymarch early when transmittance is close to 0.

[CLICK]

The main performance issue that was left was the scaling of the sample count due to the projection matrix.

[CLICK]

The cost of rendering scales with the screen coverage of it.

[CLICK]

I had 3 weeks in January planned in to deal with that particular issue.

Production realities hit again though, our overall render performance was still not where it needed to be at this point and I had to keep focusing on that.

So those 3 weeks in January eventually turned into 2 Saturdays in April.

I had to focus on the most obvious idea; to render the volumes at half or quarter res when close to the camera. Clearly I didn't learn my lesson from the volumetric particle attempt.

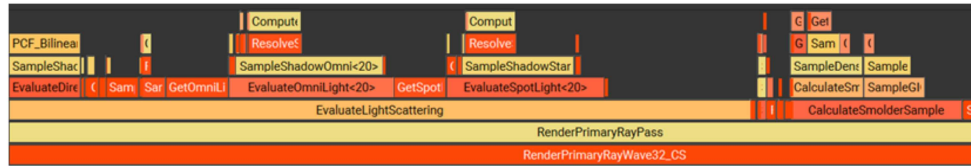


- Most obvious idea is to render half or quarter res close to the camera
- Spatial upsampling quality wasn't great for quarter res
- Broke per-pixel rule of the TAA integration – large blocky noise artefacts



© 2026 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

It was very easy to implement, but the spatial upsampling was not producing good results. Also, it broke the per pixel noise rule for the TAA integration, and it resulted in extremely heavy ghosting. I decided that I needed to find a different solution for this performance issue.

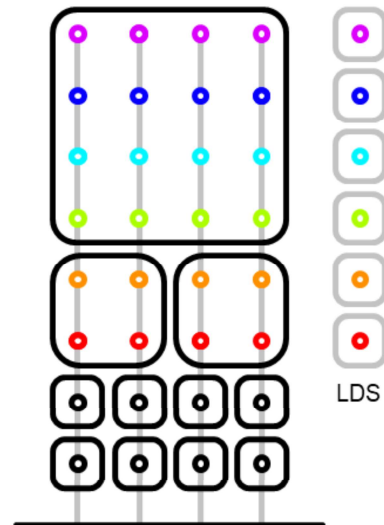


- 80-85% of sampling cost is lighting
 - Instead of changing resolution for the entire sample, what if we just share the lighting work?

Looking at a capture of Smolder, I saw that a large amount of the sample cost came from the lighting calculations, the density lookup was only a small fraction of it. So instead of downscaling the entire image, I focused on evaluating lighting at lower fidelity.



- Track light sampling resolution per wave, evaluate in 4x4 pixel groups (full, half, quarter)
- Start at full resolution
- Drop to a lower sampling rate when transmittance is below a threshold
- Reduced quality inside of the volumes where precision isn't needed as much
- Still keep per pixel depth testing and OIT/Fog integration



© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

The idea is pretty simple; we track a sampling rate for the lighting in the wave, and we have groups of threads drop to a lower sampling rate and share the work within the group.

[CLICK]

We start at full resolution.

[CLICK]

Once transmittance drops below a defined threshold for the entire wave we drop to half res.

[CLICK]

The threads in the group now each still take 1 sample, but they take it at different depths.

They keep their original rays for it though; this way we spatially distribute the samples in the group.

[CLICK]

The threads then store their sample in LDS.

[CLICK]

Finally, we use those cached samples to continue the per pixel raymarch.

[CLICK]

We do the same once we drop to quarter res. The visualization here is in 2D, half rate means 4 samples are shared, and quarter rate means 16 are shared in a single step.

Dropping the rate based on transmittance means we render at lower resolution within the volume, where we are already partially occluded and precision isn't needed as much.

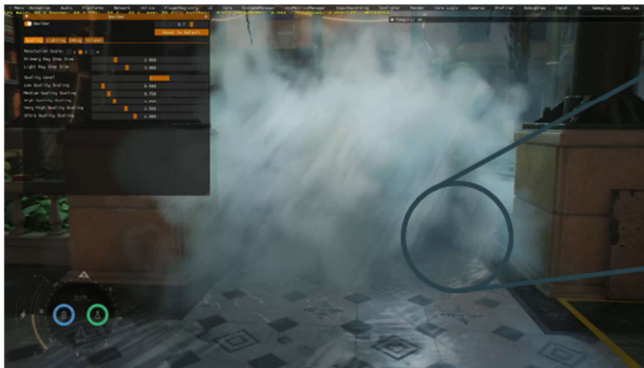
Also, since we still accumulate the results per pixel we still benefit from per pixel depth testing and per pixel OIT and fog integration.

Since everything is within a wave we don't need to go through LDS, wave intrinsics would have worked. But I had LDS space to spare and needed all the occupancy I could get, so

this was the better solution in that case.



- Still some artefacts left
 - TAA really needs per pixel noise to resolve
 - Thresholds would need to be so low that almost no performance was saved



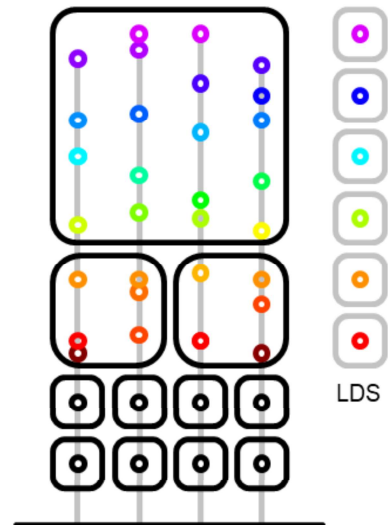
But once again, TAA was not having it. Even when deep inside the volumes, it caused massive noise and ghosting. To avoid it the thresholds would need to be so low that we don't save any performance at all. It was clear that we need the per pixel blue noise back.

Adaptive Sampling Groups

SIGGRAPH



- We already blue noise jitter the depth samples to avoid artefacts
- Apply the same logic for the shared samples
 - Store group samples in LDS
 - Interpolate between two of them at a time based on blue noise depth jitter offset
- Resolves artefacts, allows for much more aggressive thresholds
 - Saved as much as 50% in heavy smoke gadget case
- Two new quality/performance scaling options
 - Adaptive group starting size
 - Adaptive group thresholds



© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

I ended up still storing the samples in LDS first, up to that part the logic is identical.

[CLICK]

But then instead of just reading and applying them, we interpolate between two of them at a time with a per pixel blue noise jitter for the lerp value.

[CLICK]

This mimics the exact thing we already did when jittering the sample depth with blue noise earlier, just with less precision.

[CLICK]

And we do the same for quarter res again.

It worked great, and it allowed for much more aggressive thresholds, and roughly 50% savings in performance in some of the heavy smoke gadget cases.

It also gave us yet another method to scale to lower end targets and trade some quality if needed, for example we could just choose to start the raymarch at half precision there.



© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

Here's an example of the quality we still got and the debug view for the adaptive sample groups. In terms of quality it's extremely close to running without the optimization, we still get the precision where it matters most.

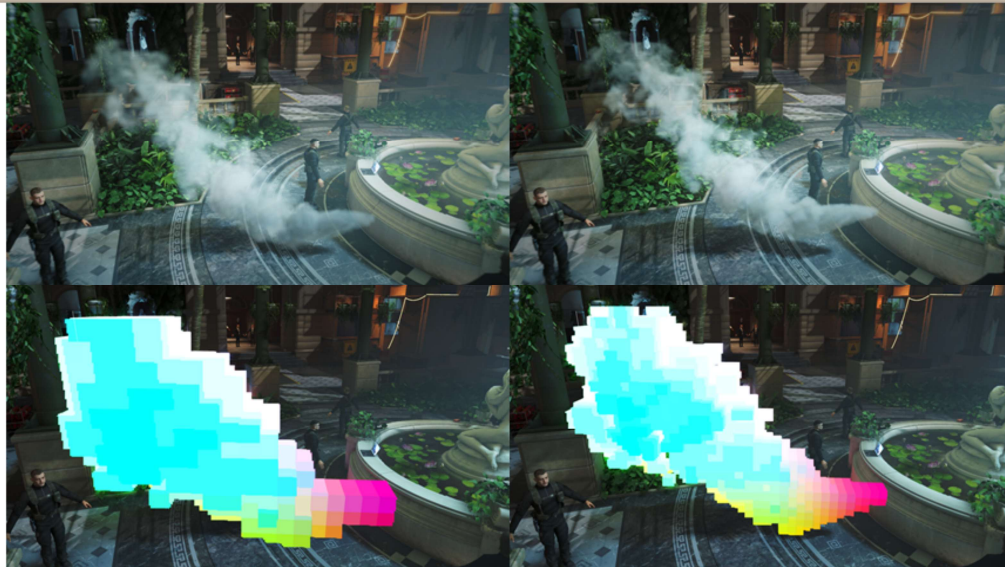
Red in the debug view means the wave went all the way down to quarter res, green means half res and blue means the wave always stayed at full res.

The optimization worked great for the dense smoke gadget....



© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

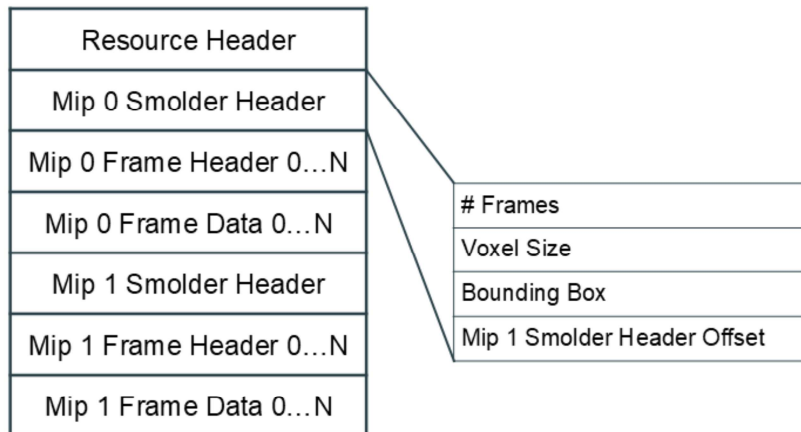
... but it is less efficient for less dense volumes. It still provides some very good savings there too, and since these kind of volumes didn't cause us any real performance issues to begin with, this was good enough to ship with.



© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

Another optimization we had to do was specifically for the XBOX Series S. We were fighting with the memory budget there until the very end. Since we couldn't stream individual Smolder frames and only the entire animations at once, Smolder could use a few hundred MBs of memory total in some cases. Not an issue on the bigger consoles where we even had some memory to spare, but not something we could get away with on the Series S.

I decided to just generate a half res version of the volume in the packer, that reduced the memory footprint of each animation by roughly 80 - 85%. One of our core engine programmers actually ended up implementing the change to help me out during those busy times, since it didn't require any changes to the runtime code.



Same resource

- Higher storage footprint
- XBOX Series S and X share resources
- Scaling options on PC

What he did was to simply use the OpenVDB library in the resource baker to downscale the input to it, and then he just ran the existing code as is. The Smolder header got an additional field that contained the offset to the next mip. XBOX Series S and X share the same resources in our engine, so we had to put both mips on disc and chose which one to use at load time. Not perfect, but also not an issue. As a nice side effect this also gave us some additional scaling options on PC, where we also use the half res mips on lowest settings to fit into our 6gb vram min spec.

What we shipped

SIGGRAPH



© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

And that's it for the implementation side. Here's a couple of examples from within the game.

What we shipped

SIGGRAPH



© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

For the clouds, this generic approach produced some good results, but ideally the lighting model would be slightly different. That might be something to investigate in the future.

What we shipped

SIGGRAPH



© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

Another shot I really like...

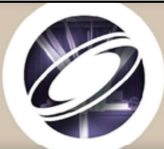
What we shipped

SIGGRAPH



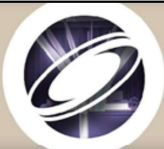
© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

... and by far my favourite shot from the entire game, right at the start of it.

Performance Numbers SIGGRAPH 					
PS5 Equivalent Settings	RTX 3060	RX 6700 XT	RTX 5060 Ti (DLSS Quality)	RTX 5060 Ti	RTX 5060 Ti
	1080p	1080p	1080p(720p)	1080p	4k
	1.230ms	0.775ms	0.367ms	0.637ms	2.410ms
	1.080ms	0.642ms	0.299ms	0.630ms	2.010ms
	3.462ms	1.970ms	1.020ms	1.820ms	7.160ms
© 2026 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.					

Let's talk about some performance numbers, here's a bunch of cases with PS5 equivalent quality settings. These do not contain the async shadow rendering cost.

Overall, the system performs pretty well for what it does I think, but there is definitely a lot of room for improvement. I have more than 20 performance related ideas on my todo list though, so I'm reasonably confident that Smolder can still run a lot faster.

Performance Numbers - Wave64					
PS5 Equivalent Settings	RTX 3060	RX 6700 XT	RTX 5060 Ti (DLSS Quality)	RTX 5060 Ti	RTX 5060 Ti
	1080p	1080p	1080p(720p)	1080p	4k
	1.230ms	0.775ms	0.367ms	0.637ms	2.410ms
	1.080ms	0.642ms	0.299ms	0.630ms	2.010ms
	3.462ms	1.970ms	1.020ms	1.820ms	7.160ms
© 2026 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.					

I want to point out a few things here, starting with the fact that AMD cards tend to perform way above their weight class in Smolder. This also applies to the AMD based consoles.

This is due to them running in wave64 mode and Smolder's heavily collaborative code can utilize that a lot. We get some inefficiencies in the culling compared to 32 sized waves, but that doesn't impact performance a lot.

That being said, I think this is also a hint that it is possible to get a lot more performance out of Smolder on cards that run with 32 sized waves, and I will have to investigate that going forward.

Performance Numbers					
PS5 Equivalent Settings	RTX 3060	RX 6700 XT	RTX 5060 Ti (DLSS Quality)	RTX 5060 Ti	RTX 5060 Ti
	1080p	1080p	1080p(720p)	1080p	4k
	1.230ms	0.775ms	0.367ms	0.637ms	2.410ms
	1.080ms	0.642ms	0.299ms	0.630ms	2.010ms
	3.462ms	1.970ms	1.020ms	1.820ms	7.160ms

© 2026 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

Another thing to point out are the performance numbers on the 5060Ti at different resolutions. What you can see is that we get extremely linear performance scaling with the number of samples we take, in this case based on the number of pixels we render. This tells us that the decoupling of primary rays and shadow rays definitely worked.

We see the same linear scaling on console where we run with dynamic resolution scaling. Thanks to that linear behavior, dynamic resolution scaling is extremely efficient, and doesn't need to drop resolutions overly aggressively.

What we shipped

SIGGRAPH



- Core tech goals achieved
 - Generic volumetric effect renderer
 - Scales to all of our target platforms, shipped on XBOX Series S, Steam Deck, low end PCs (GTX 1660, RX 5700)
 - Integrated with the rest of the game
 - Player placed effects
- Workflow goals achieved
 - >450 volumes in the game
 - Artists adopted the new tech very quickly
- Still very much a first version
 - Performance is good but some scenarios still need work
 - Way less time to work on Smolder than expected due to other performance work

© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

So what did we end up shipping in 007 First Light? If you remember the original requirements list, everything from there made it into the game. We achieved our goal of having a fully integrated, generic volumetric effect renderer that performs well enough to ship it in the game. While we didn't release on Switch 2 yet, we already shipped Smolder on XBOX Series S, Steam Deck and our low-end PC targets.

[CLICK]

One stat I looked up just before release was that we ended up having more than 450 Smolder volumes in the game, so much for building some tech for the smoke gadget 😊 To me, this is a great sign, and I am extremely happy to see that. The artists adapted to the uniqueness of working with volumetric effects quickly, and I think it shows that the goal of supporting a great workflow with Smolder worked out. I could have just done without some of the stress and headache that was caused by pushing the tech into the game this late.

[CLICK]

However, I also need to make it clear that this is still very much a first version. The performance is good but some scenarios still need work. I also had way less time for Smolder than I hoped for, and I'm looking forward to getting back to it at some point and continuing where I left it.

What's missing

SIGGRAPH



- More performance optimizations
 - Near rendering
 - Utilize mips outside of low end platforms
 - SDF based empty space skipping
 - FP16?
 - ...
- Memory optimization / streaming individual frames
 - Better quality scaling to high end GPUs for baked effects
- Real-time fluid simulation
- Full integration with OIT and volumetric fog
- More artist tools
 - Color/Emissive support
 - Per volume phase function support
 - Volumetric materials
- ...

© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

In terms of the next todos, there's quite a long list.

[CLICK]

Apart from the performance work, I would like to properly stream the animations to not be limited by memory constraints.

[CLICK]

Also we will definitely work on getting that real-time fluid simulation into Smolder, that would be a massive upgrade to the tech.

[CLICK]

I have to fix the integration of OIT and volumetric fog and do it properly, and I would also like to give the artists more tools to work with.

[CLICK]

Color and emissive support come to mind, having support for a per volume phase function instead of using a single generic one would also be a great thing to have. Long term I would like to have support for volumetric materials that artists can design with their usual material shader graphs, but that one might be a bit too ambitious.

[CLICK]

Lots to do still, and I am definitely looking forward to talking about Smolder part 2 at some point in the future.

Shoutouts

SIGGRAPH



Martin Bjarke Kragh Nielsen

Asger Meldgaard

Owen Angell / Jon Rocatis

Arnau Prat Perez / Ivan Zaplava

IOI Render Team

IOI Tech Art Teams

Natalya Tatarchuk

© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

I have to give some shoutouts to the team. Most of all Martin for pushing this tech with me and Asger who is building the fluid simulation tech. Both constantly provided extremely valuable feedback and I very much appreciate that they didn't lose their patience with me when the first attempt didn't work out.

On render side I want to especially thank Owen and Jon for doing hundreds of code reviews for me, as well as sitting through some sanity check meetings I had with them whenever I needed a second opinion.

I also want to thank Arnau and Ivan on the core team for the work on the OpenVDB resource packer, as well as the entire IOI tech art and render teams for their work.

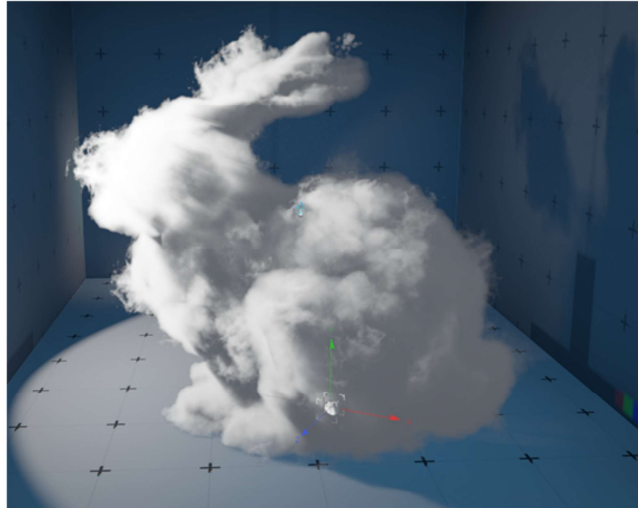
Lastly, I want to thank Natasha for letting me present here and taking the time to review all the slides and provide feedback.

Thank You!

SIGGRAPH



Q&A Time



© 2020 SIGGRAPH Advances in Real-Time Rendering in Games course. All Rights Reserved.

Thank you for joining me today for this talk. I leave you with a Smolder version of the legendary Stanford bunny and I think we have some time for Q&A now.