Path Tracing in Production - Volumes

Christopher Kulla

SIGGRAPH 2019

Hello, I'm Chris Kulla from Sony Imageworks and I will be now talking about volumes.

(production image omitted)

Spider-Man: Homecoming ©2017, CTMG Inc.

Just to set the stage of what I'll be discussing today, here are a few images from some movies we worked on recently at Imageworks …

(production image omitted)

Spider-Man: Far From Home. ©2019, Columbia Pictures

…and as you can see we're filling most of the screen with volumetric effects.

These are usually combined from hundreds of individual elements because many artists might be collaborating to make a single shot like this.

(production image omitted)

The Meg. ©2018, Warner Bros

Volume rendering isn’t just for smoke and explosions, even rendering water uses the volume features of the renderer because we need to keep track of its thickness and IOR.

(production image omitted)

Smallfoot 2018, ©Warner Animation Group

Here is a frame from the movie Smallfoot where volumes were used for the clouds and snow.

(production image omitted)

Spider-Man: Into The Spider Verse, ©2018, Columbia Pictures

Even when the film is very stylized, volumetric lighting is still a fundamental tool that artists rely on all the time.

Outline

- Sampling Techniques
- Volume Intersection
- Overlap Handling
- Open Problems

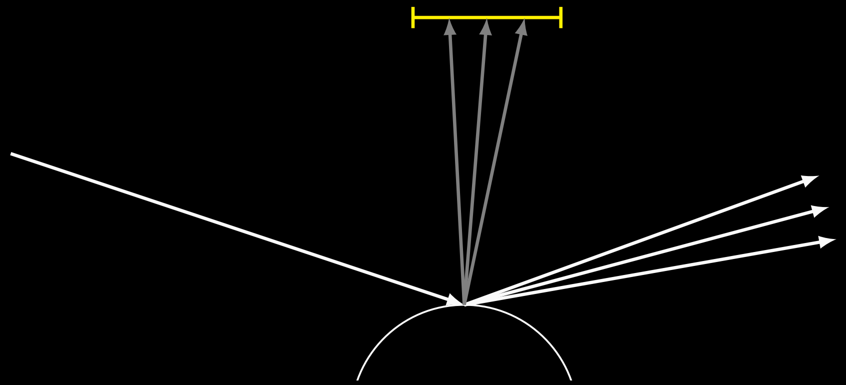
So here is a quick outline of what I'll be talking about:

I'll try to give an overview of the sampling techniques and compare a few different types of ray marching.

Then I'll talk about the problem of volume intersection for the interesting cases that come up in production like frustum aligned volumes and motion blur. I'll also talk about the rules for overlapping volumes which are usually ignored in research oriented renderers.

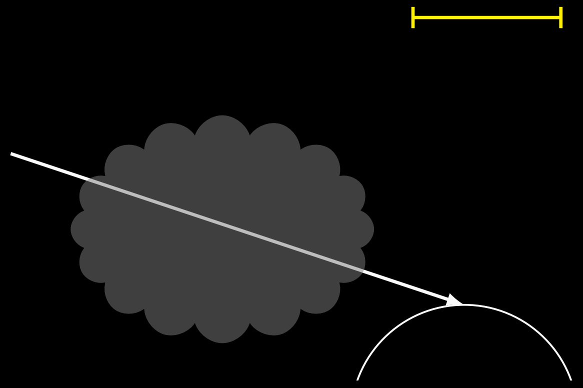
And finally I'll talk about some of the open problems that are of interest to us in the film industry.

Path Tracing + Next Event Estimation



So by now in the course you have a pretty good understanding of basic path tracing for surfaces. At a high level, we just scatter rays randomly according to the BSDF and connect to light sources.

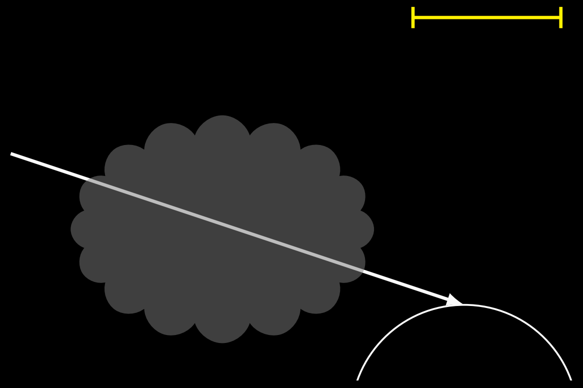
Volume Path Tracing



So how do we extend this to account for volumes?

The volume is continuous representation, so the ray tracing call needs to be replaced with some kind of sampling decision.

Volume Path Tracing

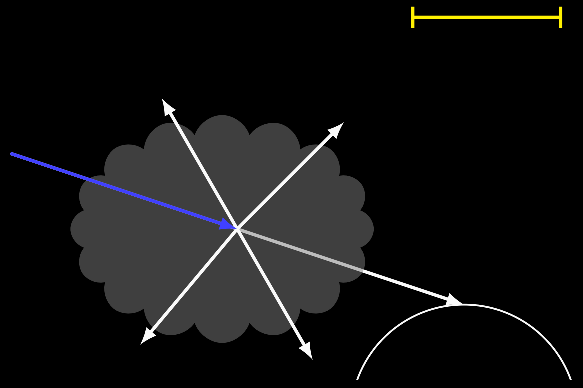


Sample t from $\mathrm{pdf}(t)$

Lets assume for a moment we have a pdf defined along the ray.

Sampling that pdf would produce candidate distances for us to continue our random walk from.

Volume Path Tracing



Sample t from $\mathrm{pdf}(t)$

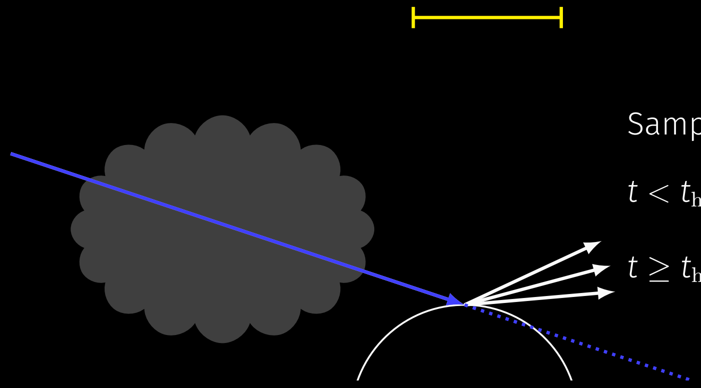
$$t < t_{\text{hit}} \rightarrow \frac{\mu_s(t)T(t)}{\mathrm{pdf}(t)}$$

If that random position is closer than the surface, we scatter in the volume.

The formula just takes the scattering coefficient times transmission and divides by the pdf. This is just normal importance sampling.

Remember that scattering and transmittance are colors while the pdf is a scalar. So its good to keep this form in mind even though lots of things can cancel out in simple cases.

Volume Path Tracing



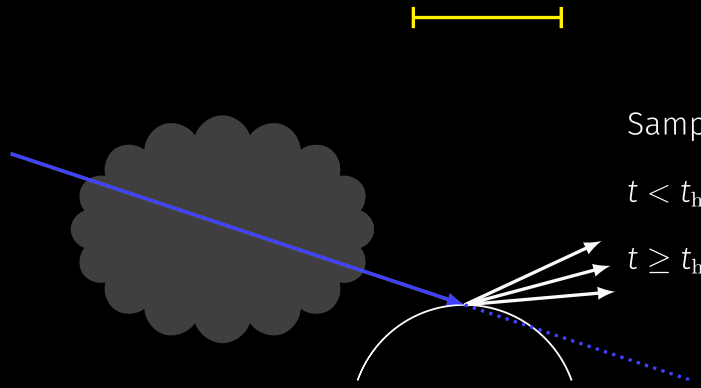
Sample t from $\mathrm{pdf}(t)$

$$t < t_{\mathrm{hit}} \rightarrow \frac{\mu_S(t)T(t)}{\mathrm{pdf}(t)}$$

$$t \geq t_{\mathrm{hit}} \rightarrow \frac{T(t_{\mathrm{hit}})}{\mathrm{P}(t \geq t_{\mathrm{hit}})}$$

If the sampled distance is behind the surface, we divide the transmittance by the probability of generating a distance beyond that hit point.

Volume Path Tracing



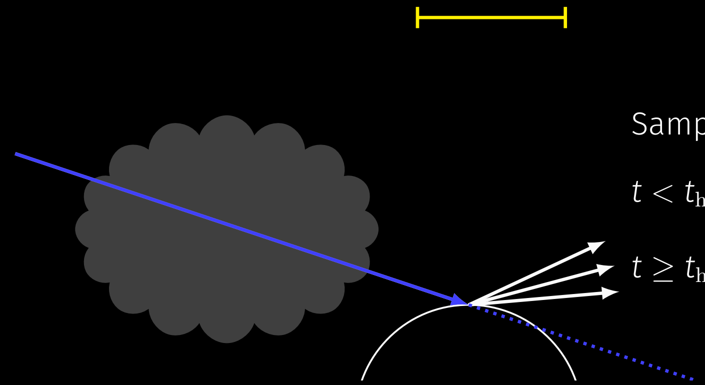
Sample t from $\mathrm{pdf}(t)$

$$t < t_{\mathrm{hit}} \rightarrow \frac{\mu_S(t)T(t)}{\mathrm{pdf}(t)}$$

$$t \geq t_{\mathrm{hit}} \rightarrow \frac{T(t_{\mathrm{hit}})}{\int_{t_{\mathrm{hit}}}^{\infty} \mathrm{pdf}(s)ds}$$

To get that probability we have to add up the probability of all possible ways we might have gone beyond the surface which is a small integral…

Volume Path Tracing



Sample t from $\mathrm{pdf}(t)$

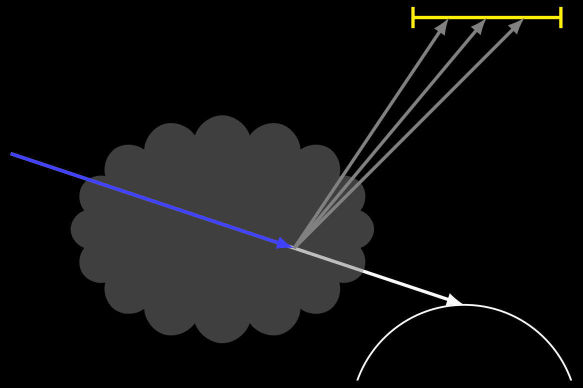
$$t < t_{\text{hit}} \rightarrow \frac{\mu_S(t)T(t)}{\mathrm{pdf}(t)}$$

$$t \geq t_{\text{hit}} \rightarrow \frac{T(t_{\text{hit}})}{1-\mathrm{cdf}(t_{\text{hit}})}$$

...but we know that the integral of the pdf is the cdf, so we can compute it easily.

And that's it! Most of the difficulty of path tracing volumes comes from estimating the transmittance and pdf terms in these equations, so its good to keep this general form in mind.

Volume Path Tracing + Next Event Estimation

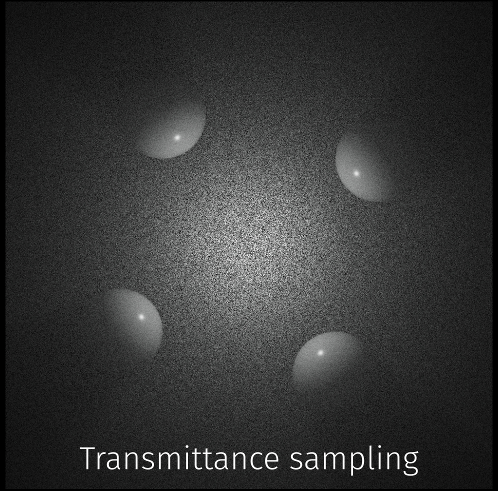


NEE can use different pdf!

Just like with surfaces, we can use next event estimation to connect to the lights.

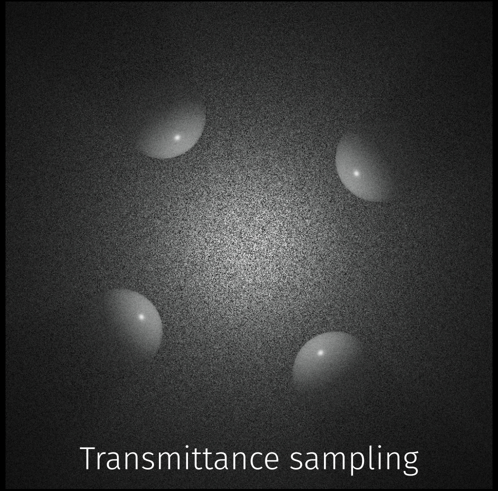
In fact, we probably want to use a different pdf in this case because we have extra information.

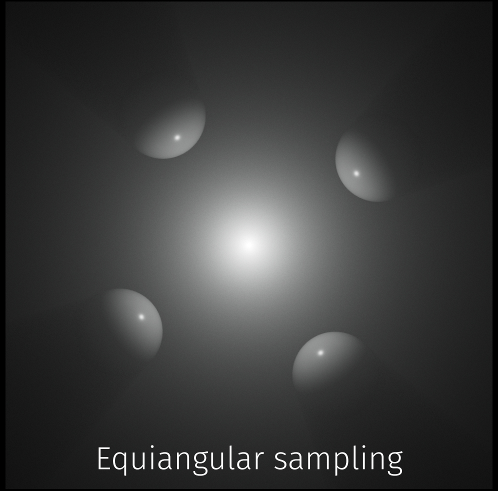
Single Scattering Results (16 samples / pixels)



The light source is a strong source of variance when its embedded in the volume. So if we sample only according the transmittance function, the samples aren't really taken where the light is strongest.

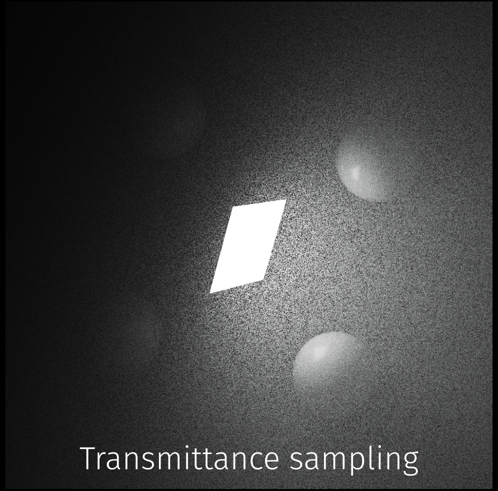
Single Scattering Results (16 samples / pixels)

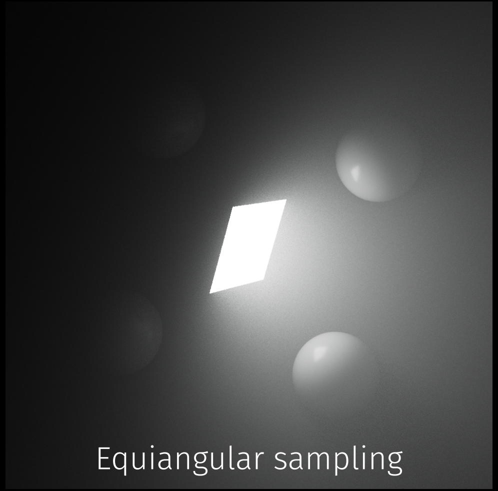




You can do much better with a pdf proportional to the inverse squared falloff from the light which is what equiangular sampling does.

Area Light Results

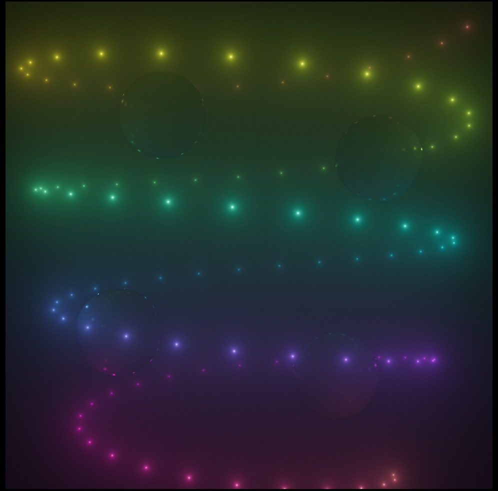

Transmittance sampling


Equiangular sampling

This works even for area lights because we always have an inverse squared falloff in the geometry term when connecting two points.

The results will be even better if you combine this with solid angle sampling of the light because the solid angle has an inverse squared falloff in it implicitly. So then the weight of each ray is almost constant which reduces variance even more.

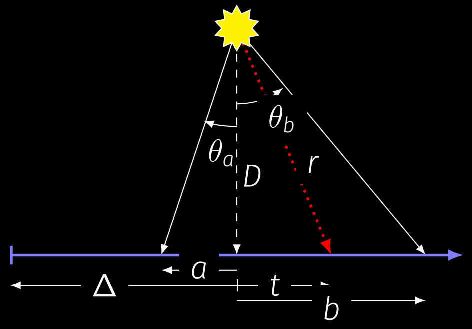
Many Lights Results [Conty et al., 2018]



The same thing works if you have lots of lights.

We just heard about light hierarchies. In the case of volumes you want to traverse the hierarchy using the entire ray as the query so that the light you select can decide where to put the samples along the ray.

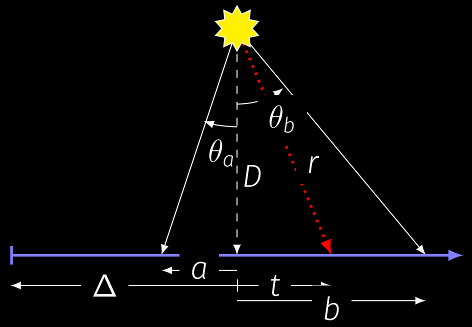
Equiangular Sampling [Kulla et al., 2012]



$$
\begin{aligned}
\mathrm{pdf}(t) &= \frac{D}{(\theta_b - \theta_a)(D^2 + t^2)} \\
\mathrm{t}(\xi) &= \Delta + D \tan\left((1 - \xi)\,\theta_a + \xi\theta_b\right)
\end{aligned}
$$

I'll quickly just show the equations here. I think this technique is fairly well known by now so I won't go through the derivation.

Equiangular Sampling [Kulla et al., 2012]



$$\mathrm{pdf}(t) = \frac{D}{(\theta_b - \theta_a)(D^2 + t^2)}$$

$$t(\xi) = \Delta + D\tan\left((1-\xi)\,\theta_a + \xi\theta_b\right)$$

If $D = 0$:

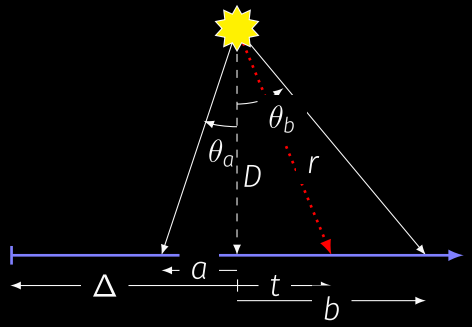
$$\mathrm{pdf}(t) = \frac{ab}{(b-a)t^2}$$

$$t(\xi) = \Delta + \frac{ab}{b + (a-b)\xi}$$

I'll just point out there is a limit case when the light is exactly on the ray.

It might sound like a corner case but it can happen if an artist attaches a point light to the camera for example, which is how we noticed we were missing it.

Equiangular Sampling [Kulla et al., 2012], [Conty et al., 2018]



$$\mathrm{pdf}(t) = \frac{D}{(\theta_b - \theta_a)(D^2 + t^2)}$$

$$\mathrm{t}(\xi) = \Delta + D \tan\left((1-\xi)\,\theta_a + \xi\theta_b\right)$$

If $D = 0$:

$$\mathrm{pdf}(t) = \frac{ab}{(b-a)t^2}$$

$$\mathrm{t}(\xi) = \Delta + \frac{ab}{b + (a-b)\xi}$$

Choose among many lights $\propto 1/D$

And finally I'll mention that for many lights you want to select a light according to $1/D$: the inverse of the distance between the light and the ray. Equiangular sampling is already going to cancel out the inverse squared falloff so all that is left is the distance D from the numerator of the pdf that will move to the denominator when you divide by it.

We have more details about this in our HPG paper from last year.

Heterogeneous Volumes

Homogeneous Transmittance:

$$
\begin{aligned}
T(t) &= \exp\left(-\mu_t\, t\right) \\
\mathrm{pdf}(t) &= \mu_t \exp\left(-\mu_t\, t\right) \\
\mathrm{cdf}(t) &= 1 - \exp\left(-\mu_t\, t\right) \\
\mathrm{t}(\xi) &= \frac{-\log\left(1-\xi\right)}{\mu_t}
\end{aligned}
$$

So those are the basics of single scattering, but I haven't said anything about transmittance. If the volume has constant density, the formula is simple, and finding a pdf is easy...

Heterogeneous Volumes

Heterogeneous Transmittance:

$$\begin{aligned} T(t) &= \exp\left(-\int_0^t \mu_t\left(\mathbf{x}_s\right) ds\right) \\ \mathrm{pdf}(t) &= ??? \\ \mathrm{cdf}(t) &= ??? \\ \mathrm{t}(\xi) &= ??? \end{aligned}$$

…but as soon as we allow the density to change as a function of position, the equation looks pretty intractable. Remember that in a production renderer, the inside of that integral is computed by a procedural shader or by accessing some voxel data.

Heterogeneous Volumes

Heterogeneous Transmittance:

$$\begin{aligned} T(t) &= \exp\left(-\int_0^t \mu_t(\mathbf{x}_s)\,ds\right) \\ \mathrm{pdf}(t) &= ??? \\ \mathrm{cdf}(t) &= ??? \\ \mathrm{t}(\xi) &= ??? \end{aligned}$$

Three strategies:

- Analytic (Exact)
- Ray Marching (Biased)
- Null Scattering (Stochastic)

There are three basic strategies we can take to tackle this equation:

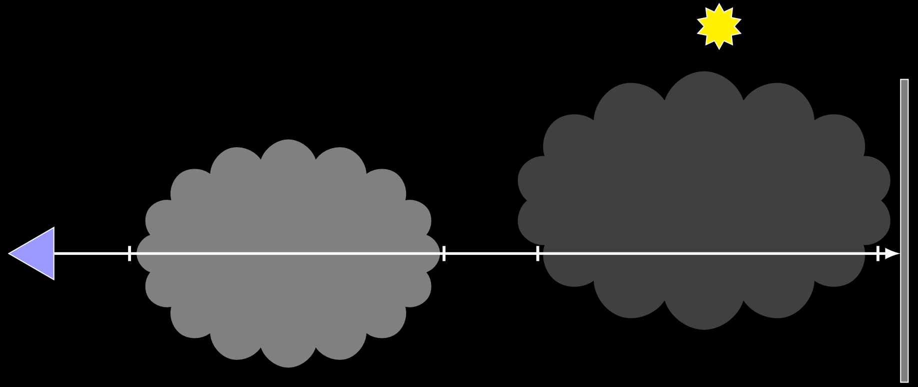
The Analytic scheme, where we can somehow exactly solve the equation (this actually includes regular voxel grids).

Ray marching, which is a biased but consistent technique (and is also the one we use).

And finally the null scattering method which is an unbiased but stochastic solution.

So let me describe each case.

Analytic / Biased Ray Marching

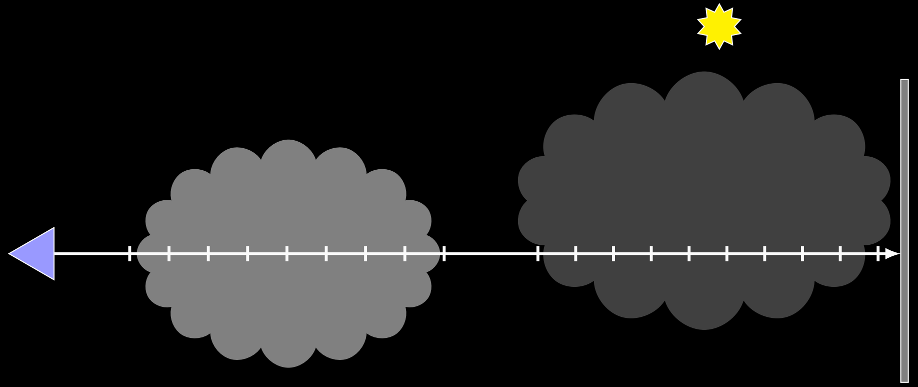


The analytic and biased techniques are actually fairly similar so we'll start with those.

I'll assume that we've already figured out where the ray intersects the volumes (I'll be talking more about this in a minute).

Analytic / Biased Ray Marching

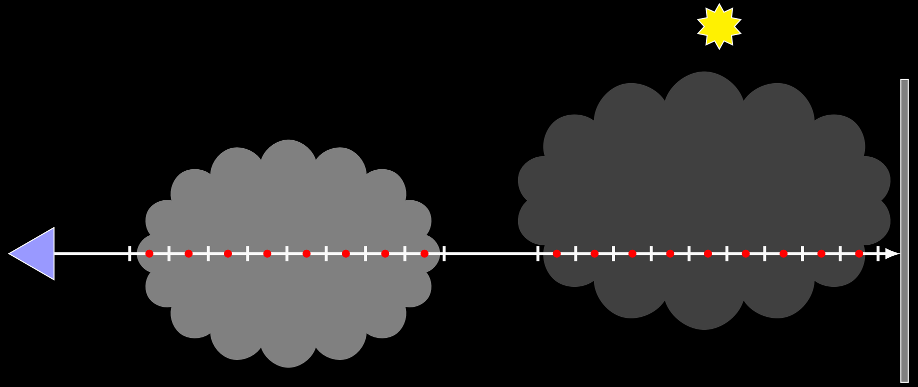
Split ray into segments



Then we split the ray into little segments. Either by exactly intersecting the voxels, or by just taking uniform steps.

Analytic / Biased Ray Marching

Run shader once per segment (front to back)

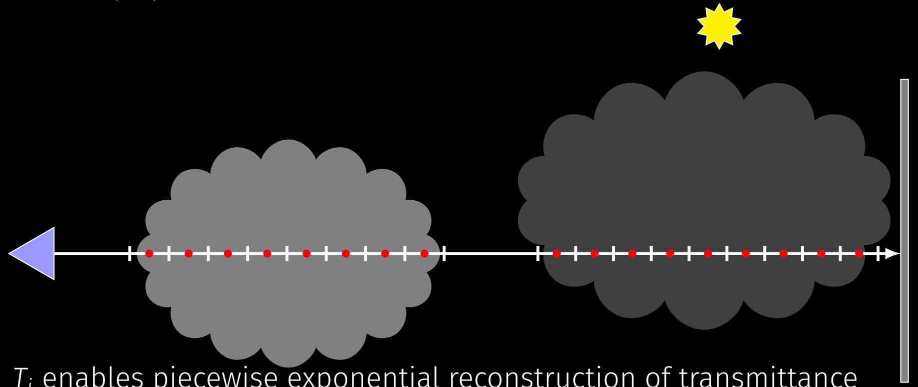


Then we can run the shader in each step (or for the analytic case you use the voxel corners to exactly add up the density).

Of course we do this from front to back so we can stop early if we accumulate enough opacity.

Analytic / Biased Ray Marching

Store μ_{s_i},μ_{t_i},$T_i = T_{i-1}e^{-\mu_{t_{i-1}}\Delta_{i-1}}$ and phase function



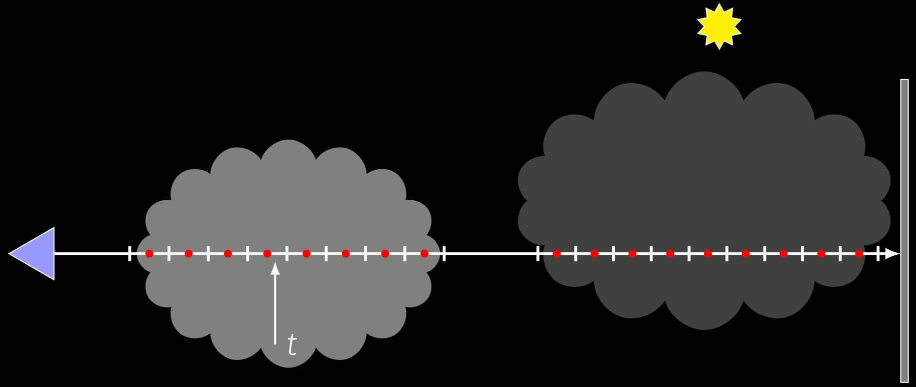
T_i enables piecewise exponential reconstruction of transmittance

If we store the density information about each step, we can build a piecewise representation of the volume that lets us compute the transmittance for any point along the ray.

In the biased case we assume we have small homogeneous segments. In the analytic case we have small gradients of density depending on which interpolation kernel we want.

Analytic / Biased Ray Marching

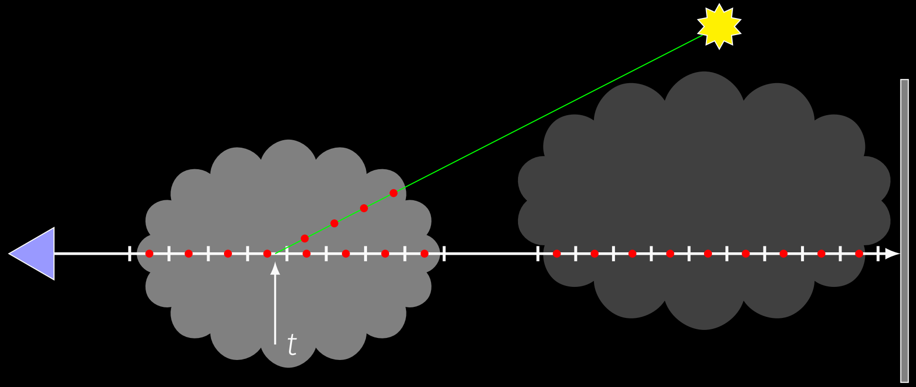
Given any *t*, locate segment by binary search



Once we have this table, we can quickly evaluate the transmittance anywhere we want.

Analytic / Biased Ray Marching

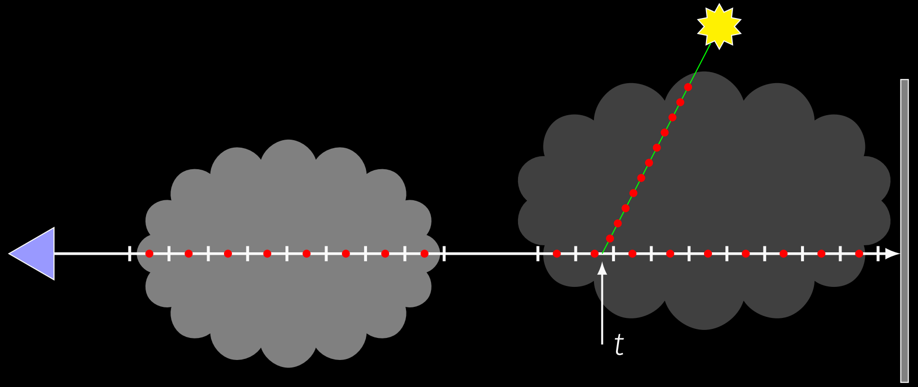
Can calculate lighting at any point (using any pdf)



That means we are free to use whatever pdf we want as well.

Analytic / Biased Ray Marching

$\mathrm{pdf}(t) \propto \mu_s(t)\, T(t)$ works well (+ MIS with equiangular for NEE)

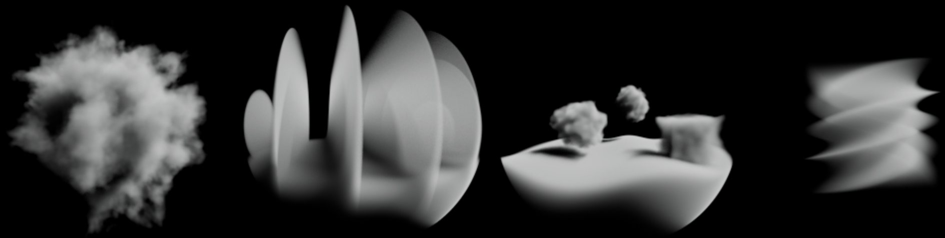


In fact, since we already built a table for the transmittance, we might as well tabulate a pdf at the same time. Making it proportional to scattering times transmittance works well.

And for next event estimation you can combine this with equiangular sampling through MIS.

Or if you want to be even more accurate, you can tabulate information about the lights directly into the table and get joint importance sampling. Of course this is slower, so in our renderer we just use the MIS approach.

Biased Ray Marching - Step Size



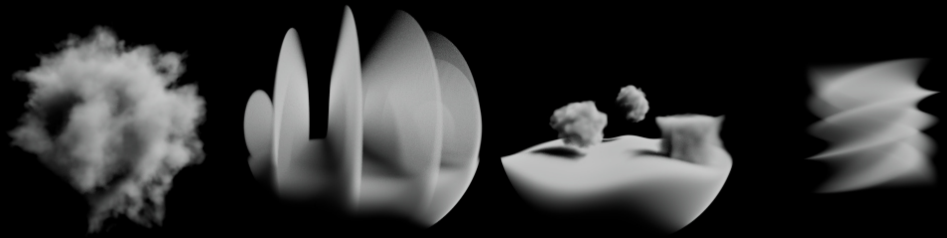
Step Size = 1×

But let me just concentrate on the ray marching itself.

If you did your ray marching very carefully through the voxels, you can make this method exact. If you don't have voxels and just assumed each little segment was homogeneous you have a bit of bias.

Let's see what happens when we increase the step size on these 4 procedural scenes...

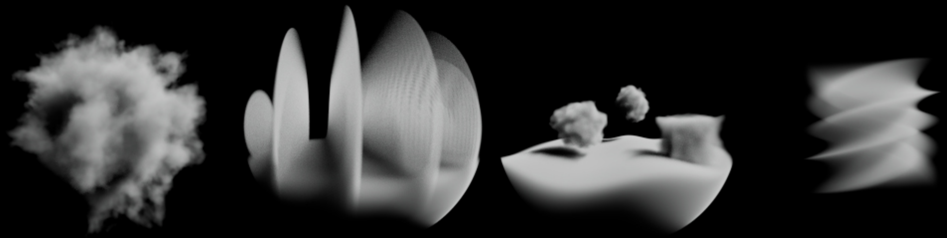
Biased Ray Marching - Step Size



Step Size = 2×

…doubling the step-size…

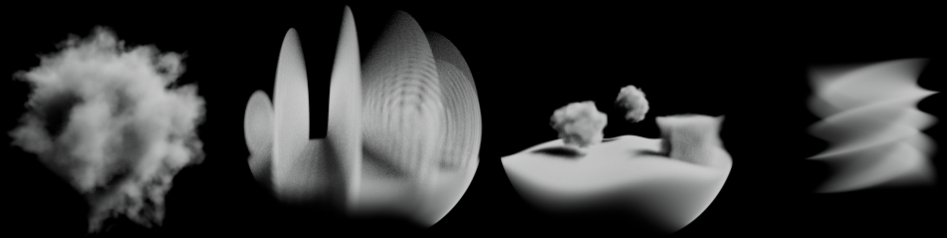
Biased Ray Marching - Step Size



Step Size = 4×

…4 times…The second volume is starting to show some artifacts

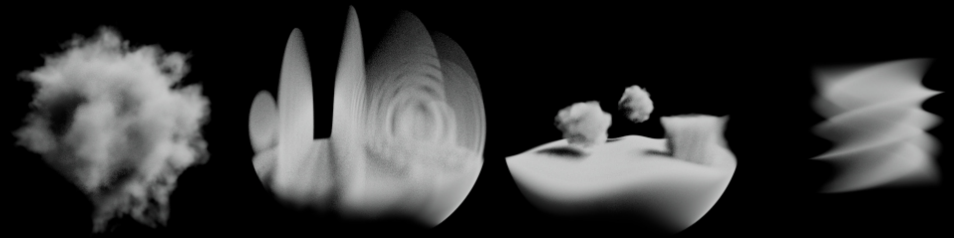
Biased Ray Marching - Step Size



Step Size = 8×

...8 times...

Biased Ray Marching - Step Size



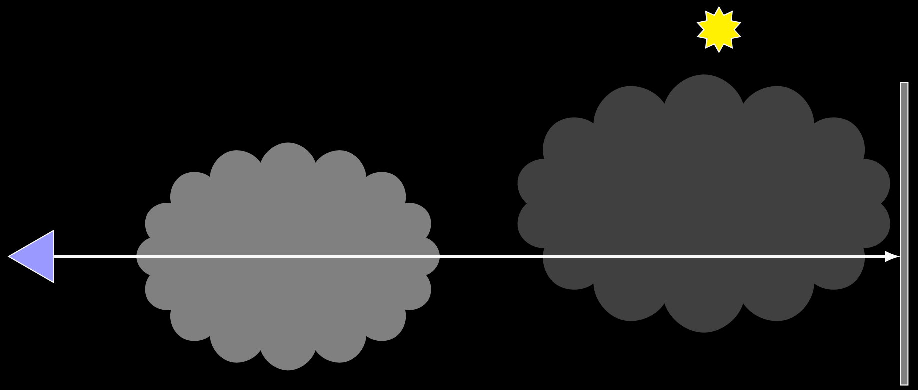
Step Size = 16×

Speed ⇔ Bias
Faster renders without extra variance, but slight loss in density

At 16 times the original step size we definitely see some issues on the volume with thin features, but the others still look reasonable even if they aren't correct.

Keep in mind that these renders were 16 times faster. So we are just trading speed for bias, but not extra variance which is a nice property.

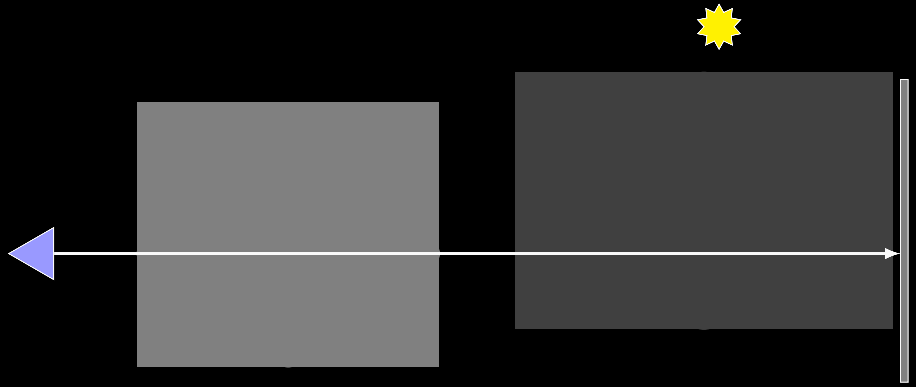
Null Scattering [Miller et al., 2019]



Now let’s look at null scattering methods. Hopefully you had a chance to catch the paper presented yesterday in the volume session, but I’ll try to cover the main idea again quickly.

Null Scattering [Miller et al., 2019]

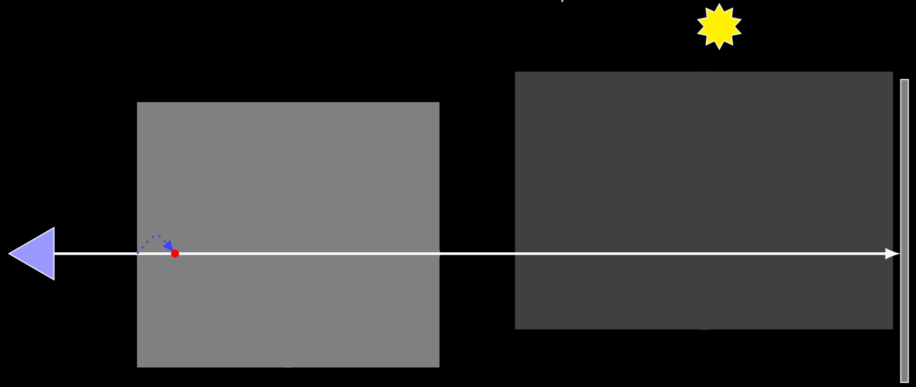
Increase the density to $\bar{\mu} = \max \mu_t(\mathrm{x})$



First we increase the density of each volume to be equal to its maximum. This turns every heterogeneous volume into a homogeneous one.

Null Scattering [Miller et al., 2019]

Sample with $\bar{\mu}e^{-\bar{\mu}t}$, average step length is $\frac{1}{\bar{\mu}}$

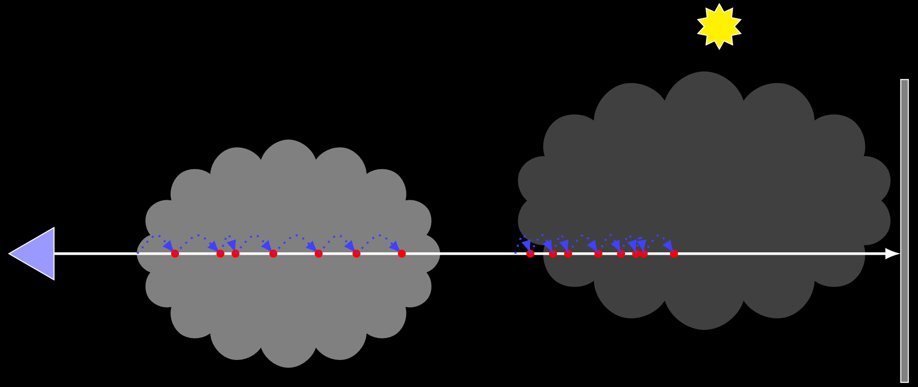


This makes the transmittance easy to sample again! We can sample from this transmittance and get candidate points inside the volume.

The steps are going to be random, but on average they'll be 1 / density. In other words, the more dense the volume is, the smaller steps we'll need to take.

Null Scattering [Miller et al., 2019]

Restore energy with $\mu_n = \bar{\mu} - \mu_t$ forward scattering

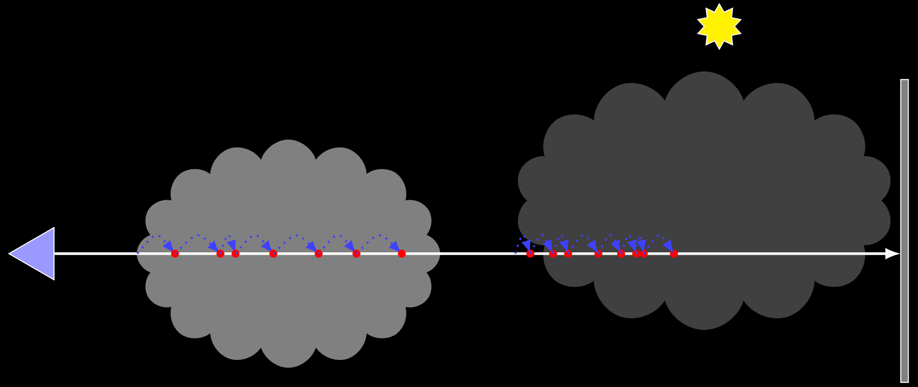


Of course we want a picture of clouds, not giant boxes. To get back to the picture of the volume we actually want, we turn the particles that filled up our volume into *null scattering* particles. We call it null scattering because it just scatters all the light forward as if it wasn't there at all.

So basically we gave up on the complicated expression for transmittance and just replaced it with a multiple scattering problem instead. But since the null scattering event is really simple, we just end up with a form of stochastic ray marching.

Null Scattering [Miller et al., 2019]

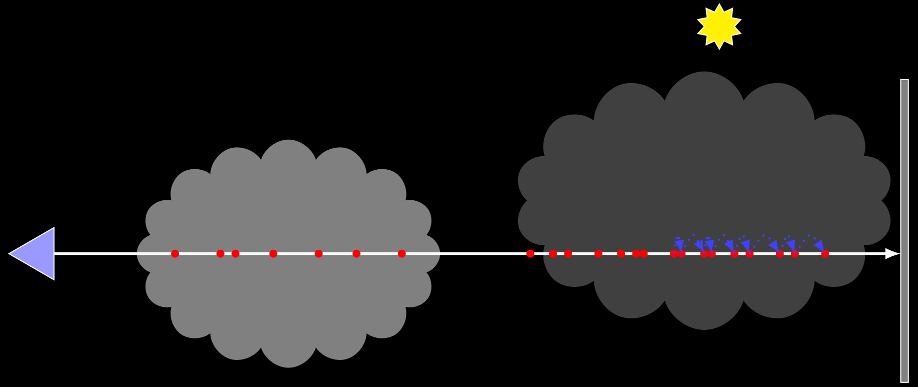
Random walk stops if $\mu_t = \bar{\mu}$ because $\mu_n = 0$



This random walk down the ray will terminate if we ever sample a density exactly equal to the maximum because we wouldn't have any null particles there to scatter us forward. This makes the ray marching fast, but can be a problem in thin volumes.

Null Scattering [Miller et al., 2019]

Increasing $\bar{\mu}$ increases chance of visiting the whole ray

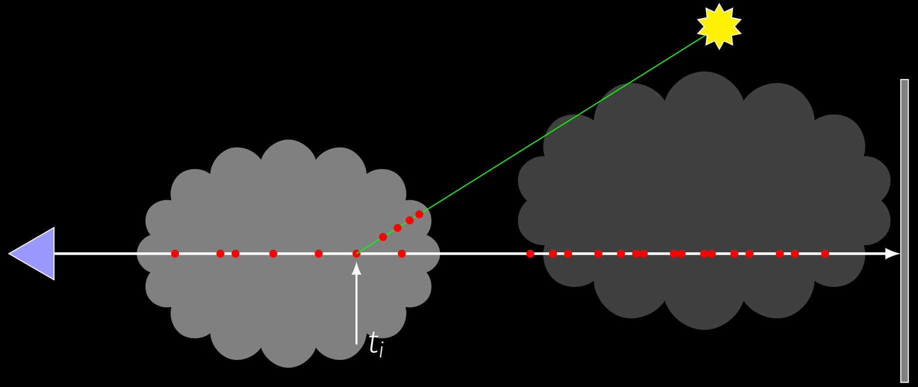


What you can do is force the ray marching to take extra steps by increasing the density bound.

This helps a lot with thin volumes or if there is bright emission either inside or behind the volume.

Null Scattering [Miller et al., 2019]

Continue path from any sample (choose $\propto \mu_{s_i} T_i$)



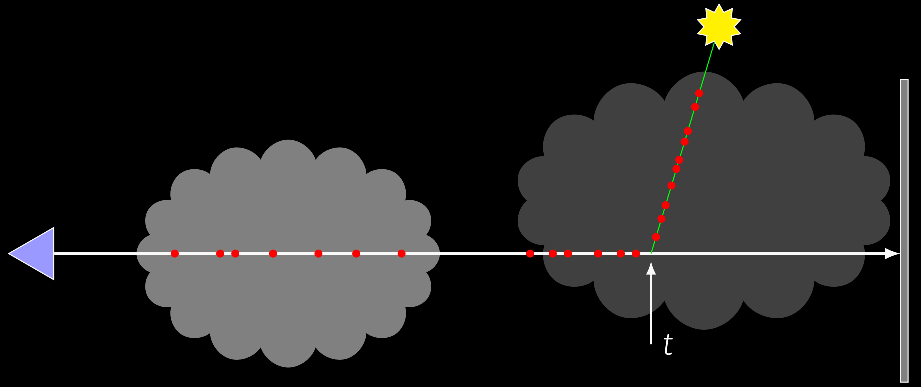
And once we're done with the random walk, we have a list of interaction points with the volume that we can choose from to continue the path.

Just like before, we can make this choice proportionally to scattering times transmittance.

You could also decide to stop marching earlier by randomly choosing real scattering over null scattering, but that also leads to higher variance for the same reason that using a tight density bound adds variance - you'd be stopping early and ignoring anything beyond that point.

Null Scattering [Miller et al., 2019]

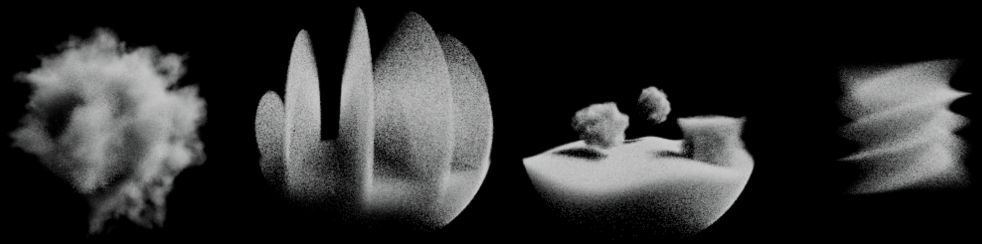
Mix with arbitrary pdfs as well!



And like you heard the presentation yesterday - its ok to mix other techniques like equiangular sampling because we now have a way of thinking about the pdf for the entire path.

When you pick an arbitrary distance *t* along the ray, you just need to remember to account for the null scattering events that lead up to that point.

Null Scattering - Step Size (16 rays/pixel)



Exact $\bar{\mu}$ is noisy for thin media

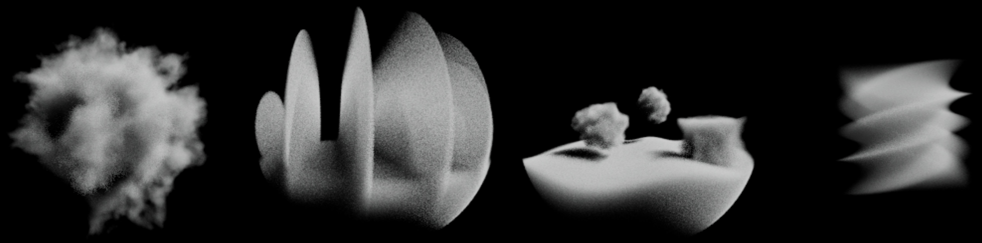
The great thing about this approach is that we can be confident that we'll eventually get the right answer, as long as our density bound is correct.

The downside is that we've introduced a lot more stochastic decisions which has added lots of noise.

Here are the same 4 procedural scenes as before. When we use the exact bounding density, the picture is fairly noisy because of that random early termination.

But on the plus side, these renders were much faster to compute.

Null Scattering - Step Size (16 rays/pixel)

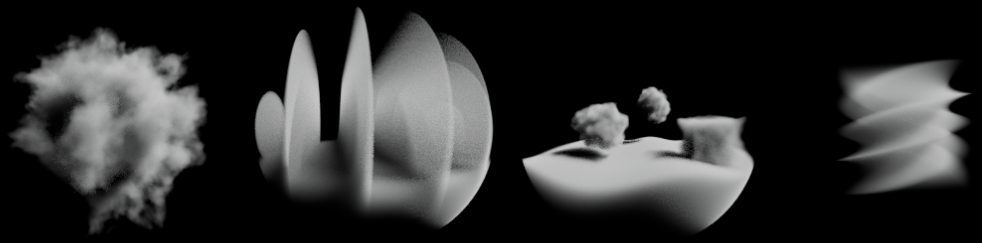


Relaxing to $\max(\bar{\mu}, \frac{1}{\text{stepsize}})$ improves quality

We can converge a bit faster by artificially increasing the bounding density to a value related to the step size we used before. The number of ray marching steps is now roughly equal to the biased method, which reduces the variance in those thin regions.

Of course the volumes that were fairly dense don't show as much improvement because the step size was already small.

Null Scattering - Step Size (16 rays/pixel)



Ray Marching produces less noise for an equal step count

VANCOUVER, BC | CULVER CITY, CA

But even though we can improve quality by forcing more steps, ray marching generally gives a lot less noise because its making fewer random decisions.

I'm showing results with a tiny number of rays here so that you can hopefully see the difference in the slides, but the difference is still visible even after taking hundreds of samples.

Decomposition Tracking [Kutz et al., 2017]

Sampling overlapping densities μ_A and μ_B can be done independently:

- Sample $t_A \propto T(t, \mu_A)$
- Sample $t_B \propto T(t, \mu_B)$
- $\min(t_A, t_B) \iff t_{A+B} \propto T(t, \mu_{A+B})$

Another important technique to know about for sampling transmittance is decomposition tracking.

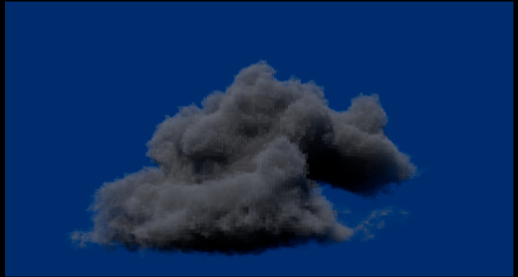
Transmittance has the nice property that sampling a random distance from two mediums independently and then taking the minimum gives the same distribution as sampling transmittance from the combined medium.

So if we somehow had a cheap medium *A* that overlapped with a more expensive medium *B*, we can limit the amount of work we spend on *B* by sampling *A* first. I'll refer you to the paper by Kutz et al from 2017 for all the details, but this can be used to speed up null scattering methods even more by doing fewer lookups in dense regions.

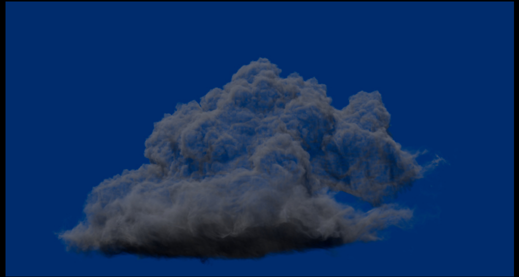
Decomposition Tracking [Kutz et al., 2017]

Just to illustrate the idea, we can take the Disney cloud dataset, and decompose it…

Decomposition Tracking [Kutz et al., 2017]


Sparse Control Volume

+


Thin Residual Volume

Into one sparse volume that is dense but has only coarse voxels…

…and a residual volume that has all the outer details. Of course in practice you don't really make two volumes, you just store the minimum density for every block of voxels together with the maximum density.

Decomposition Tracking [Kutz et al., 2017]

Fewer lookups $\Longrightarrow$ Deeper paths

This is one of the techniques to allow the null scattering methods to scale to deeper paths.

But actually you can do something very similar with ray marching methods which is how this picture was rendered. If the inside of the volume is flagged as a dense block, we can take a single large step there and save a similar amount of work.

But decomposition tracking is slightly more general since it also works in regions that aren't completely uniform.

Ray Marching Comparison

Analytic	Ray Marching	Null Scattering

So I showed three different methods, which one should you use?

Ray Marching Comparison

Analytic	Ray Marching	Null Scattering
✓ Exact	✗ Biased	✓ Unbiased

Ray marching's main weakness is that it is biased. Most of the time this bias isn't visually objectionable, but changing the appearance might not be acceptable if you are just trying to speedup a render.

Ray Marching Comparison

Analytic	Ray Marching	Null Scattering
✓ Exact	✗ Biased	✓ Unbiased
✓ Parameter Free	✗ Step Size	✗ $\bar{\mu}$

Ray marching and null scattering both need an extra parameter in practice. Of course both step size and density bounds can be automated, but it is something extra that you have to worry about.

If your volume is fully procedural, usually figuring out the maximum density is easier than figuring out the scale of the smallest detail, which is also worth keeping in mind.

Ray Marching Comparison

Analytic	Ray Marching	Null Scattering
✓ Exact	✗ Biased	✓ Unbiased
✓ Parameter Free	✗ Step Size	✗ $\bar{\mu}$
✓ 1 voxel/step	✓ $\sim$ 1 voxel/step	✗ $<$ 1 voxel/step
✓ No rand. calls	✓ 2 rand. calls	✗ $O(n)$ rand. calls

Analytic methods visit each voxel exactly once without needing any random numbers.

Ray marching just needs two random numbers to get a stratified offset and jitter the number of steps. Then it can take roughly one step per voxel.

But with null scattering, the steps are completely random which means we might visit the same voxel many times or skip over several voxels at once. This makes the data access less coherent. If the medium is very dense, we might be forced to take very tiny steps along the ray.

We also need to take a new random number in every step which adds some overhead.

Ray Marching Comparison

Analytic	Ray Marching	Null Scattering
✓ Exact	✗ Biased	✓ Unbiased
✓ Parameter Free	✗ Step Size	✗ $\bar{\mu}$
✓ 1 voxel/step	✓ $\sim$ 1 voxel/step	✗ $<$ 1 voxel/step
✓ No rand. calls	✓ 2 rand. calls	✗ $O(n)$ rand. calls
✗ Speed fixed	✓ Speed $\leftrightarrow$ Bias	✓ Speed $\leftrightarrow$ Variance

With analytic methods execution speed is pre-determined by the resolution of the data. This means if you want your render to go faster, you will have to somehow reduce the number of voxels. The TOG paper on the Manuka renderer describes a data structure that changes resolution with distance to camera for instance.

We already talked about the speed/bias tradeoff for ray marching. The speed/variance tradeoff for null scattering is harder to analyse since the extra variance will usually have to be compensated some other way. So its a bit harder to quantify the benefit in isolation.

Ray Marching Comparison

Analytic	Ray Marching	Null Scattering
✓ Exact	✗ Biased	✓ Unbiased
✓ Parameter Free	✗ Step Size	✗ $\bar{\mu}$
✓ 1 voxel/step	✓ $\sim$ 1 voxel/step	✗ $<$ 1 voxel/step
✓ No rand. calls	✓ 2 rand. calls	✗ $O(n)$ rand. calls
✗ Speed fixed	✓ Speed $\leftrightarrow$ Bias	✓ Speed $\leftrightarrow$ Variance
✗ Voxels only	✓ Procedural	✓ Procedural

Analytic methods also have the drawback of only working for regular voxel data.

On the other hand ray marching and null scattering work even for procedural densities.

Ray Marching Comparison

Analytic	Ray Marching	Null Scattering
✓ Exact	✗ Biased	✓ Unbiased
✓ Parameter Free	✗ Step Size	✗ $\bar{\mu}$
✓ 1 voxel/step	✓ $\sim$ 1 voxel/step	✗ $<$ 1 voxel/step
✓ No rand. calls	✓ 2 rand. calls	✗ $O(n)$ rand. calls
✗ Speed fixed	✓ Speed $\leftrightarrow$ Bias	✓ Speed $\leftrightarrow$ Variance
✗ Voxels only	✓ Procedural	✓ Procedural

Optimal method still active area of research

This isn't meant to be an exhaustive list, but these are some of the pros and cons of the various methods.

The paper presented yesterday on the path integral formulation of null scattering has definitely opened up a new way of thinking about the null scattering techniques that suggests lots of new ways to tackle the problem.

- Intersecting Volumes
    - Meshes
    - Sparse Regular Grids
    - Sparse Frustum Grids
    - Motion Blur
- Overlap Handling
    - Volume Primitives
    - Surface Defined Volumes

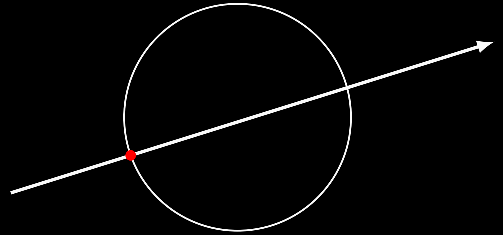
Now that we've covered the basics of sampling, let me switch gears a bit and talk about some other implementation aspects of volumes.

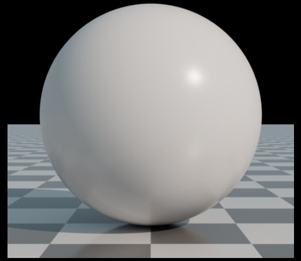
We talked about marching along the ray, but not about how we come up with the intervals that we do this marching between.

Related to this I'll cover overlap handling in its various forms.

Intersecting Volumes - Shapes

Surface tracing only needs nearest hit



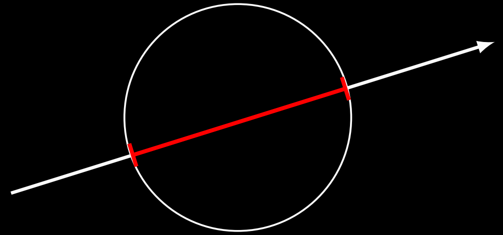


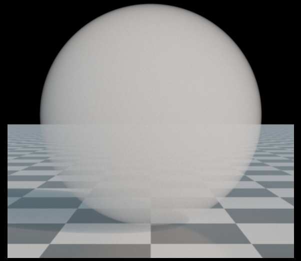
```
struct srfhit {
 float t;
 int geomID;
};
```

When ray tracing surfaces, Intersections are easy to represent: just a distance along the ray and a primitive identifier.

Intersecting Volumes - Shapes

Volume tracing needs hit *interval*





```
struct volhit {
 float t[2];
 int geomID;
};
```

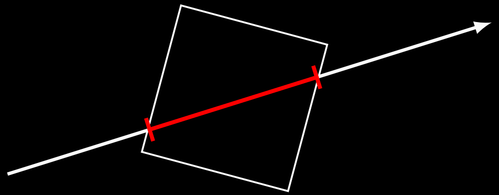
When rendering volumes, we want to know a range of distances that corresponds to the overlap between the shape and the ray segment.

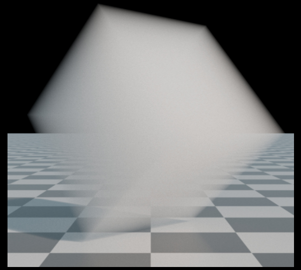
This is still simple to represent, we just have a pair of distances now instead of just one.

This is easy for convex shapes like a sphere…

Intersecting Volumes - Shapes

Easy for convex shapes



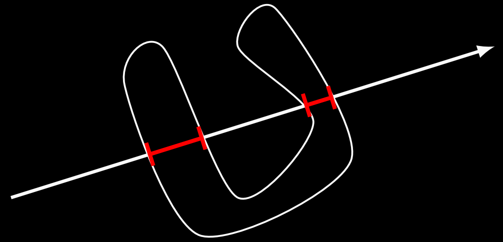


```
struct volhit {
 float t[2];
 int geomID;
};
```

…or a box, because the intersection test always produces a front and back hit.

Intersecting Volumes - Meshes

Concave shapes may need several hit intervals



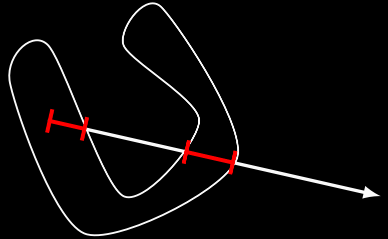
```
struct volhit {
 float t[2];
 int geomID;
 volhit* next;
};
```

For meshes, things get more interesting. One approach is to use the surface intersection routine to build a list of *all* possible hits along the ray and pair them up to get the intervals.

And because we might have more than one interval, we'll need to keep a *list* of intervals.

Intersecting Volumes - Meshes

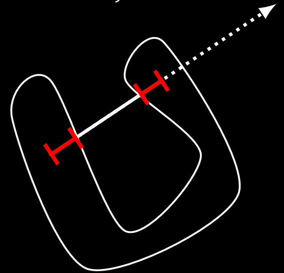
Count hits to determine overlap



```
struct volhit {
 float t[2];
 int geomID;
 volhit* next;
};
```

If the ray starts inside the shape, the number of hits will be odd and you have to pair the first hit with the origin.

Intersecting Volumes - Meshes



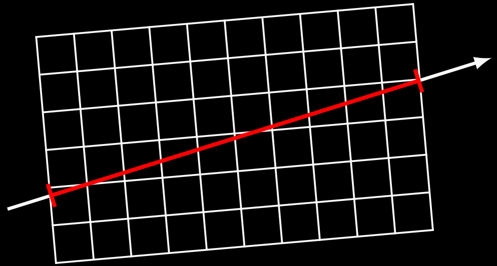
```
struct volhit {
 float t[2];
 int geomID;
 volhit* next;
};
```

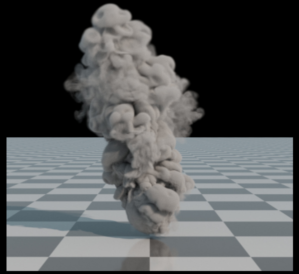
If the ray starts and ends inside the mesh - you need to make sure you still count hits using the *infinite* ray to resolve the intervals correctly.

The good thing about the intersection counting approach is that it figures out the inside/outside status of each ray from scratch, so there's no need to maintain a stack.

Intersecting Volumes - Sparse Grids

Intersect bounds for voxel data





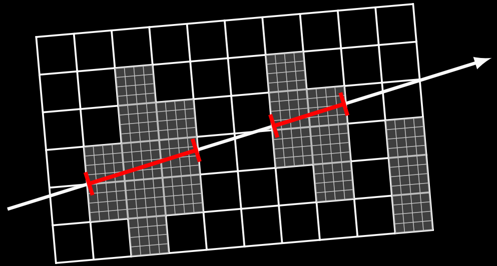
```
struct volhit {
 float t[2];
 int geomID;
 volhit* next;
};
```

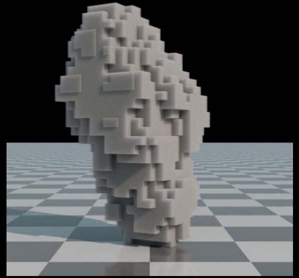
Of course the most common type of volume primitive is the voxel grid.

You could just intersect the bounding box of the the grid, but this is not efficient because large portions might not contain any data at all.

Intersecting Volumes - Sparse Grids

Exploit sparse structure to skip empty space





```
struct volhit {
 float t[2];
 int geomID;
 volhit* next;
};
```

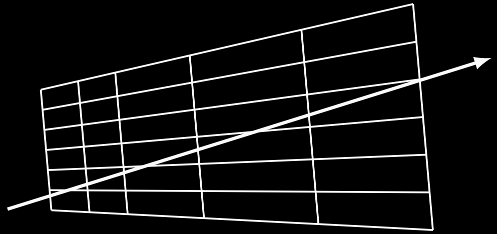
Voxel grids are typically stored as *sparse*, two level grids to save memory.

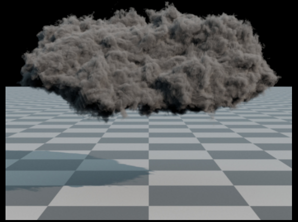
We can take advantage of this to also get shorter volume hit intervals by traversing the coarse portion of the grid and keeping track of when we cross into a region with data defined.

The OpenVDB format is an even fancier version of this with more than two levels, but the open source library implement this traversal function for you.

Intersecting Volumes - Sparse Frustum Grids

Perspective improves detail along x, y



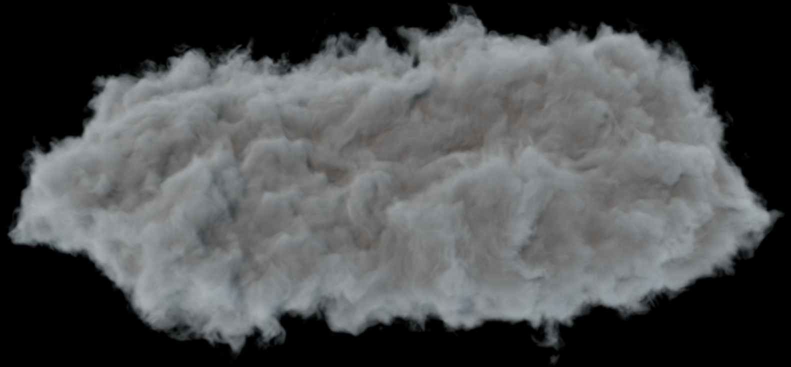


```
struct volhit {
 float t[2];
 int geomID;
 volhit* next;
};
```

Finally, we have frustum aligned sparse grids. This is when we warp our voxel data to match the camera frustum to increase resolution where it matters more.

Here finding the intersection is a bit more involved, so I'll discuss it in a bit more detail.

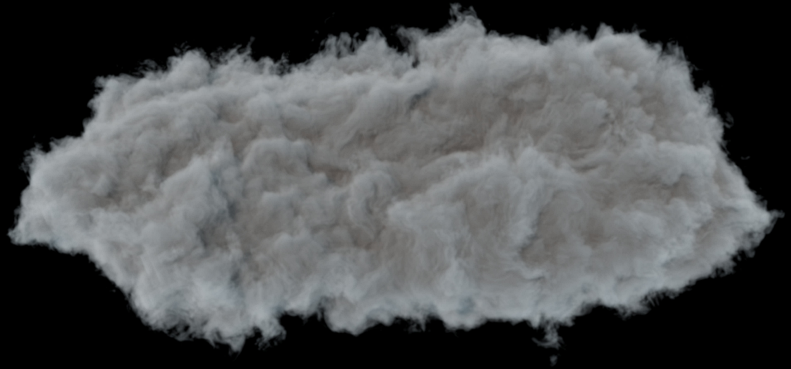
Intersecting Volumes - Sparse Frustum Grids



Regular Grid - 954 × 398 × 960 - 540Mb

First just to show an example, this volume is about 500Mb stored as a regular grid

Intersecting Volumes - Sparse Frustum Grids



Frustum Grid - 874 × 409 × 85 - 52Mb

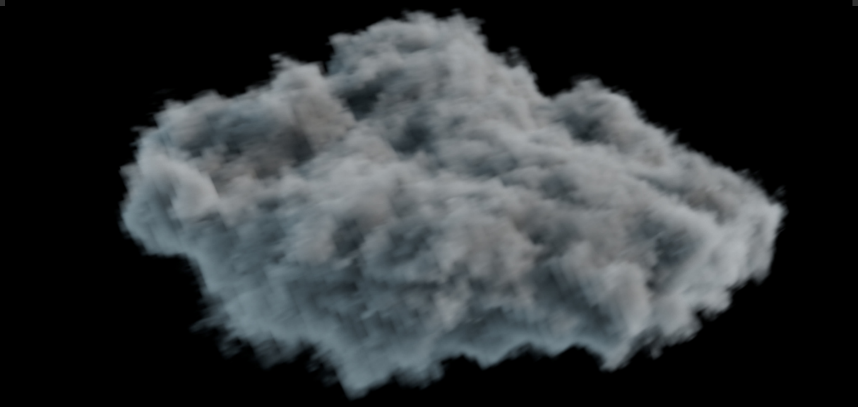
And only 52Mb as a frustum grid.

Intersecting Volumes - Sparse Frustum Grids

Regular Grid - $954 \times 398 \times 960$ - 540Mb

Only if we looked at the volume from the side would we see that the volume…

Intersecting Volumes - Sparse Frustum Grids

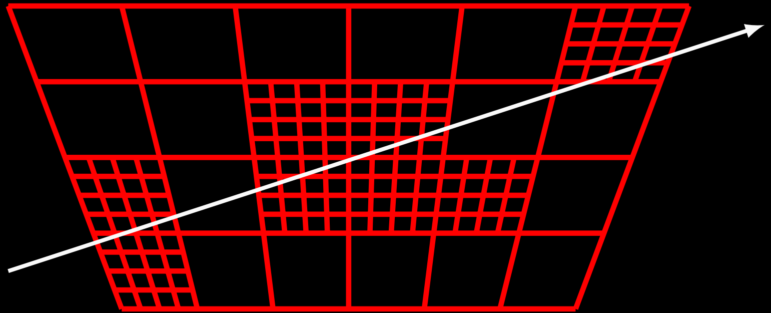


Frustum Grid - 874 × 409 × 85 - 52Mb

…is actually much lower resolution, but this isn’t visible at all from the camera.

Ray intervals for Sparse Frustum Grids

How to extend the traversal algorithm to warped grids?



So frustum grids are a great way to save memory, especially for animated volumes.

But now its not so easy to walk through the grid anymore, because the planes of the grid aren’t parallel.

Undoing the mapping isn’t simple either because the perspective division is going to affect the distribution of distances along the ray.

Ray intervals for Sparse Frustum Grids [Wrenninge et al., 2013]

Approach:

- Build kd-tree in grid space [Yue et al., 2010]
- Transform planes to world space during traversal
- Supports motion blurred transforms

So rather than try to figure out exactly how to account for the distortion of the mapping, we instead decided to build a kd-tree in grid space, and then map it to world space during traversal.

In other words, rather than trying to move the ray into the space of the acceleration structure, we move the acceleration structure into the space of the ray.

This works even the mapping is animated because we do the transformation on the fly during traversal.

Ray intervals for Sparse Frustum Grids [Wrenninge et al., 2013]

Axis-aligned plane equations have a simple form:

$$x = \mathrm{p}_x \Rightarrow \begin{pmatrix} x & y & z & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \\ -\mathrm{p}_x \end{pmatrix} = 0$$

(similar equations for Y and Z)

Just to explain how this works, you can represent any plane as a column vector like this. If its axis aligned it will have this very simple form.

Ray intervals for Sparse Frustum Grids [Wrenninge et al., 2013]

Transformed plane can be directly extracted from the frustum matrix:

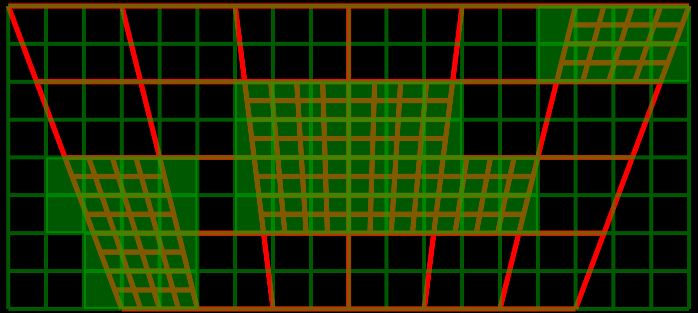
$$\begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \\ -\mathrm{p}_x \end{pmatrix} = \begin{pmatrix} m_{11} - \mathrm{p}_x \cdot m_{14} \\ m_{21} - \mathrm{p}_x \cdot m_{24} \\ m_{31} - \mathrm{p}_x \cdot m_{34} \\ m_{41} - \mathrm{p}_x \cdot m_{44} \end{pmatrix}$$

We can transform that column vector by the matrix, and since it has a simple form this boils down to a simplified expression that just picks up certain matrix entries.

And that’s it! During traversal we use that transformed plane and traverse the kd-tree in the normal way.

Ray intervals for Sparse Frustum Grids [Museth, 2014]

Alternative approach: Overlay a regular grid

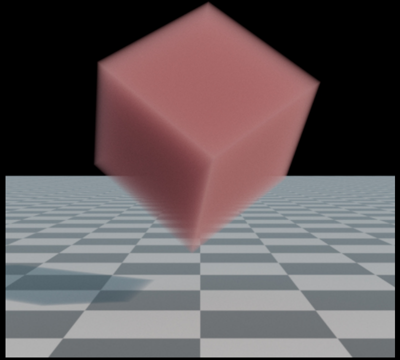


So that was our approach, but I'll quickly mention another technique that used a similar trick only using a regular grid instead of a kd-tree.

This is not quite as accurate though, because here you need to resample onto a regular structure. So you might need extra resolution to capture the edges and you might have to be extra conservative if the mapping is animated.

Motion Blur

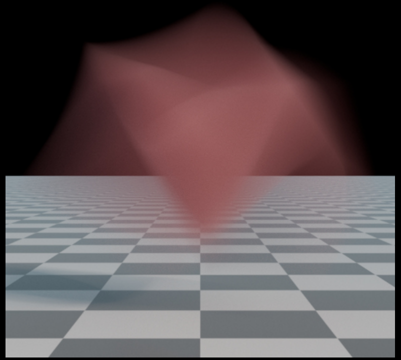
- Transformation Blur



Now let me say a few words about motion blur. This is obviously an important ingredient of production rendering.

Motion Blur

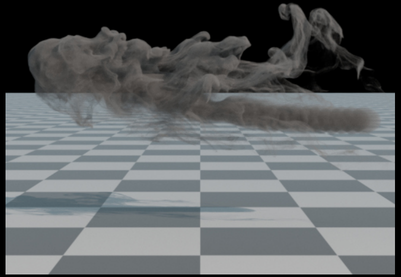
- Transformation Blur



Transformation motion blur when only the object transform is changing is easy to deal with and mostly comes for free if your volume primitives are part of the same acceleration structure as surfaces.

Motion Blur

- Transformation Blur
- Deformation Blur
    - Velocity Blur

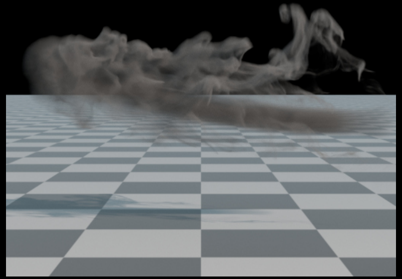


$$\mu(\mathbf{x}, t) \approx \mu(\mathbf{x} - t\,\mathbf{v}(\mathbf{x}))$$

Deformation blur on voxel data is a bit harder.

Motion Blur

- Transformation Blur
- Deformation Blur
    - Velocity Blur



$$\mu(\mathrm{x}, t) \approx \mu(\mathrm{x} - t\,\mathrm{v}(\mathrm{x}))$$

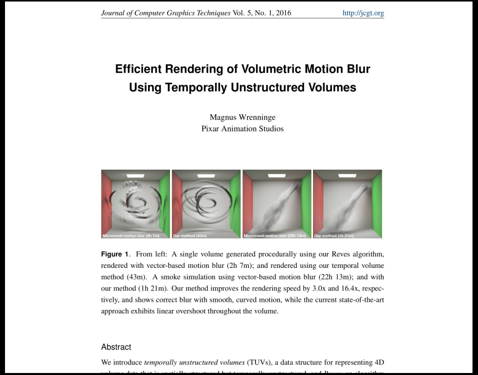
One approach is to use a velocity field to blur the density like I'm showing here. This is similar to what fluid simulators do internally. But this is just a first order approximation.

For example the source of the fluid on the right there is blurred even though it should be stationary.

On top of this this we have to do extra ray marching with this approach because empty regions might have density pulled into them by the velocity. So we have to dilate the grid of active blocks and that makes things slower.

Motion Blur

- Transformation Blur
- Deformation Blur
  - Velocity Blur
  - Temporal Volumes (4D)



Microvoxel motion blur (2h 7m)
Our method (43m)
Microvoxel motion blur (22h 13m)
Our method (1h 21m)

Journal of Computer Graphics Techniques Vol. 5, No. 1, 2016 http://jcgt.org

Efficient Rendering of Volumetric Motion Blur Using Temporally Unstructured Volumes

Magnus Wrenninge
Pixar Animation Studios

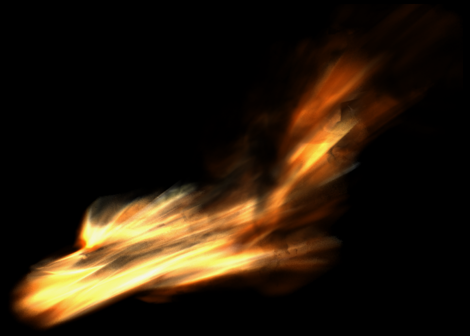
Figure 1. From left: A single volume generated procedurally using our Reves algorithm, rendered with vector-based motion blur (2h 7m); and rendered using our temporal volume method (43m). A smoke simulation using vector-based motion blur (22h 13m); and with our method (1h 21m). Our method improves the rendering speed by 3.0x and 16.4x, respectively, and shows correct blur with smooth, curved motion, while the current state-of-the-art approach exhibits linear overshoot throughout the volume.

Abstract

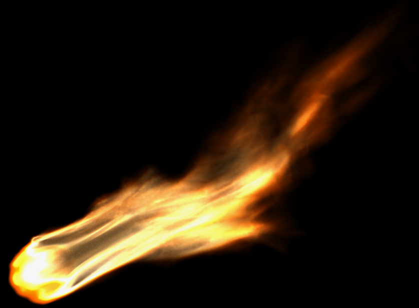
We introduce *temporally unstructured volumes* (TUVs), a data structure for representing 4D

A better solution is to actually use a 4 dimensional grid. This sounds like it would be a lot more data, but actually most voxels are either empty or slowly changing. So it compresses quite well.

Motion Blur - Temporal Volumes [Wrenninge, 2016]



Velocity Blur



Temporal Volumes

Here are some images I borrowed from the paper showing how a moving flame is properly blurred with this method.

Another benefit is that we don't need to pad the grids more than necessary for finding intervals.

And even though each voxel stores a bit more data, its all kept close together in memory, which is faster than doing a lookup for velocity followed by an offset lookup for density.

This approach is implemented in Field3D 2.0 and higher. The only real downside is that to create high quality temporal data, the compression needs to be interleaved into the fluid simulator.

Overlap Handling

- Volume Primitives (IOR $= 1$, no boundary)
    - Overlap is *additive*, all mediums are summed
    - Need list of overlapping intervals along the ray
- Surface Defined Volumes (IOR ≥ 1, with boundary)
    - Overlap is *exclusive*, one medium "wins"
    - Priority defined by the artist

Now that we know how to find where the volume along the ray, we need some rules to decide what to do when the intervals of different volume primitives overlap.

Our renderer distinguishes between two cases.

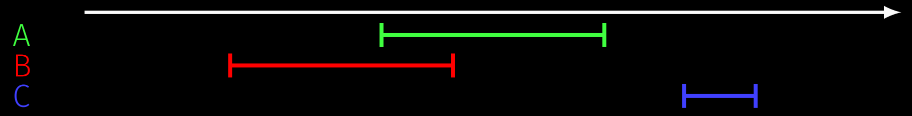
First we have the volume *primitives* I just mentioned. These don't have any boundary and the densities are additive.

Second, we have surface defined volumes. These are actually defined through surface shading, in other words we hit the boundary and the surface shader will tell us if there is a medium inside the shape or not. For these the overlap rules is exclusive and driven by a priority system.

Let me briefly describe each case.

Overlap Handling - Volume Primitives

Sort and split all segments into non-overlapping intervals.

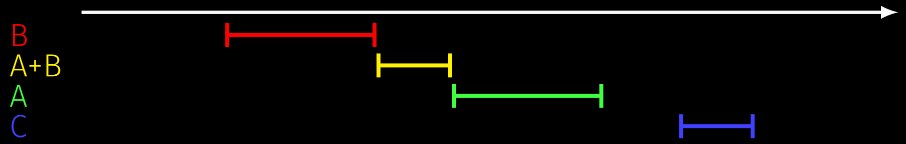


Like I mentioned before, the ray intersection returns a list of segments, and these might overlap.

When they do we need to split them into disjoint sets and sort them…

Overlap Handling - Volume Primitives

Sort and split all segments into non-overlapping intervals.
Each segment can refer to multiple primitives.

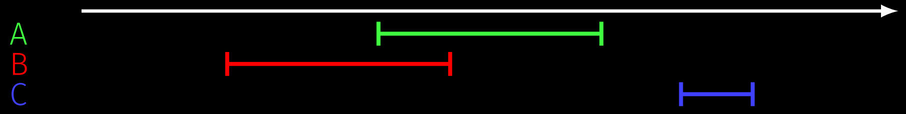


…like this. The segment that overlapped both A and B keeps a reference to both since we'll need to run both shaders if we ray march through that region.

Overlap Handling - Volume Primitives

Transmittance for shadow rays can be computed without splitting!

$T_{A+B+C} = T_A T_B T_C$



Also remember that you *only* need to do this rays along the primary path.

For shadow rays, you can ray march each segment independently and just multiply the results together so the sorting and splitting isn’t required.

Overlap Handling - Alternative Designs

- Aggregate Volumes [Fong et al., 2017]
  - Spatially partition volume primitives
  - ✓ Lower cost per ray
  - ✓ Acceleration structure can store $\bar{\mu}$ for null scattering
  - ✗ Higher upfront cost
  - ✗ Less accurate around edges

But there are other ways of dealing with overlap:

For example, instead of sorting and splitting each ray, you can partition the volume primitives spatially ahead of time. Then each region of space knows which volumes it contains and each ray can just walk through the structure and you don't even need those volume intersection tests at all.

You can store density bounds in the structure directly if you are using null scattering so ray marching and traversal happen together.

On the other hand, you do have a new kind of structure to build before rendering starts. And because the structure is axis aligned, volumes that aren't might lead to too much ray marching.

This was all described in the volume rendering course from 2 years ago, so be sure to check those course notes for all the details. Both approaches can be made to work in a production context, so its really down to implementation details as to which one is faster, especially if structure is also driving the ray marching scheme.

Overlap Handling - Alternative Designs

- Aggregate Volumes [Fong et al., 2017]
  - Spatially partition volume primitives
  - ✓ Lower cost per ray
  - ✓ Acceleration structure can store $\bar{\mu}$ for null scattering
  - ✗ Higher upfront cost
  - ✗ Less accurate around edges
- Decomposition Tracking [Kutz et al., 2017]
  - Samples transmittance stochastically

And like I mentioned earlier, decomposition tracking is also a way to handle overlapping volumes, but it does limit you to transmittance sampling only.

Medium Tracking - Surface Defined Volumes

- Surfaces can define an interior medium

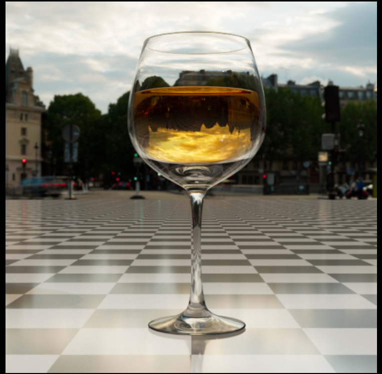
That was volume primitives with additive overlap, now lets talk about surface defined volumes.

This is the case where the surface shader is the one responsible for deciding what the medium properties are.

From the point of view of the artist, everything is contained in the surface shader. In this picture I just have a glass shader and a liquid shader.

Medium Tracking - Surface Defined Volumes

- Surfaces can define an interior medium
- Critical for correct rendering of liquids

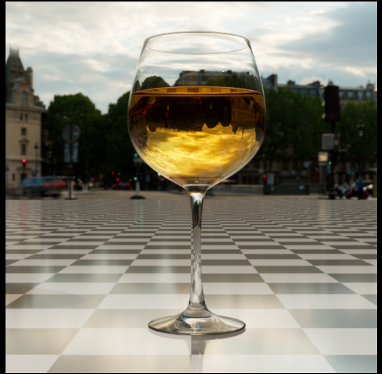


This is really critical for rendering liquids correctly.

Here I've shown how things look if you just model the liquid as slightly smaller than the glass. The air gap gets magnified and it looks like the liquid is floating inside the glass (which it actually is).

Medium Tracking - Surface Defined Volumes

- Surfaces can define an interior medium
- Critical for correct rendering of liquids
- Liquid is modeled slightly overlapping glass
- IOR computed from medium on each side



By modeling the liquid slightly overlapping the glass, we get the right picture because we get a clean transition from one medium directly to the next and also because we can get the right ratio of refractive indices.

Overlap Handling - Surface Defined Volumes

Schmidt et al., “Simple Nested Dielectrics in Ray Traced Images”, 2002

- Each ray maintains a stack of mediums it has entered
- At each interface, decide which medium “wins”
- Some hits will be discarded (but still update the stack)

Most renderers that support this, follow the 2002 paper from Schmidt and Budge. The basic idea is that each ray maintains the stack of all mediums its entered so far.

Each time we hit a surface, there are rules to decide which medium “wins” (based on their priority). This decides if the surface is actually visible or not.

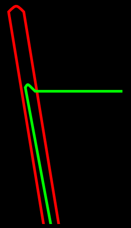
Either way, the stack is updated so we remember that we’ve entered or left the given medium.

Overlap Handling - Surface Defined Volumes

Schmidt et al., “Simple Nested Dielectrics in Ray Traced Images”, 2002

- Each ray maintains a stack of mediums it has entered
- At each interface, decide which medium “wins”
- Some hits will be discarded (but still update the stack)

🟥 Glass
🟩 Liquid



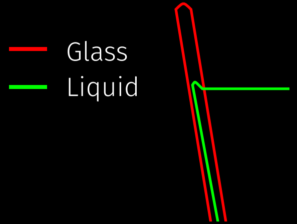
← Model

The example from before would looks like this…

Overlap Handling - Surface Defined Volumes

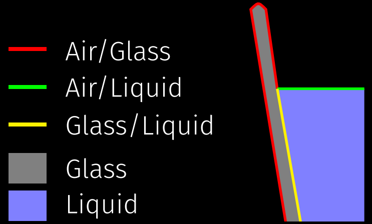
Schmidt et al., “Simple Nested Dielectrics in Ray Traced Images”, 2002

- Each ray maintains a stack of mediums it has entered
- At each interface, decide which medium “wins”
- Some hits will be discarded (but still update the stack)



← Model

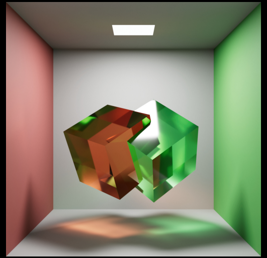
Render →

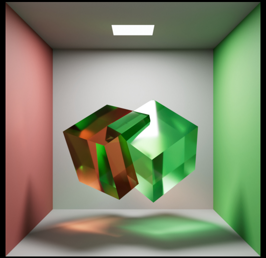


…and at render time it turns into this through these rules.

So the glass wins over the liquid where they overlap, but because we enter the liquid before leaving the glass we can get a clean transition from glass to liquid without having had to model it exactly which is super important for cases where the liquid might be animated.

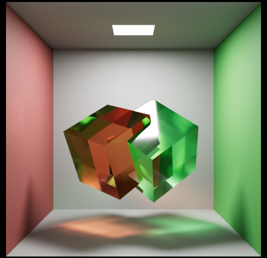
Overlap Handling - Medium Priority

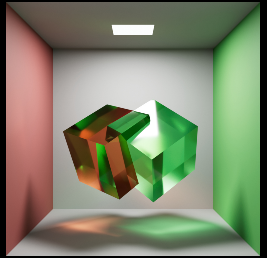

Left > Right

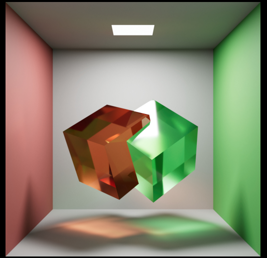

Left < Right

Just like in the paper, we give the artists control over which medium wins though a priority level.

Overlap Handling - Medium Priority

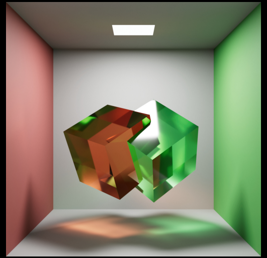

Left > Right

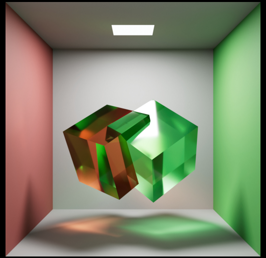

Left < Right

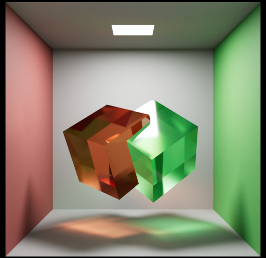

Left = Right

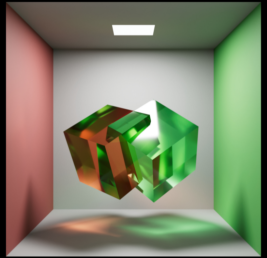
We also extended the rules a bit to cover the case where the priorities are equal to allow for merging interiors of surfaces. This can be handy when an object was modelled as multiple meshes but the artist wants them treated as a single volumetric interior.

Overlap Handling - Medium Priority


Left > Right


Left < Right

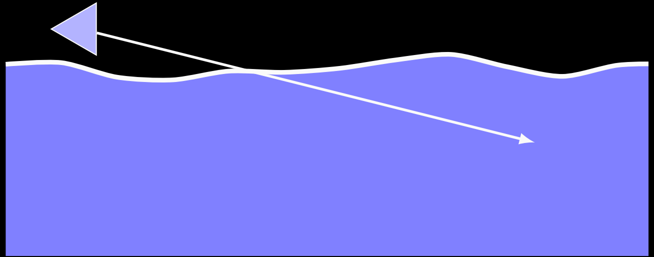

Left = Right


Left = Right = Off

We also added rules for a special *off* priority that acts as a fast default with a few shortcuts. In this case surface hits are always accepted which keeps things fast for the common case.

Medium Stack Initialization

Ray starting in vacuum behave correctly.

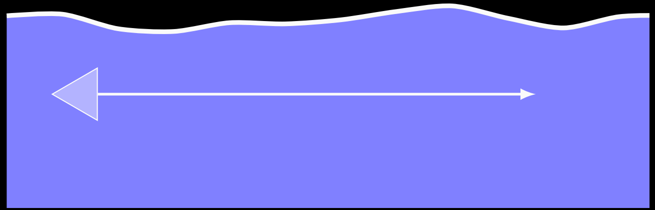


The last implementation detail to discuss is how we actually decide the starting state of the ray stack.

When the camera starts in empty space, everything is fine. An empty stack is the correct start state.

Medium Stack Initialization

Rays starting underwater need an initial stack

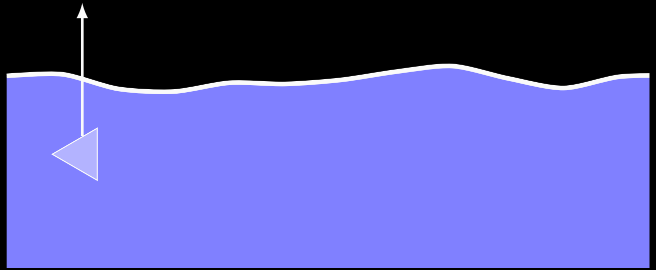


But if the camera start inside a medium (like underwater in this case) we need to have some way to specify the starting stack.

Notice how this first ray here doesn't even intersect the surface of the water all because it was modelled as just a single displaced plane.

Medium Stack Initialization

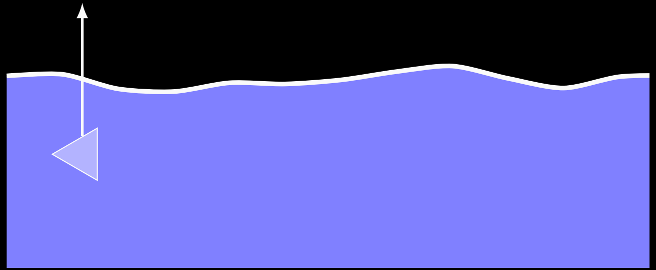
Trace a vertical probe ray to figure out initial stack



Our solution is to fire a single probe ray at the start of the render, going all the way through all surfaces to figure out the initial stack by counting hits.

Medium Stack Initialization

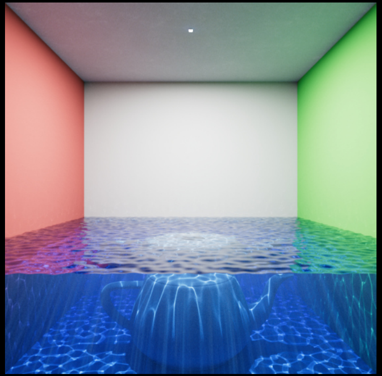
Water surface does not need to be closed

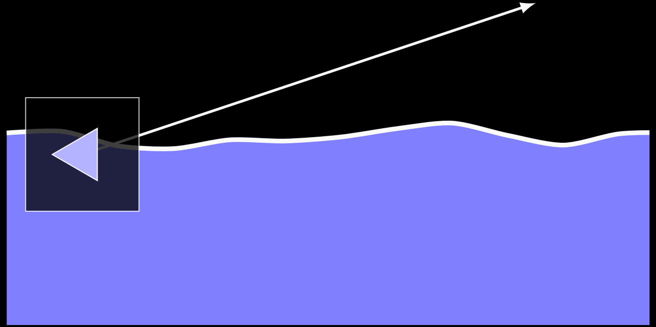


The direction you choose is arbitrary in principle, but we picked up because of this water case where the geometry might not actually be a closed volume.

Medium Stack Initialization

High priority empty medium can act as clipping plane



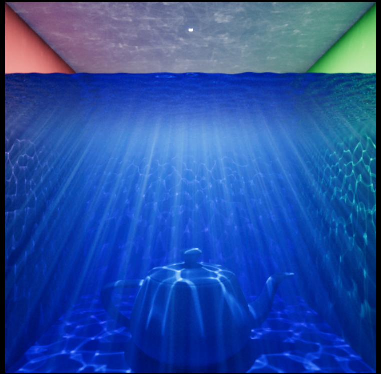


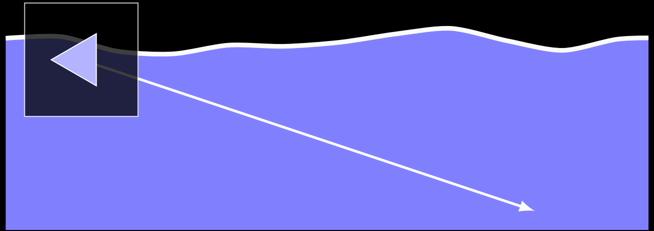
Another case we ran into was what to do when the camera is hovering near the surface and wants to see above and below the water at the same time.

We use a high priority box that acts like a clipping plane. This makes sure we can cross the water to find its medium without actually registering a hit right away.

Medium Stack Initialization

Camera rays can see above and below water in the same image





This means we can see both above and below the water in the same frame. And the artist never has to worry about manually tagging anything which is really handy because the camera and water can be animated.

Open Problems

Now I just want to say a few words about open problems going forward.

Open Problems - Ray Marching

- Unify biased/unbiased methods?
- Stratification for unbiased methods?
- Analyze speed/variance tradeoffs for null scattering

I showed 3 different approaches to ray marching which are all used in different production renderers. And they are all comparable now that we understand how to construct the pdf even in the null scattering case.

Deciding which to use is not completely clear cut just yet. The biased approach in particular has some advantages that haven't been deeply explored, like the ability to take stratified steps. I think it should be possible to analyse the bias/variance trade-off here and maybe even find intermediate schemes that bridge the gap between the two methods.

And null scattering methods can tradeoff speed and variance which is handy but since higher variance usually implies taking extra samples, there is more analysis to be done over when its appropriate to do this in the context of different path sampling techniques.

Open Problems - Correlated Media

Extending the RTE to correlated particles:

- [Bitterli et al., 2018], [Jarabo et al., 2018], [d'Eon, 2018], [Guo et al., 2019]
- Efficient sampling?
- Artist friendly parameters?
- Spatially varying correlation?

Efficient methods for explicit granular media (sand, snow, ...):

- [Meng et al., 2015], [Müller et al., 2016]
- Unify special purpose methods with RTE?

Everything I've been talking about today is actually based on the assumption that the volume is made up a statistically independent particles. Changing this assumption actually requires a fairly big change to the theory.

Several papers have been digging into this recently, but there is still a lot of work to be done in this area. The most interesting aspect to us in production is figuring out how to expose these extra degrees of freedom to the artist in a meaningful way, and figuring out what it means for the correlation parameters to vary spatially which is part of answering what happens when different media types overlap each other.

Beyond this, there has been some very nice work on modelling case where particles are visible like sand and snow. Integrating these special purpose solutions in a unified way would be nice.

Open Problems - Subsurface Scattering

- Account for dielectric boundary?
- Improve Dwivedi sampling?
  - [Křivánek et al., 2014], [Meng et al., 2016]
  - Boundary awareness
  - Anisotropic scattering
  - Corners, thin regions

And finally, although I didn't have time to talk specifically about subsurface scattering today, most production renderers these days are modelling this volumetrically.

Except that in most cases we are still relying on approximations to deal with how light escapes the boundary. Properly treating the boundary has a big impact on the appearance but not many efficient solutions are known.

And while some basic techniques are know for isotropic scattering, the theory hasn't been fully investigated for anisotropic media and several heuristics are still needed for corners and thin regions.

Thank You For Listening!

SPI @ SIGGRAPH:
Mon, Jul 29, 2-5PM, *Women In Animation Chapters*
Mon, Jul 29, 6:30-8:35PM, *Electronic Theatre*
Tue, Jul 30, 3-4PM, *Open Color IO BOF*
Tue, Jul 30, 4-5PM, *Open Shading Language BOF*
Wed, Jul 31, 9-12AM, *Path Tracing in Production, Part 1*
Wed, Jul 31, 1-2PM, *AWS Impact of the Cloud on Production*
Wed, Jul 31, 2:50-3:10PM, *NVIDIA Presents: Ray Tracing Gems 1.1*
Thu, Aug 1, 3:45-5:15PM, *Making of "Spider-Man: Into the Spider-Verse"*

WE ARE HIRING!
Software Engineer - Shader Writer
Software Engineer - Katana
Software Engineer - C++
Software Engineer - Infrastructure
`recruiting@imageworks.com`
`www.imageworks.com/job-postings`

And that's all I have - thanks for listening.

Be sure to come see the Spider-Verse production session tomorrow and, yes we are hiring!

References i

- Benedikt Bitterli et al. “A radiative transfer framework for non-exponential media”. In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)* 37.6 (Nov. 2018), 225:1–225:17. DOI: `10.1145/3272127.3275103`.
- Alejandro Conty and Christopher Kulla. “Importance Sampling of Many Lights with Adaptive Tree Splitting”. In: *Proc. ACM Comput. Graph. Interact. Tech.* 1.2 (Aug. 2018), 25:1–25:17. ISSN: 2577-6193. DOI: `10.1145/3233305`. URL: `http://doi.acm.org/10.1145/3233305`.

References ii

Eugene d'Eon. "A Reciprocal Formulation of Nonexponential Radiative Transfer. 1: Sketch and Motivation". In: *Journal of Computational and Theoretical Transport* 47.1-3 (2018), pp. 84–115. DOI: `10.1080/23324309.2018.1481433`. eprint: `https://doi.org/10.1080/23324309.2018.1481433`. URL: `https://doi.org/10.1080/23324309.2018.1481433`.

References iii

- Julian Fong et al. "Production Volume Rendering: SIGGRAPH 2017 Course". In: *ACM SIGGRAPH 2017 Courses*. SIGGRAPH '17. Los Angeles, California: ACM, 2017, 2:1–2:79. ISBN: 978-1-4503-5014-3. DOI: `10.1145/3084873.3084907`. URL: `http://doi.acm.org/10.1145/3084873.3084907`.
- Jie Guo et al. "Fractional Gaussian Fields for Modeling and Rendering of Spatially-Correlated Media". In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2019)* 38.4 (2019).

References iv

- Adrian Jarabo, Carlos Aliaga, and Diego Gutierrez. “A Radiative Transfer Framework for Spatially-Correlated Materials”. In: *ACM Transactions on Graphics* 37.4 (2018).
- Jaroslav Křivánek and Eugene d’Eon. “A Zero-variance-based Sampling Scheme for Monte Carlo Subsurface Scattering”. In: *ACM SIGGRAPH 2014 Talks*. SIGGRAPH ’14. Vancouver, Canada: ACM, 2014, 66:1–66:1. ISBN: 978-1-4503-2960-6. DOI: 10.1145/2614106.2614138. URL: http://doi.acm.org/10.1145/2614106.2614138.

References v

- Christopher Kulla and Marcos Fajardo. “Importance Sampling Techniques for Path Tracing in Participating Media”. In: *”Computer Graphics Forum (Proceedings of the Eurographics Symposium on Rendering)”* 31.4 (June 2012), pp. 1519–1528. DOI: `10/f35f4k`.
- Peter Kutz et al. “Spectral and Decomposition Tracking for Rendering Heterogeneous Volumes”. In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2017)* 36.4 (2017), 111:1–111:16. DOI: `10.1145/3072959.3073665`.

References vi

- Johannes Meng, Johannes Hanika, and Carsten Dachsbacher. "Improving the Dwivedi Sampling Scheme". In: *Computer Graphics Forum (Proceedings of Eurographics Symposium on Rendering)* 35.4 (June 2016), pp. 37–44.
- Johannes Meng et al. "Multi-Scale Modeling and Rendering of Granular Materials". In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* 34.4 (July 2015). DOI: `10.1145/2766949`.

References vii

- Bailey Miller, Iliyan Georgiev, and Wojciech Jarosz. “A null-scattering path integral formulation of light transport”. In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* 38.4 (July 2019). DOI: `10.1145/3306346.3323025`.
- Thomas Müller et al. “Efficient Rendering of Heterogeneous Polydisperse Granular Media”. In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)* 35.6 (Dec. 2016), 168:1–168:14. DOI: `10.1145/2980179.2982429`.

References viii

- Ken Museth. “Hierarchical Digital Differential Analyzer for Efficient Ray-marching in OpenVDB”. In: *ACM SIGGRAPH 2014 Talks*. SIGGRAPH ’14. Vancouver, Canada: ACM, 2014, 40:1–40:1. ISBN: 978-1-4503-2960-6. DOI: `10.1145/2614106.2614136`. URL: `http://doi.acm.org/10.1145/2614106.2614136`.
- Charles M. Schmidt and Brian Budge. “Simple Nested Dielectrics in Ray Traced Images”. In: *J. Graphics, GPU, & Game Tools* 7 (2002), pp. 1–8.

References ix

- Magnus Wrenninge. “Efficient Rendering of Volumetric Motion Blur Using Temporally Unstructured Volumes”. In: *Journal of Computer Graphics Techniques (JCGT)* 5.1 (Jan. 2016), pp. 1–34. ISSN: 2331-7418. URL: `http://jcgt.org/published/0005/01/01/`.
- Magnus Wrenninge, Christopher D. Kulla, and Viktor Lundqvist. “Oz: the great and volumetric”. In: *International Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 2013, Anaheim, CA, USA, July 21-25, 2013, Talks Proceedings*. 2013, 46:1. DOI: `10.1145/2504459.2504518`. URL: `https://doi.org/10.1145/2504459.2504518`.

References x

Yonghao Yue et al. “Unbiased, Adaptive Stochastic Sampling for Rendering Inhomogeneous Participating Media”. In: *ACM SIGGRAPH Asia 2010 Papers*. SIGGRAPH ASIA ’10. Seoul, South Korea: ACM, 2010, 177:1–177:8. ISBN: 978-1-4503-0439-9. DOI: 10.1145/1866158.1866199. URL: http://doi.acm.org/10.1145/1866158.1866199.