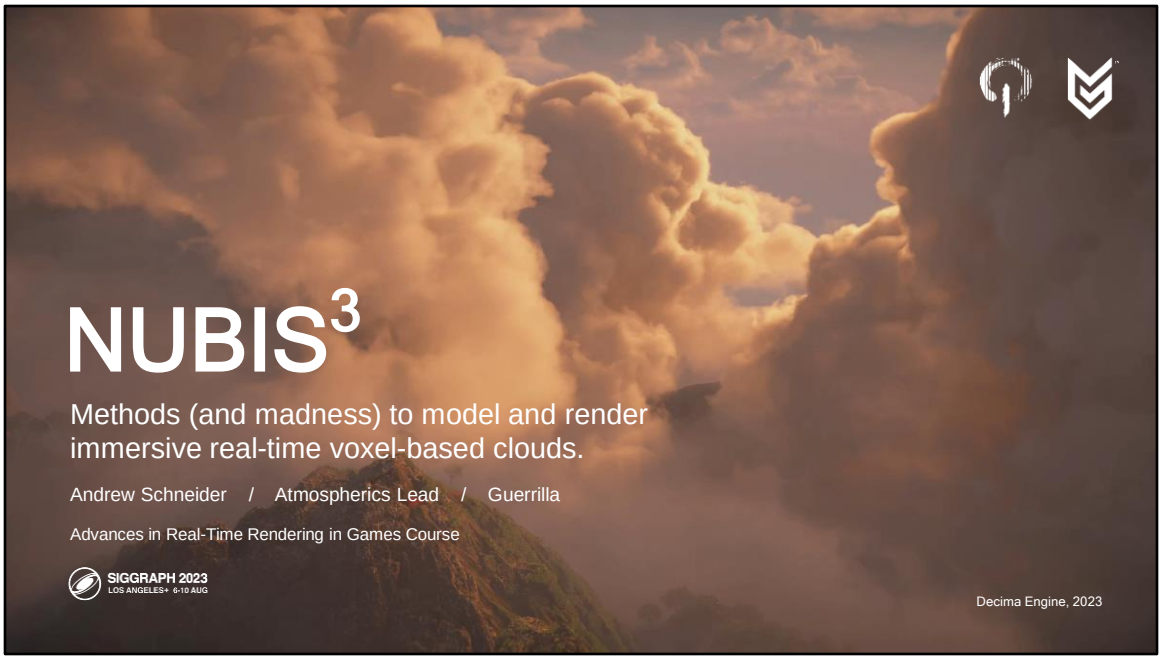
Thank you.
It is such an honor to be representing Guerrilla as a part of this course again - and at the 50th SIGGRAPH.
Both SIGGRAPH and this course are very important to me personally. I got my first job out of college at SIGGRAPH. I have made so many friends here, and gotten to watch as some pretty mind-blowing stuff was introduced to the world. I am lucky to have found this community of Artists and engineers to share ideas with. This, I guess, it's the sixth cloud talk I have given at this conference starting in 2011. I'm a little obsessive. This one in particular is something special for me though because it sees an offline rendering technique make its way into a AAA real-time game. I hope it will open some locked doors that we all knew were there, we just hadn't found the right set of keys for yet.



The phrase, pushing the envelope, is used by test pilots and engineers to describe the act of driving an airplane to the edge of its operating envelope, but it can more broadly mean to go beyond normal limits and to try out a new and different idea that is often viewed as radical or risky. While real time rendering is possibly not as cool as engineering and testing the fastest airplanes, the same phrase accurately describes our decision to re-engineer Nubis, our volumetric cloud renderer, in just a few months for a DLC expansion. Immersive High Detail Real-time Voxel-based cloud rendering for open world games is, in at least one way, like the sound barrier was to test pilots in terms of all of the memory and performance issues that make it nearly impossible on even the most recent graphics hardware. Feature animation and VFX houses still have to wait minutes to hours for one frame of the stuff to render. Today I am going to tell a story about how our methods and a little bit of madness made it possible to “cube Nubis” and cross this barrier. This story starts in an unexpected place over a decade ago in another industry.

Clouds in the skies of *Rio*

Andrew P. Schneider Trevor G. Thomson Matthew S. Wilson
Blue Sky Studios*

1 Introduction

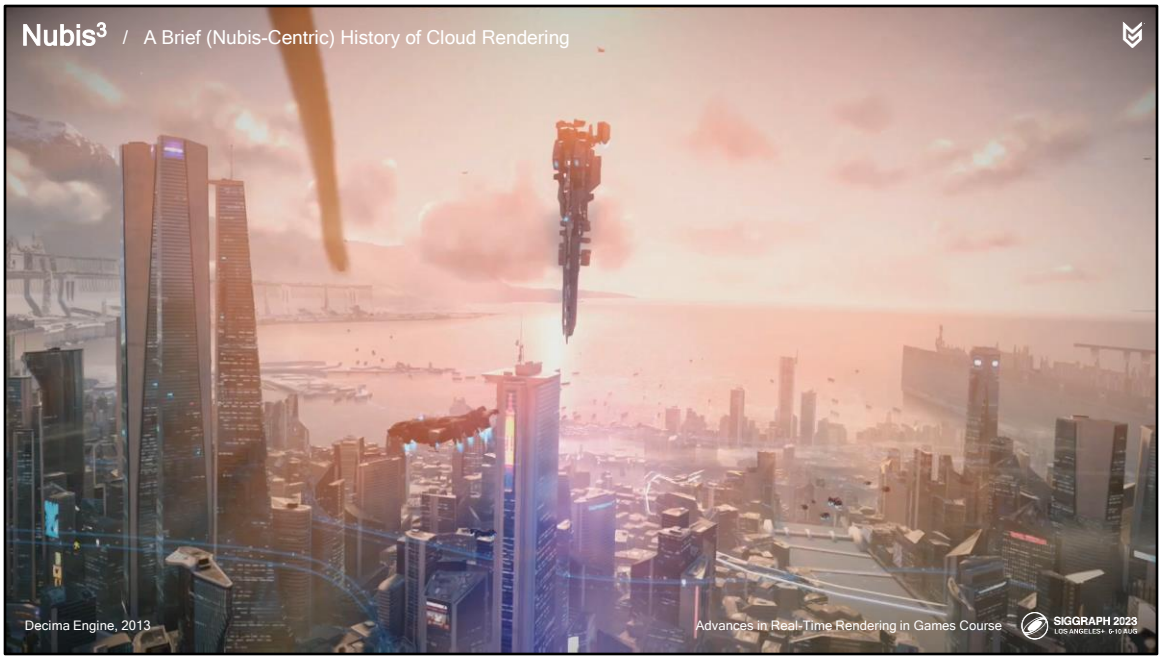
2 Synthesizing Clouds

3 Workflow

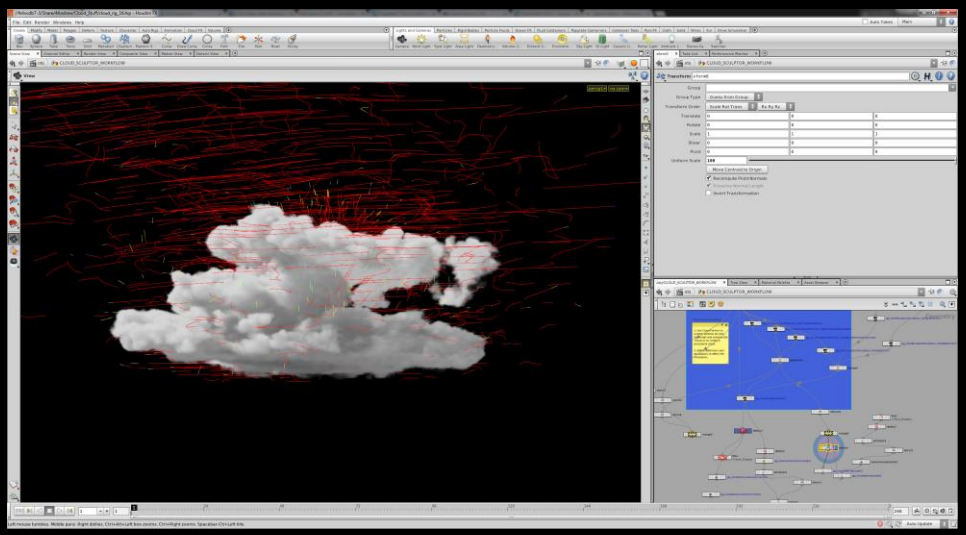
4 Volumetric Rendering

5 Conclusions

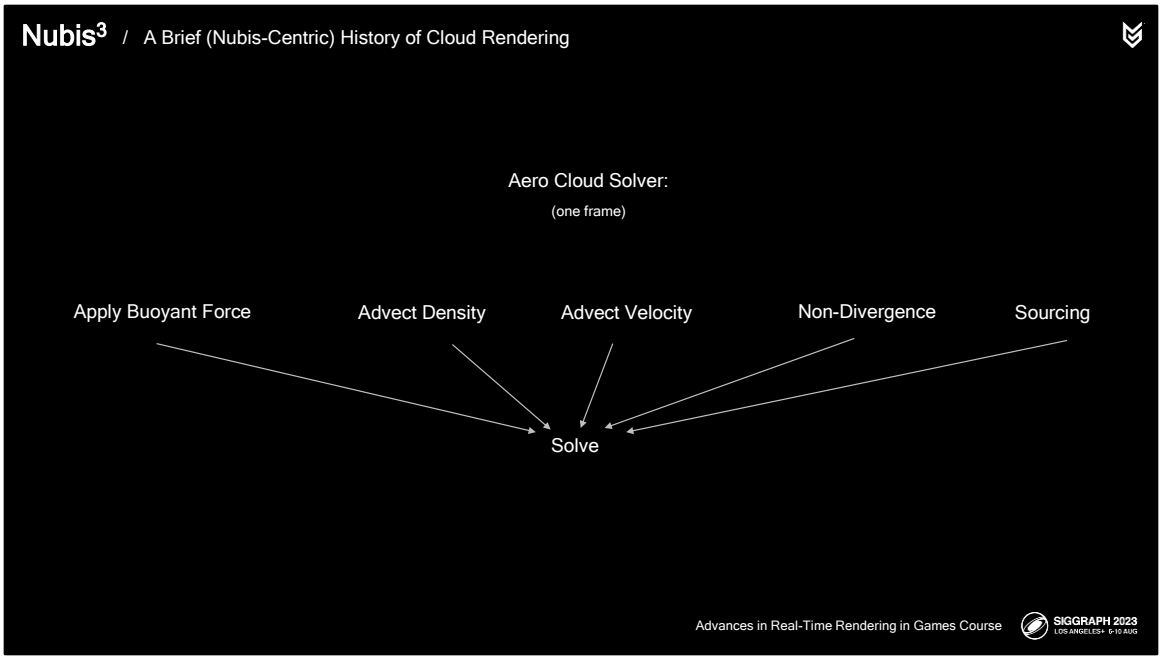
Back in 2011, I presented a talk at SIGGRAPH about how, for the Blue Sky Studios Animated Movie, Rio, I modeled clouds using fluid simulation and rendered them in a proprietary ray-tracer called CGI Studio. Our volumetric rendering tech, called SmogVox, was developed by my colleague Trevor Thomson. I created a library of simulated “voxel clouds” and then assembled the pieces into what my colleague Matt Wilson and I called “Frankencloudscapes”. All of this was necessary because Rio was a stereoscopic movie and traditional billboard-based clouds would not suffice. The results were convincing but render times ranged from 10 minutes to 4 hours on one core for a single 1920x1080 frame. This wasn't a problem in feature animation because we could just throw time and a render farm at the problem.



I joined the Horizon project at guerrilla in 2014 with the task of improving the 2D Skybox tech that was used in the previous Killzone games.



Naturally, as a naïve Offline Rendering transplant, I immediately gravitated toward voxel-based cloudscapes as THE solution to all of our problems.



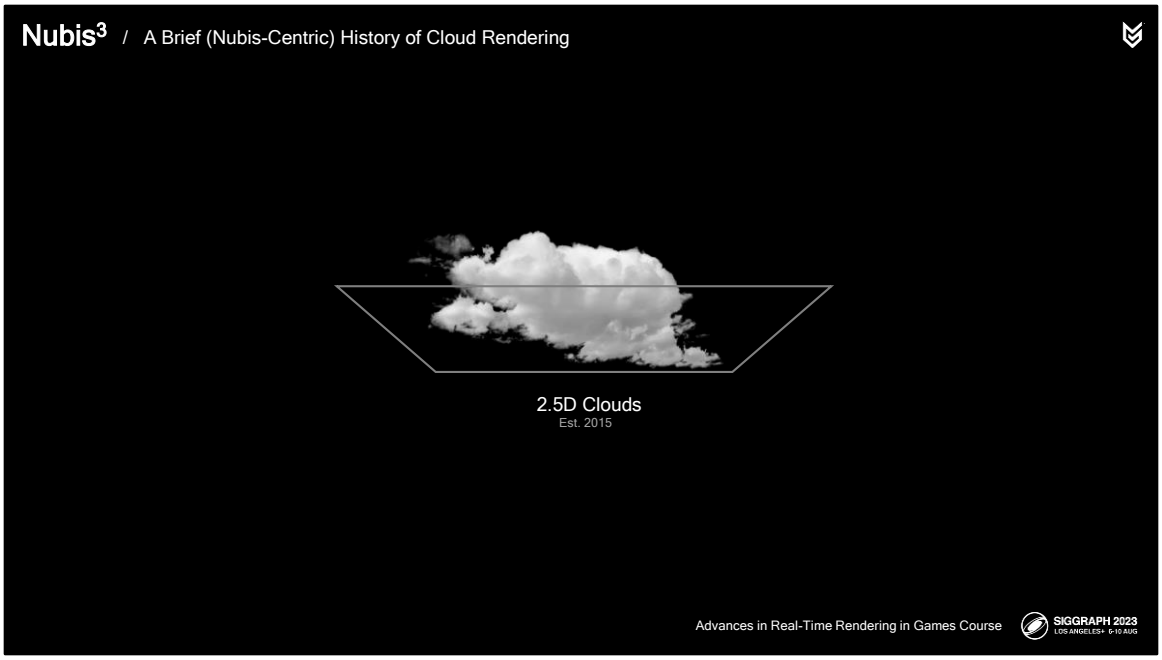
I built a custom fluid solver, nicknamed Aero, in Houdini using what are called micro solvers.

- Micro solvers are how Houdini modularizes the steps to solve for one frame of a fluid simulation, allowing the user to reorder them, modify them, introduce new solves and pretty much do anything you can think of.

In contrast to the popular combustion-style solvers, Aero was streamlined and enhanced to simulate only cloud growth as you can see here.

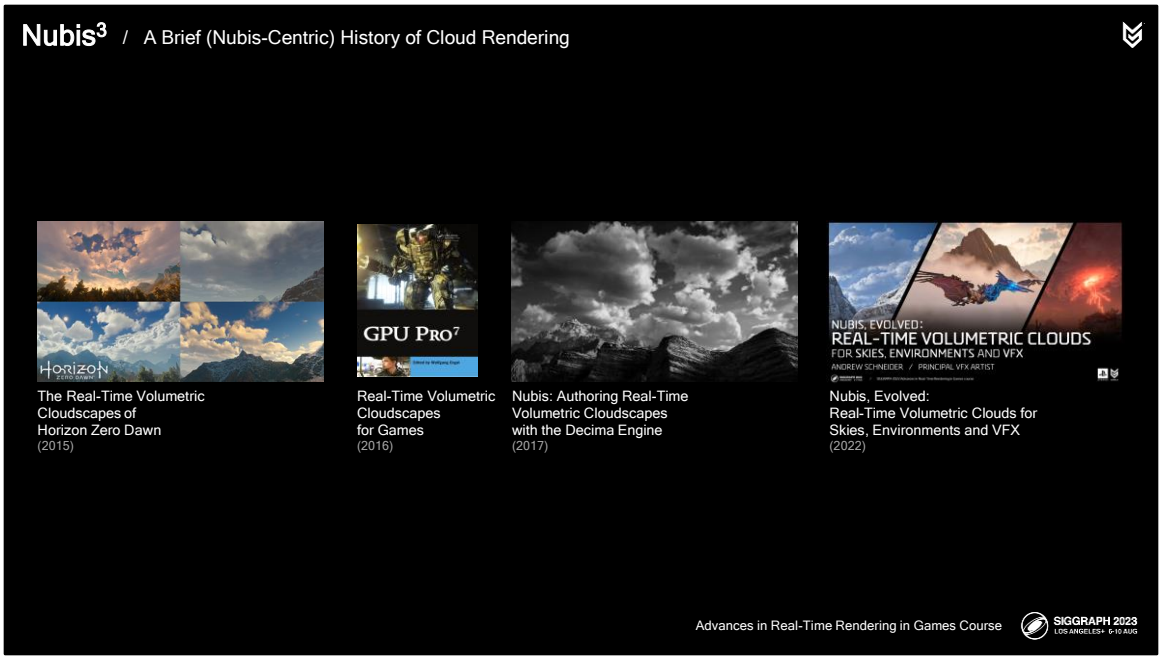
Back in 2014, We simulated several cloud shapes and tried various ways of rendering them, but none of these was performant enough for the PlayStation 4 or delivered on the requirements of our first game in the series: Horizon Zero Dawn.

- It just wasn't time yet, so the work was shelved and we went on to develop…

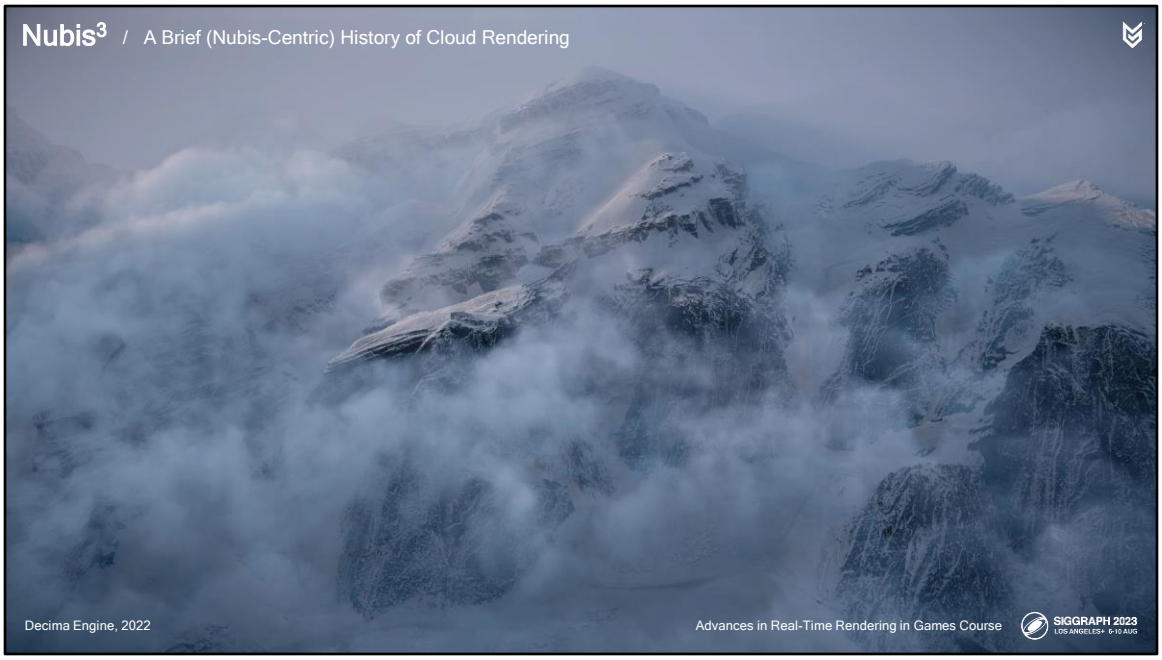


…several modeling and rendering methods that could be referred to as 2.5D clouds because these 3D clouds were not stored as 3D voxel data but rather as 2D texture data. We called our cloud system Nubis. Let’s review how this system

worked.

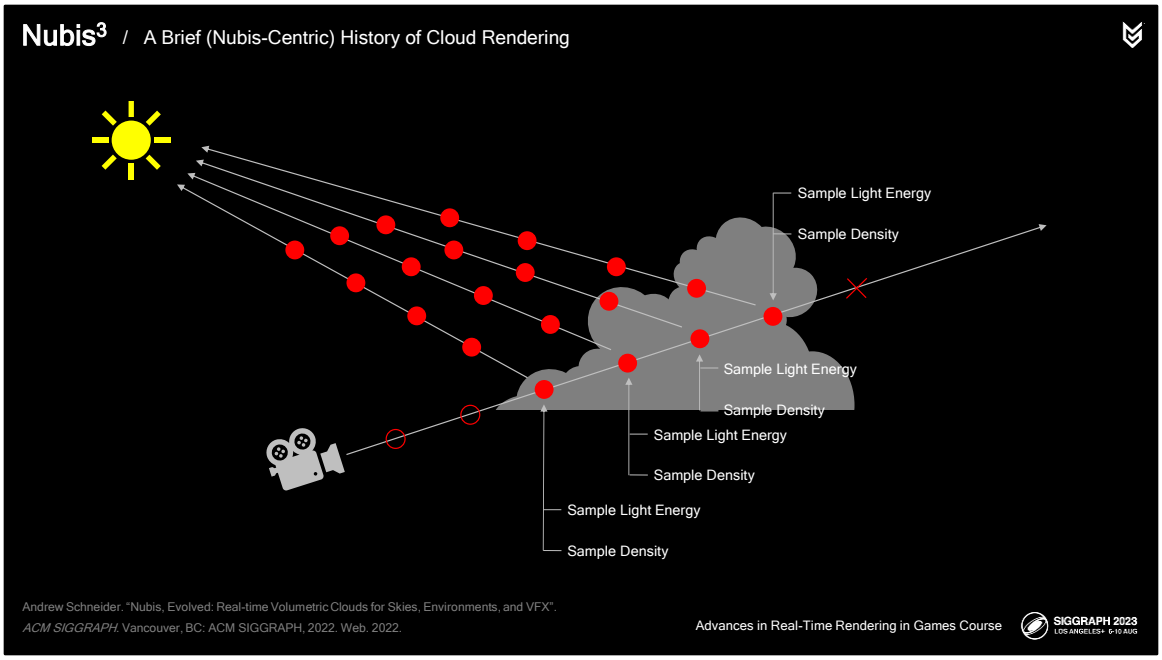


There is a lot of new material to cover today, so if you find any of this brief history a little too brief, I recommend that you look back at these publications for more information. I have listed the relevant talks in the bottom left when possible.



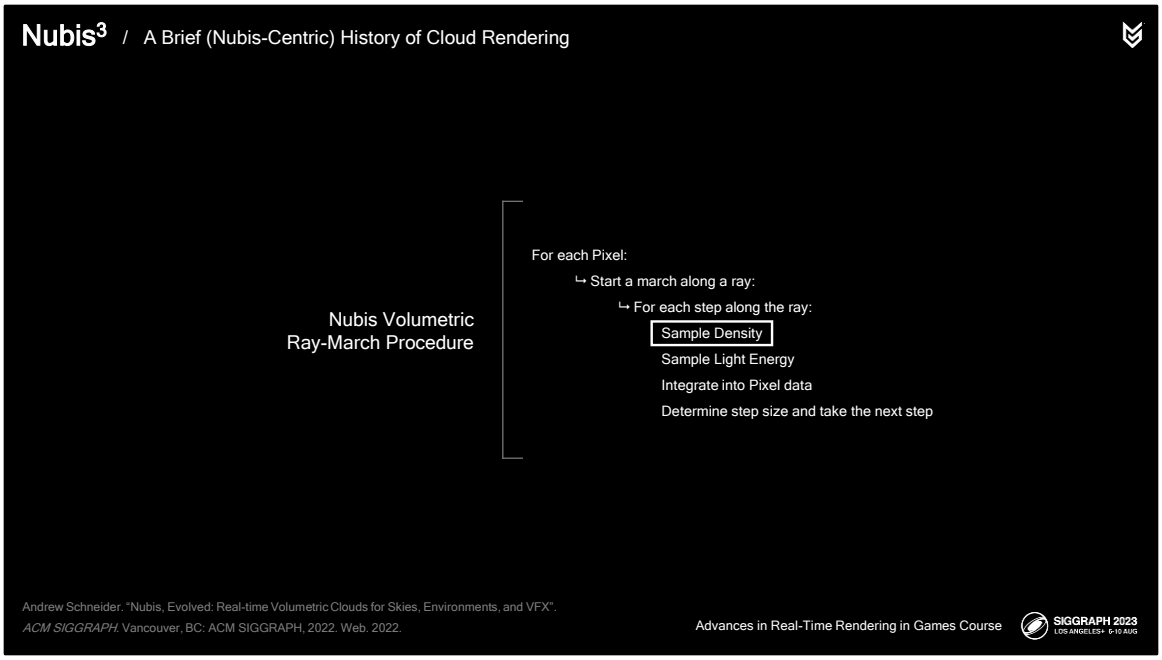
What made Nubis successful in our games was the ways in which we compressed our cloudscapes and added density and lighting details, which are specific to clouds, at render time while fine tuning a rendering technique known as volumetric ray-marching. This allowed us to model cloud evolution and time of day cycles which, in addition to producing distant clouds, could produce superstorms with red lighting, and some low altitude fog-like clouds that the player could fly through. Our method of ray-marching turned compressed cloud models into rendered frames of clouds in between 0.2

and 5 milliseconds depending on if we viewed them from afar or we if flew through them.
Let's take a brief look at how these modeling and rendering methods worked.



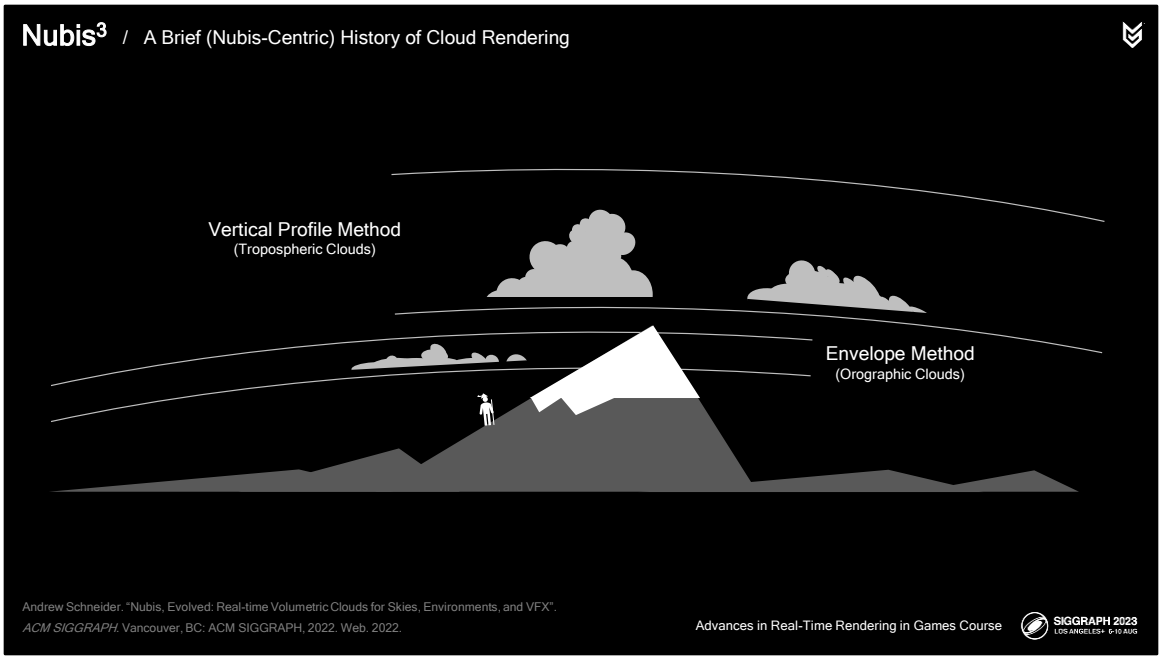
In a ray-march, once you know that your pixel contains clouds, the idea is to

- Step along the part of the ray that intersects the top and bottom of the cloud layer
- And at each step that contains cloud
- you sample density and you sample lighting, which requires a secondary and quite expensive ray-march toward the light, and then you integrate these data with the results of the previous step in a way that produces color and opacity data for each pixel.
- This is repeated until we reach full opacity and this is also done for every pixel to produce a full render.



Let's walk through this process and look at the methods we used at each step,

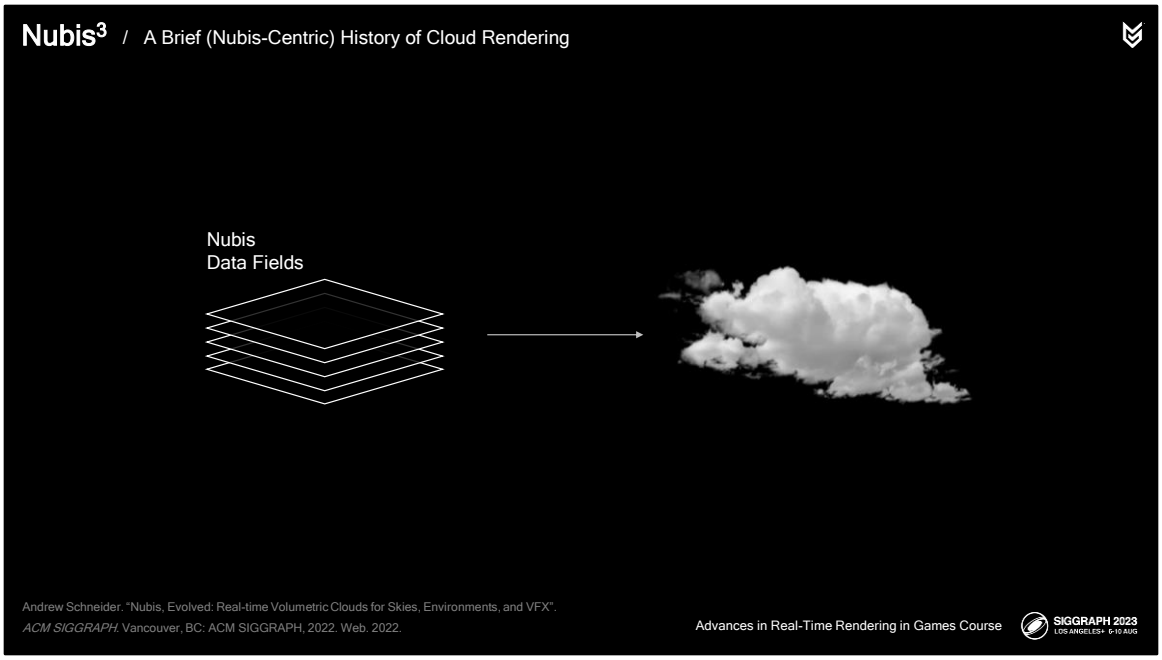
- starting at the heart of the ray-march the density sampler.



I'm going to focus on reviewing 2 of our cloud rendering methods here today, as both methods directly informed our approach to voxel-based clouds:

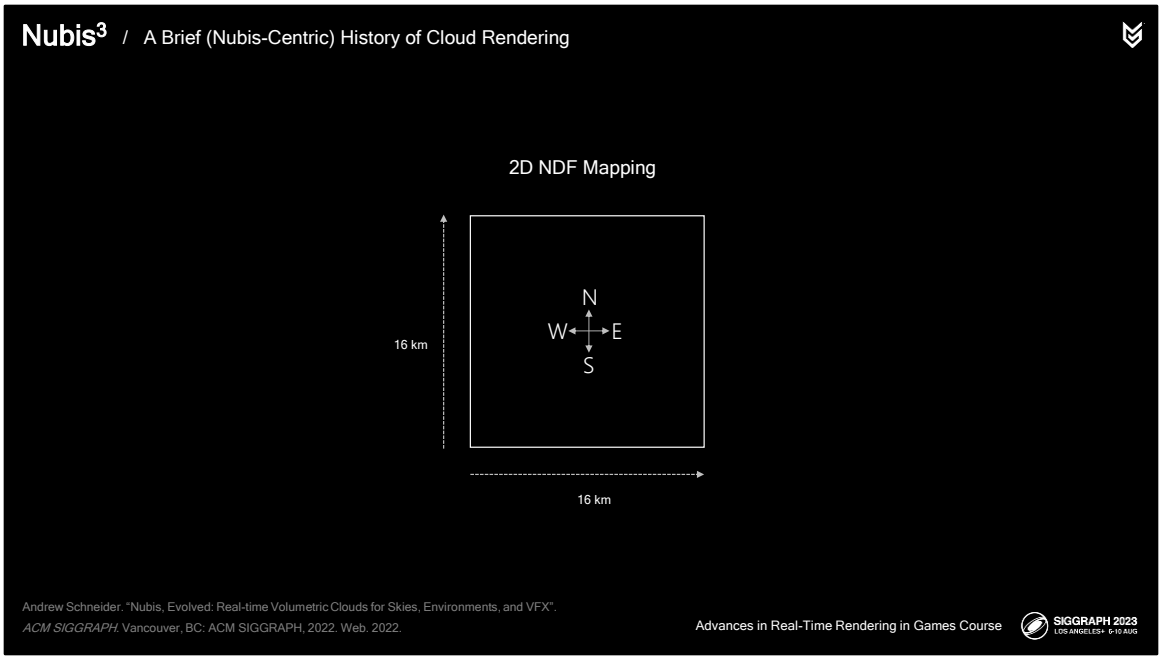
- The first is the Vertical Profile Method which were used for high altitude tropospheric clouds that the player could not reach
- and the second is the envelope Method which were used for orographic clouds that the player can fly through.

Each has their own density sampling approach but share some similarities in design.

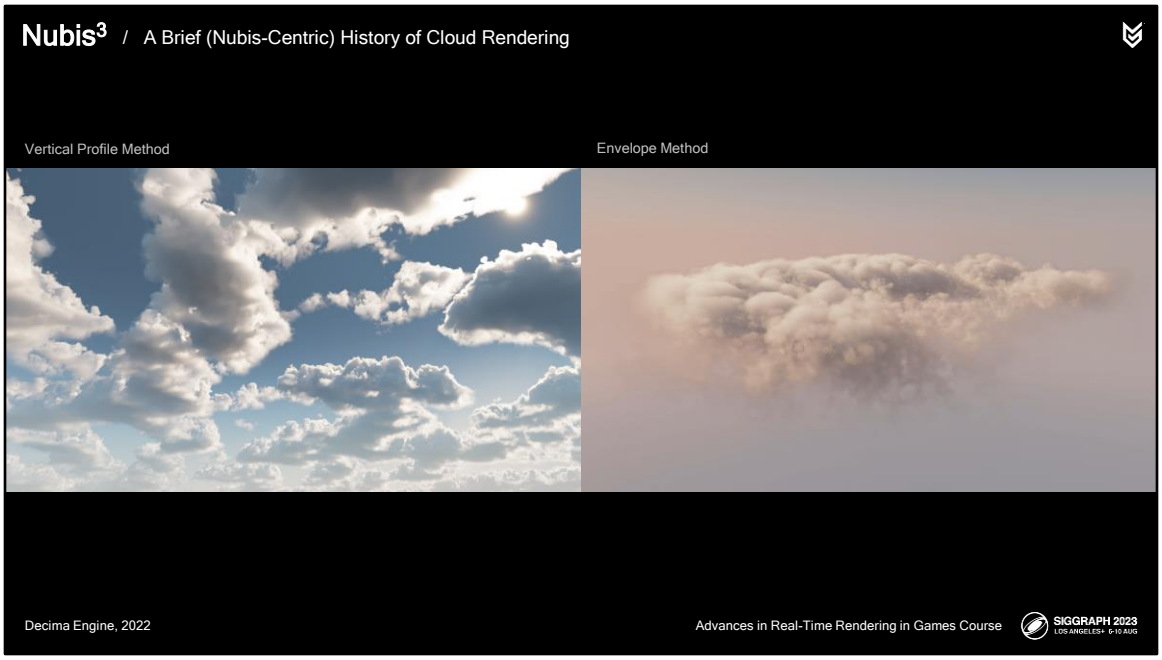


2D Data fields that we call Nubis Data Fields, or NDF's, contain instructions that the cloud rendering compute shader

•uses to "Decompress" 3D clouds from authored 2D data.



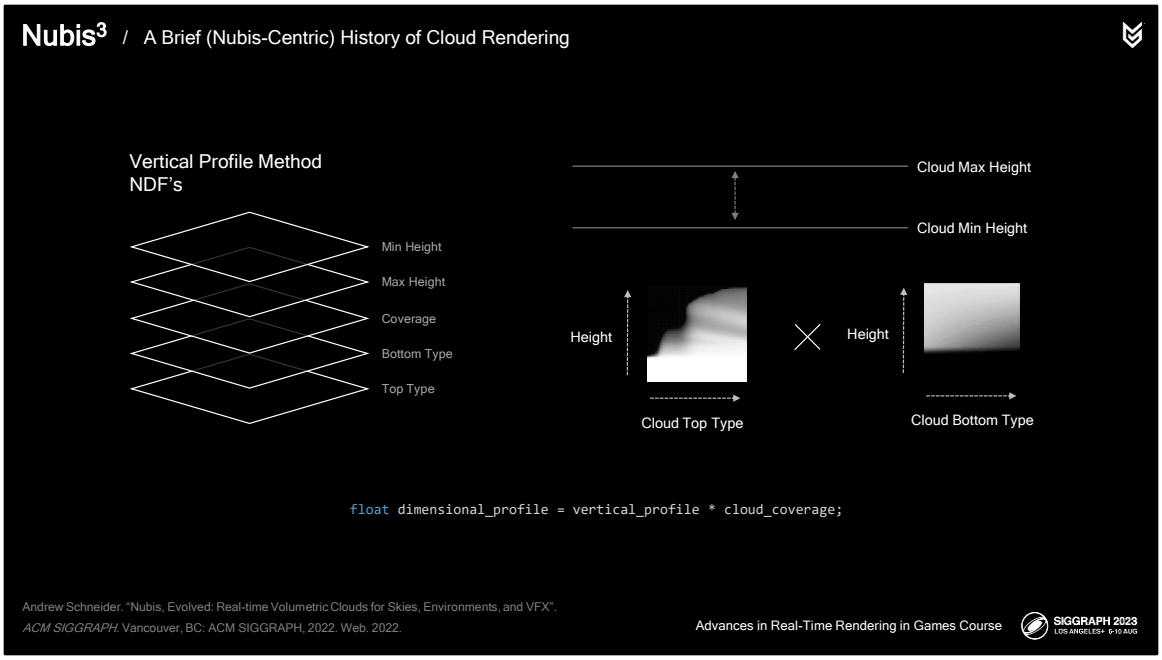
These 2D Data fields were mapped over a 16km area of our world and sampled at render time to get the data needed to construct 3D cloudscapes.



During the ray-march, when sampling density, both methods constructed a high resolution 3d volume, as seen here, from a

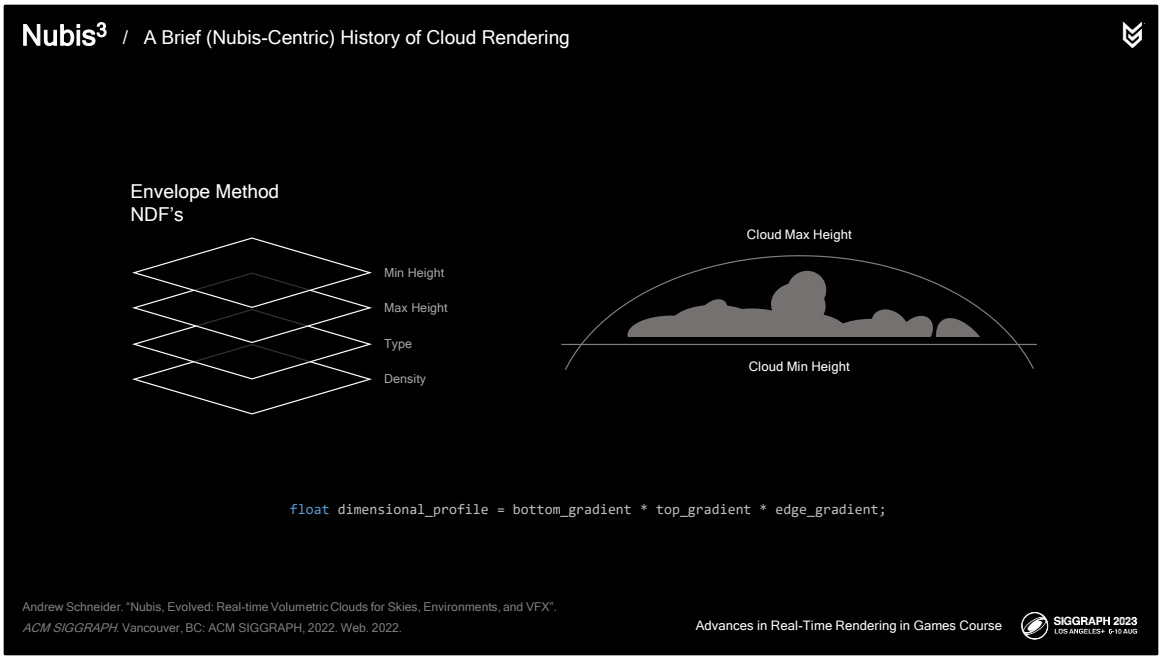
- low resolution dimensional profile, as seen here.

Even though these may look similar, the way that the dimensional profile was constructed differed per method.



In the case of the Vertical profile method, We started with 5 2D NDF's.

- Cloud Min and max height defined the vertical range for the cloud.
- Top Type and Bottom Type were used to sample vertical profile lookup textures and the results were
- multiplied to create what we call the vertical profile.
- Then, we scaled this vertical profile by the cloud coverage data to control where clouds formed.



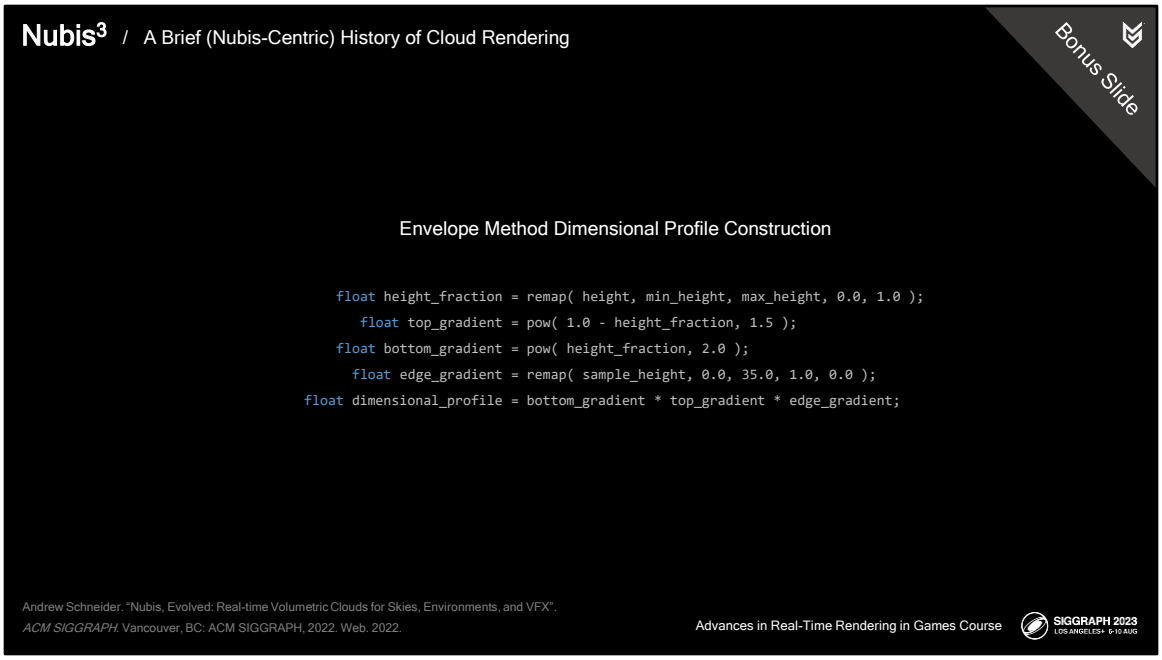
In the case of the Envelope profile method, We started with 4 2D NDF's

- The min and max height were used to define an envelope in which clouds could be rendered.
- The dimensional profile was the product of 3 gradients:

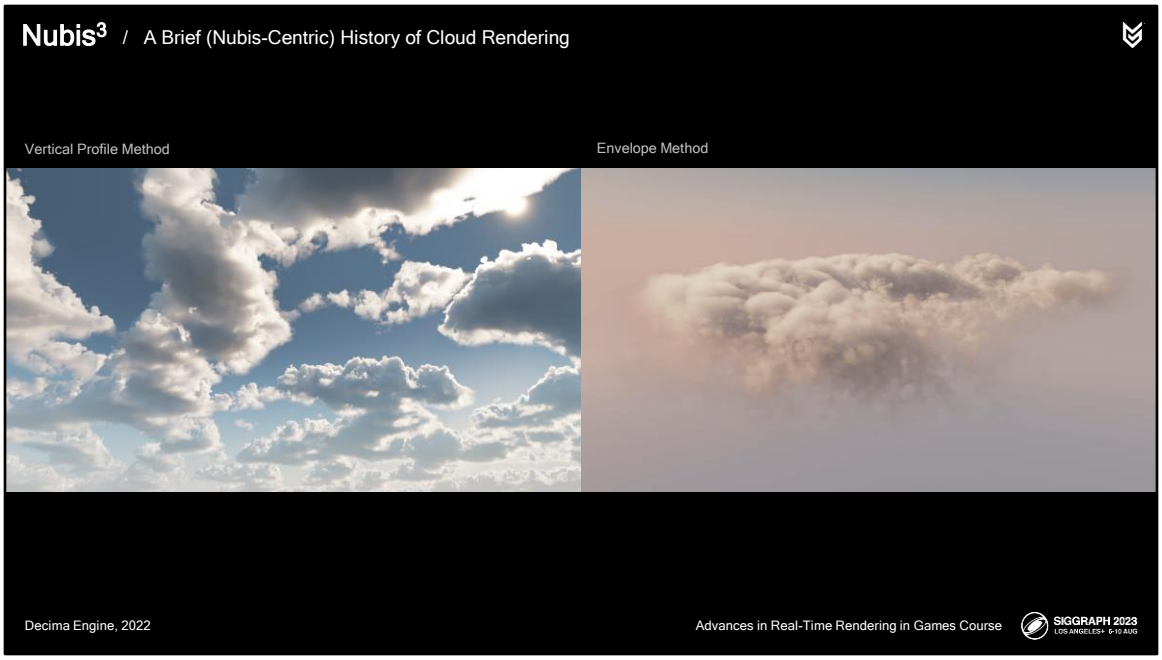
A top gradient that increased in value pointing down

A bottom gradient that increased in value going up

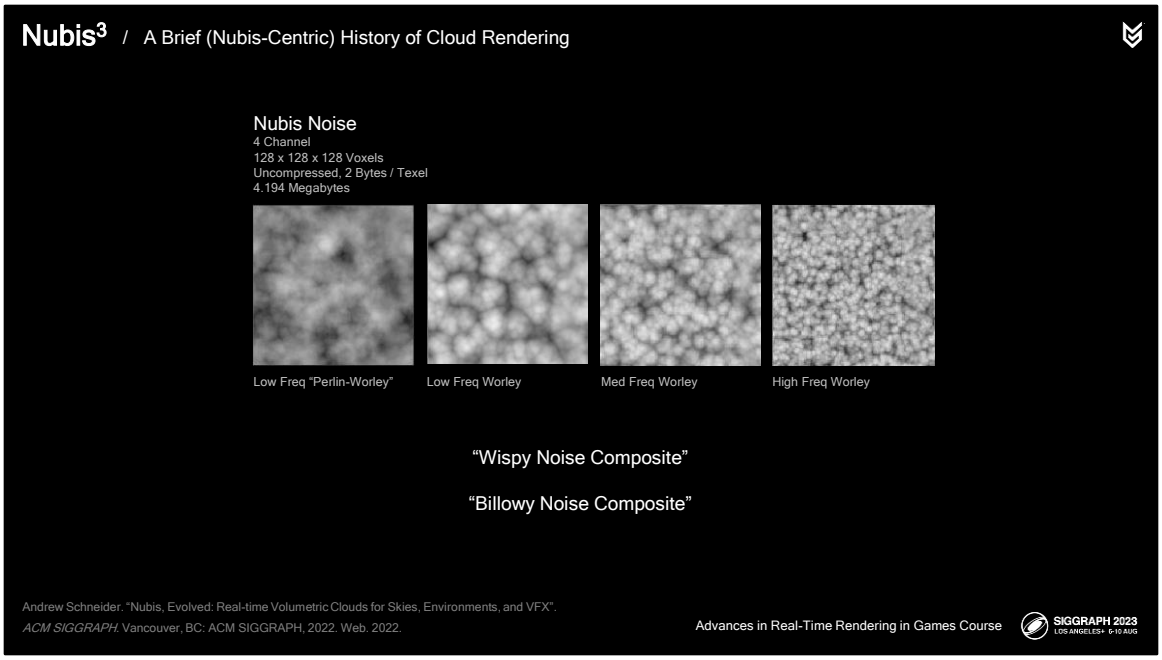
And an edge gradient that increased in value going in from the sides.



- In the case of the Envelope profile method, We started with 4 2D NDF's
- The min and max height were used to define an envelope in which clouds could be rendered.
- The dimensional profile was the product of 3 gradients.
- A top gradient that increased in value pointing down
- A bottom gradient that increased in value going up
- And an edge gradient that increased in value going in from the sides.

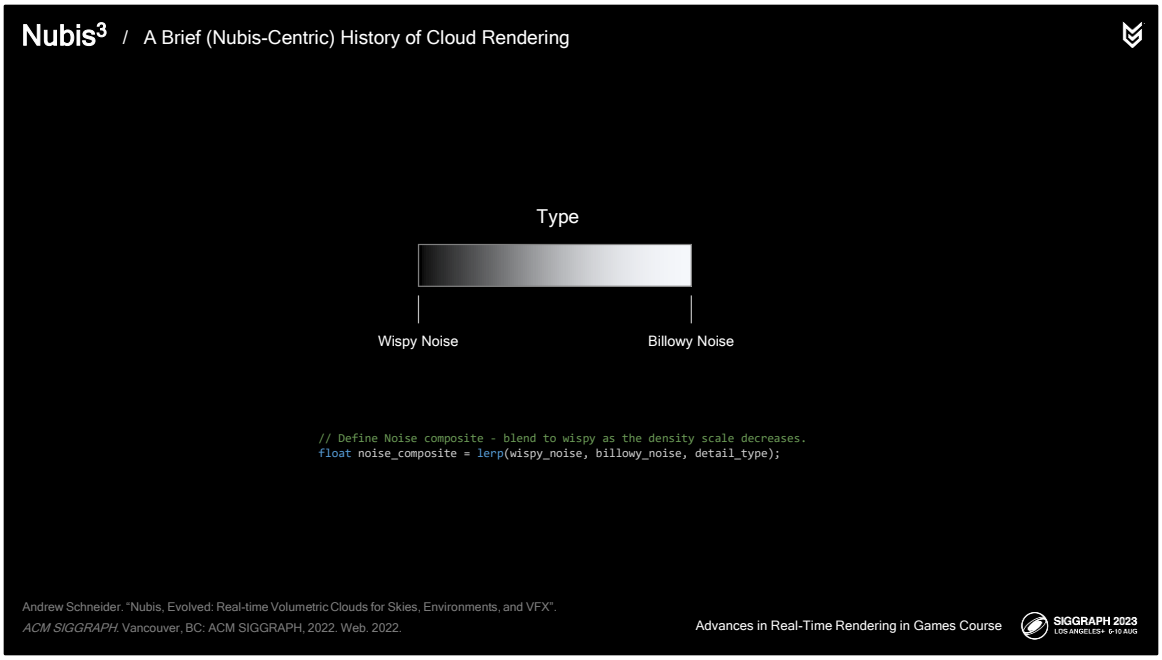


We used 3d noise to erode the dimensional profile and “up-rez” it into the detailed results you see here.



We developed what we call Perlin-Worley noise to simulate a mix of wispy and billowy details in clouds and we used Worley noise to simulate the billowy details seen in clouds.

- These were combined in various ways per model to create what we called noise composites for potential wispy and billowy details at the sample position.

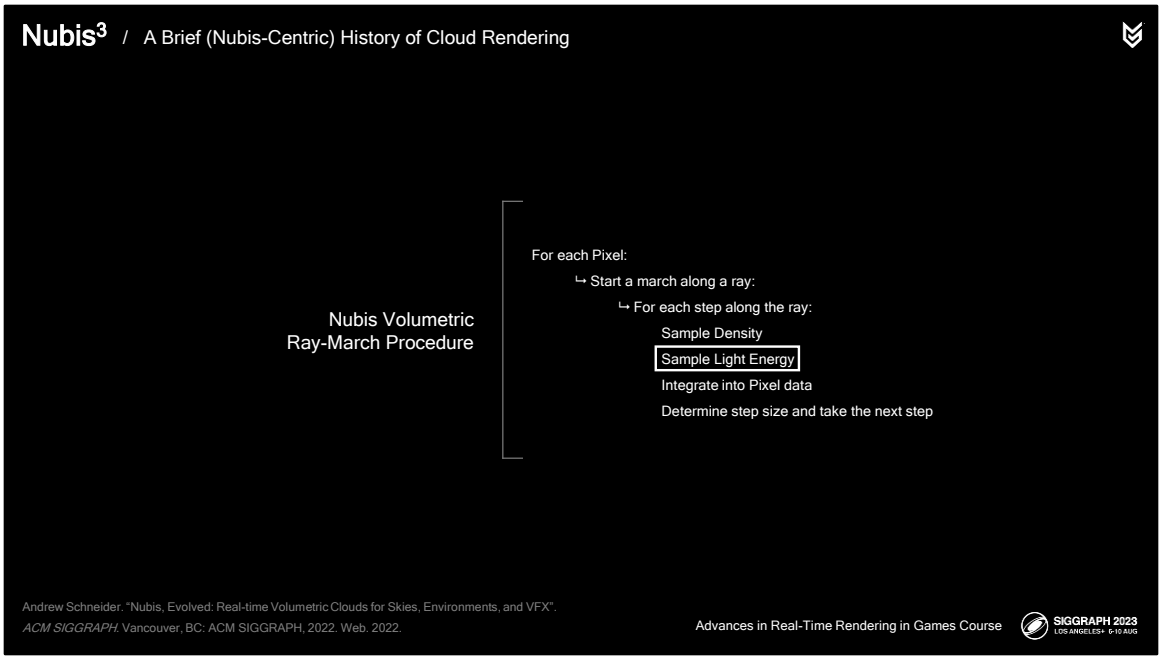


Then we blended from wispy to billowy noise composites using the Type data at the sample position.

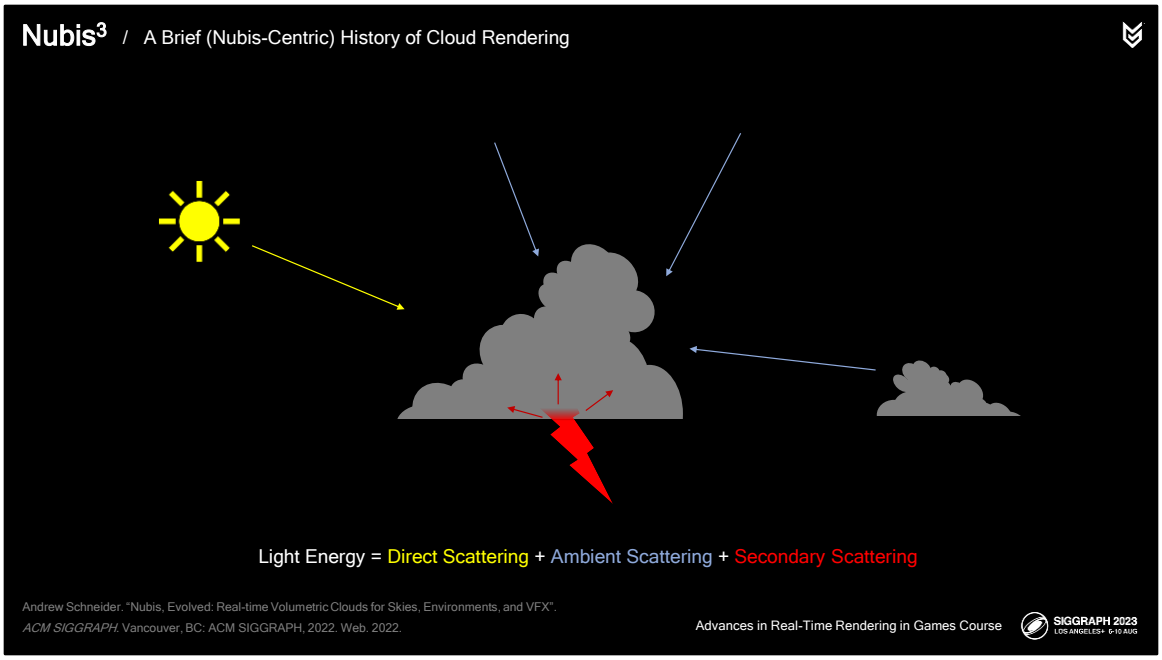
The erosion was modeled as the subtraction of the

- inverse of the dimensional profile from the noise composite.

“Pseudomotion” was simulated for both models by animating the noise in the wind direction.



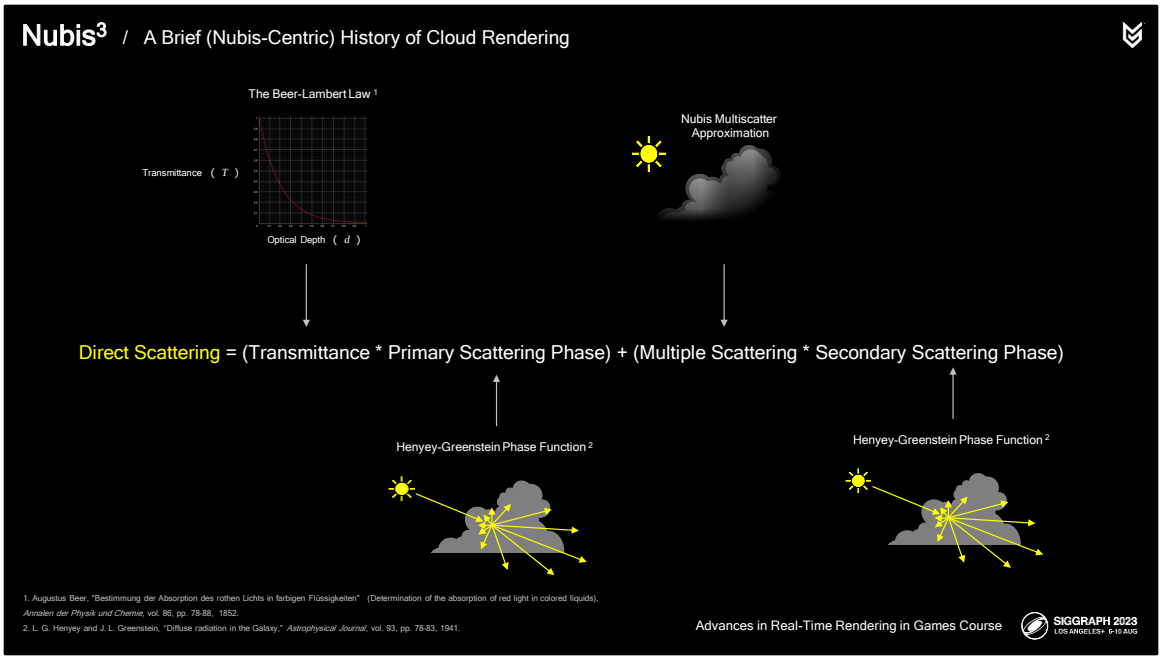
After sampling density we sample light energy.



Both methods defined light energy as the sum of 3 components:

- Direct Scattering, which represents all of the light energy coming from the sun.
- Ambient scattering which represents the light energy coming from the sky and neighboring clouds.
- And secondary scattering which represents the light energy from sources like Lightning.

There are some differences in the way that each method calculates these energies. For a full explanation, I direct you to our most recent talk in this course from 2022. In the interest of brevity, I will summarize.

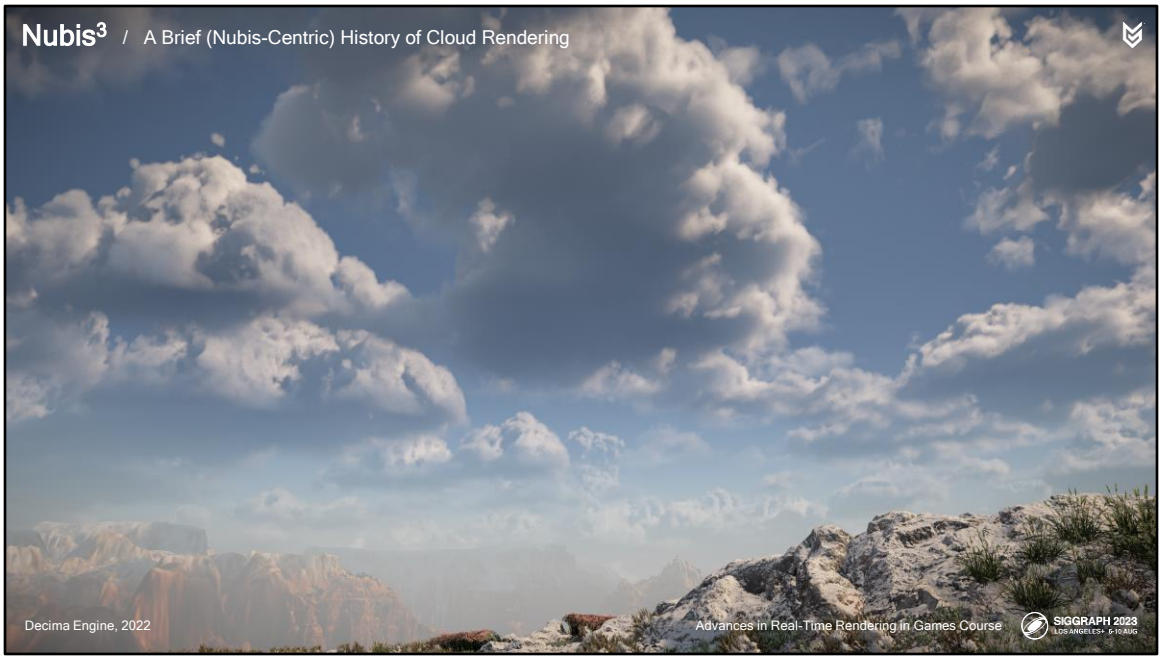


We defined Direct scattering at a given sample point in a cloud as a function of three probabilities:

- Transmittance, Scattering phase, and multiple scattering.
- Transmittance is a measure of the amount of light at a given depth in an optical medium. The graph indicates how the intensity of light decreases as more of it is absorbed over depth in the cloud. In order to gather this depth, you must conduct a second (and I will add very expensive) light ray march toward the light.
- Scattering phase is a measure of how much of the energy that reached the sample position will scatter toward our eyes, given the view vector and light vector. Light scatters in a cloud as it refracts through tiny water droplets or ice. In clouds light has a tendency to scatter more along the original path, so we use several Henyey Greenstein phase functions to produce an effect that is art directable and somewhat physically based.
- Multiple scattering describes light that scatters-in to our view vector after refracting around from multiple encounters with water molecules. We use our dimensional profile as the basis of a probability field to describe where in-scattering happens more frequently because the deeper you are in a cloud, the more potential there is that photons have scattered from their original path. We attenuate this field using another beer-lambert attenuation curve as the multiple

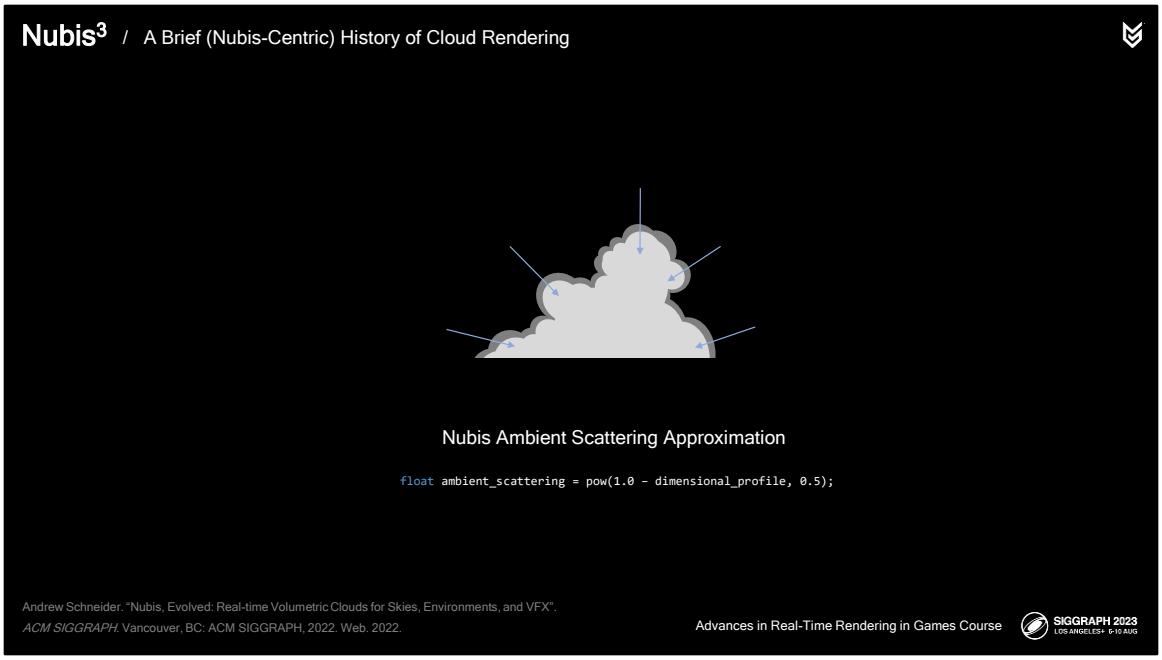
scattered light itself is eventually absorbed the more it scatters. We also attenuate this over height to account for he fact that less light scatters into the cloud from below.

- Finally, we scale this by a phase function to ensure that it too is directional.



Here is what a cloud looks like without multiple scattering approximation

- And then with this approximation method.



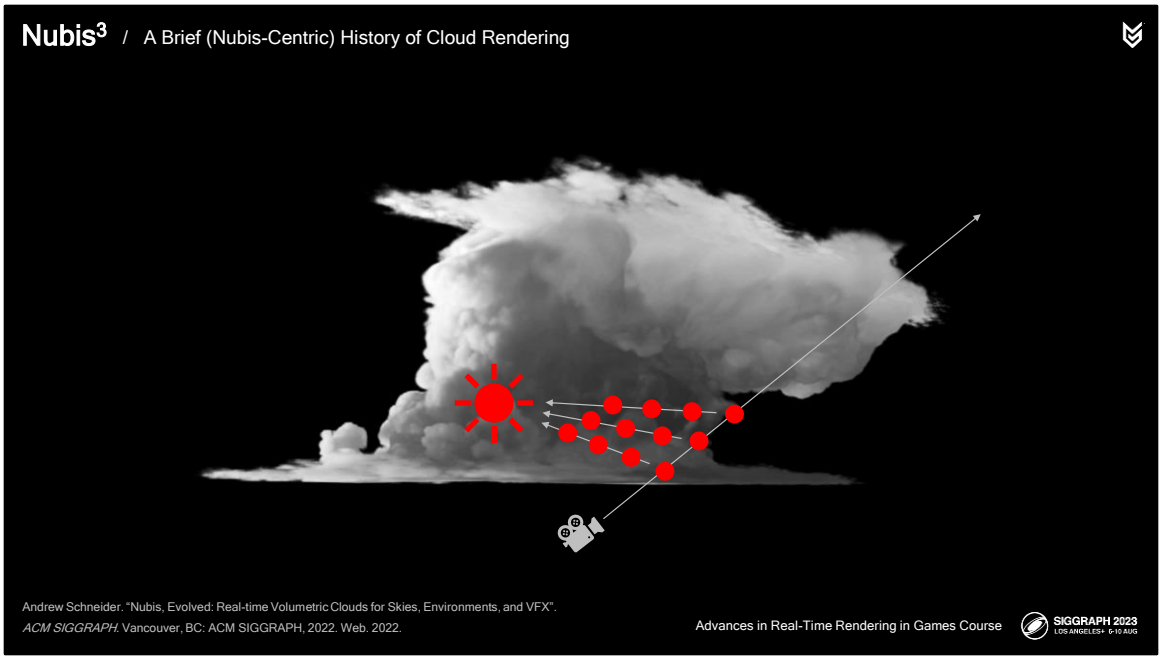
To approximate the effect of ambient light entering the cloud from all around without ray marching to many light sources, we once again modeled this geometrically as a probability field.

- The majority of light comes from above and around and penetrates just the surface of the cloud.

The dimensional profile already provided the gradient from the outside of the cloud to the inside. We used the inverse of the dimensional profile to create a gradient representing the probability that light will reach a given point inside the cloud and then scatter to our eyes.

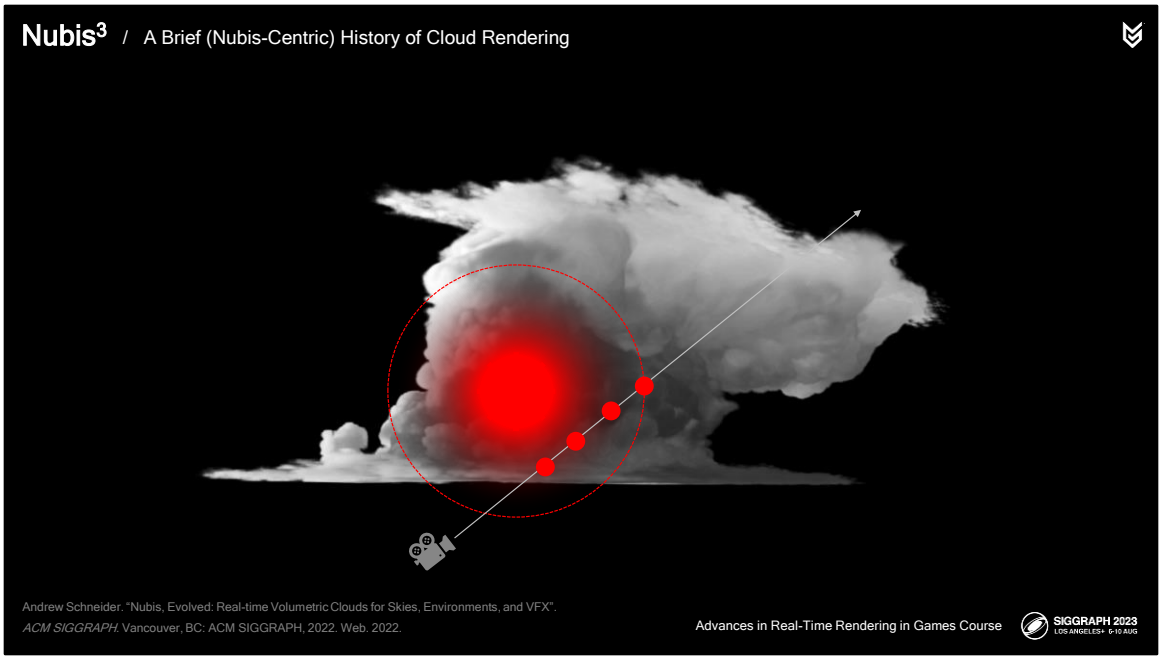
Here is what a cloud looks like with just this ambient component

- Then with just direct light energy.
- And with both Direct and ambient light energy.



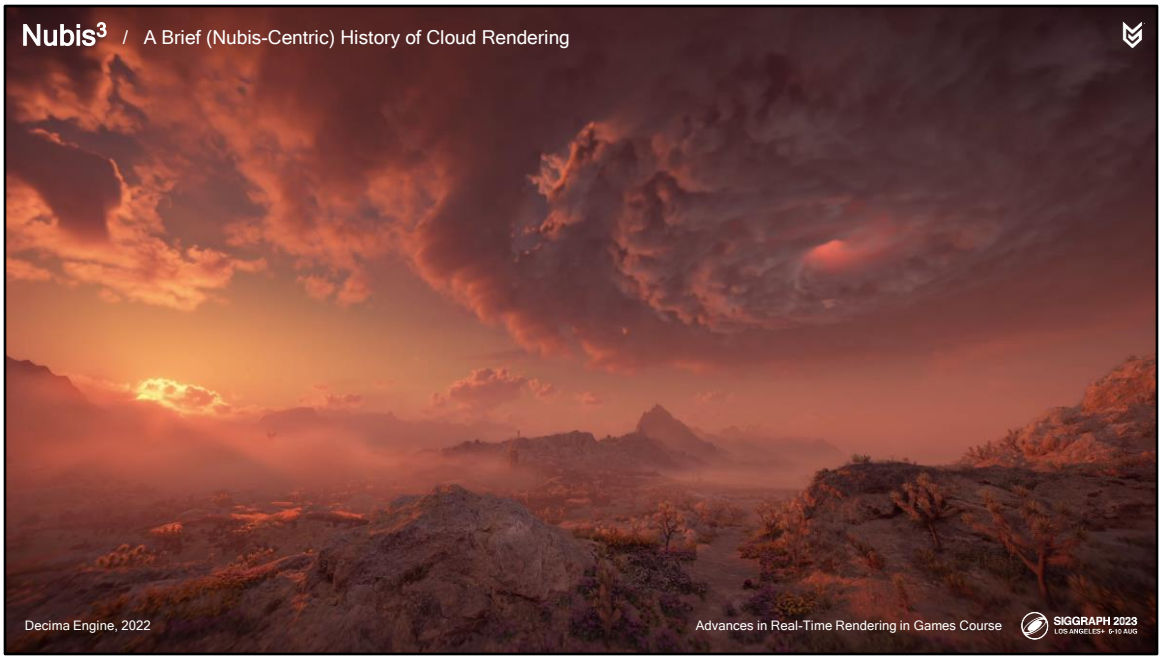
Now secondary light energies. Normally, for any light source that is positioned inside a volumetric effect,

- you need to do an expensive second light ray march for each sample taken. For those keeping track that would mean 3 Ray-marches - not an ideal approach.

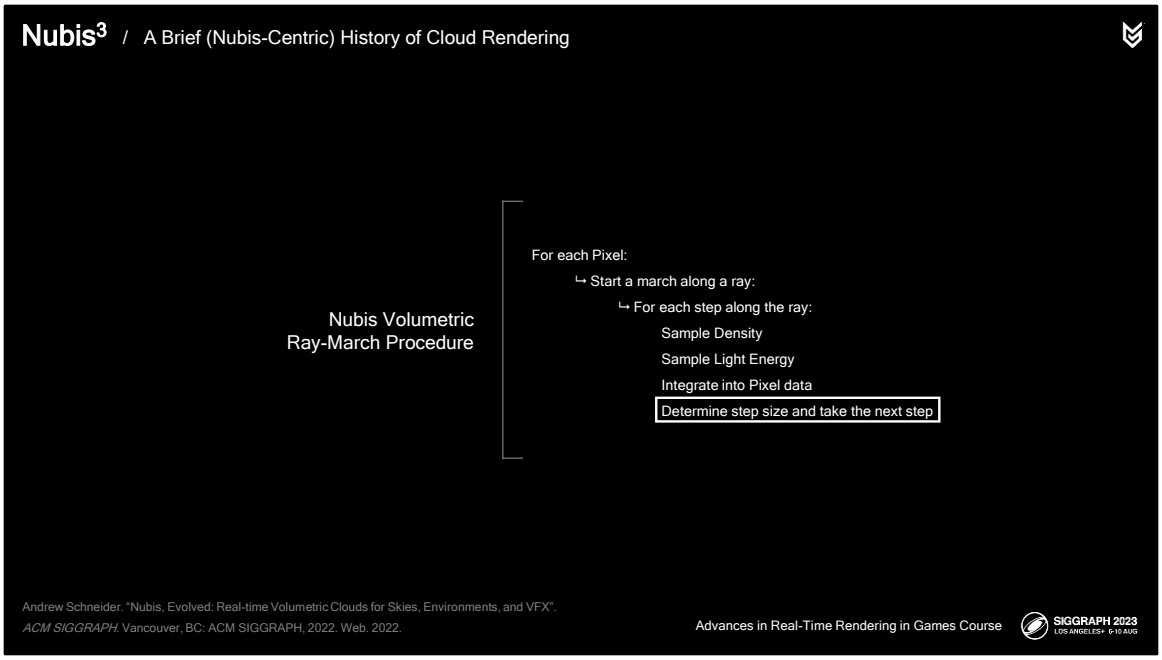


Once again we looked at lighting as a geometric probability problem.

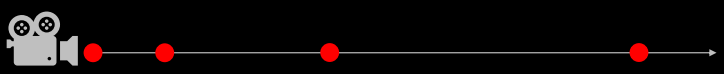
- We modeled a volume of light that would approximate the light energy present at each sample taken around the light source in the primary ray-march. There is a bit more to this but I will explain it later when we look at how we did this for voxel clouds.



Here is what this looked like in our Superstorm from Horizon Forbidden West.



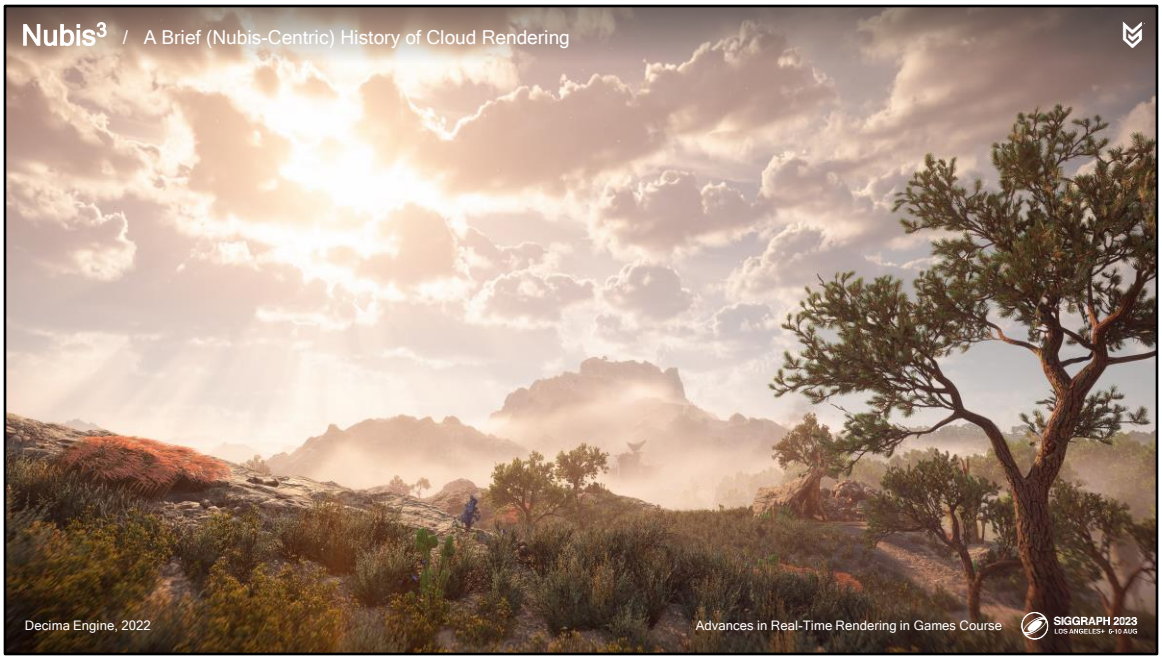
The next step (pun intended) is to determine the step size for our march so that the next sample can be placed along the ray.



```
// Define step size constants
float near_step_size = 3.0;
float far_step_size_offset = 60.0;
float step_adjustment_distance = 16384.0;

// Calculate distanced-based step size
float step_size = near_step_size + ((far_step_size_offset * distance_from_camera) / step_adjustment_distance);
```

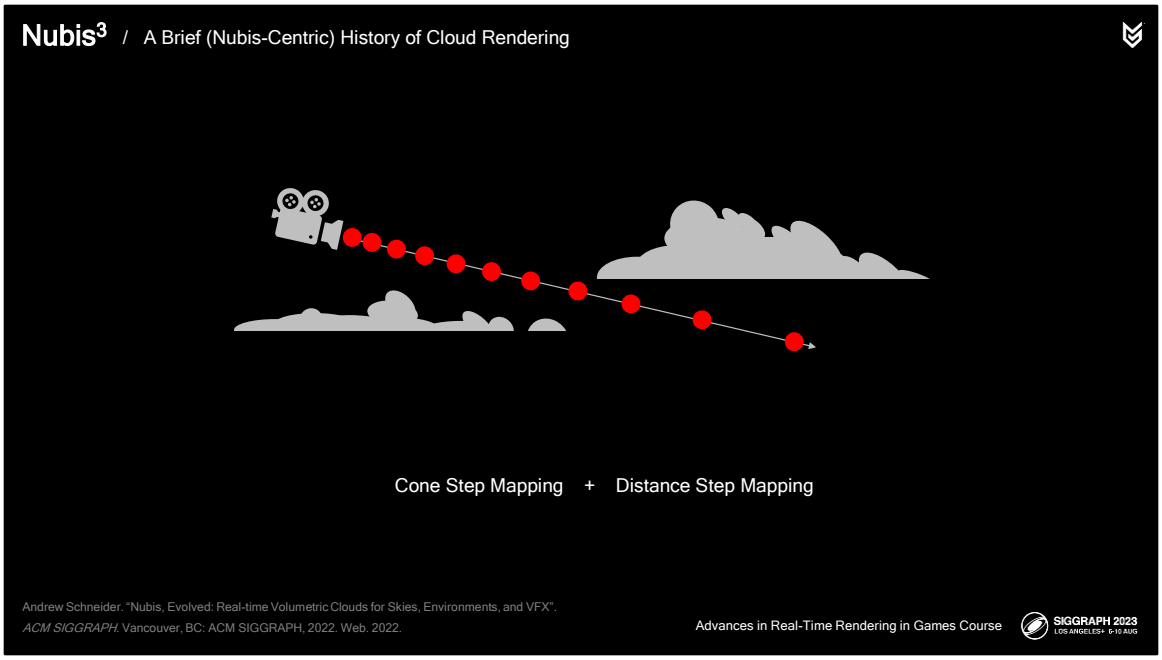
Andrew Schneider. "Nubis, Evolved: Real-time Volumetric Clouds for Skies, Environments, and VFX".
ACM SIGGRAPH. Vancouver, BC: ACM SIGGRAPH, 2022. Web. 2022.

The Vertical profile method employed an adaptive stepsize where the size of the steps increased over the distance from the camera. This resulted in fewer samples farther from camera where less precision was needed.



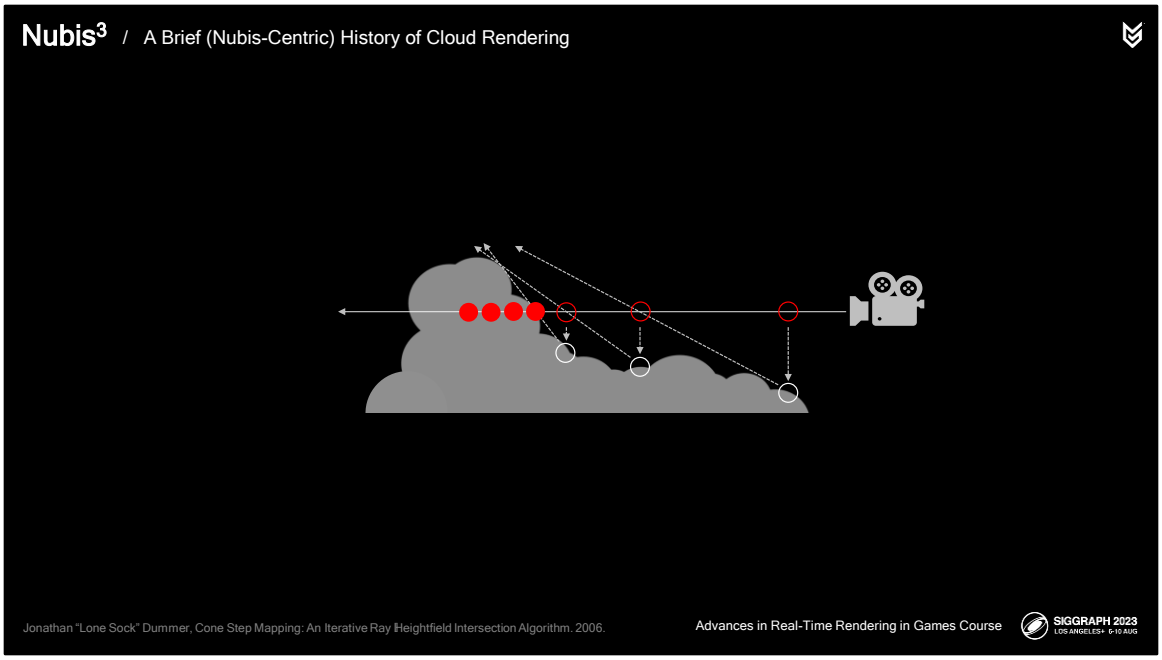
Times ranged from half a millisecond for cases like this where you are looking at partial sky,

- to 2.2 milliseconds in cases with full sky visibility.



However, to support flight through clouds created with the envelope method, which are almost all near camera and required a lot of samples close by, we had to get a bit creative in order to avoid placing a lot of samples in empty space.
We combined

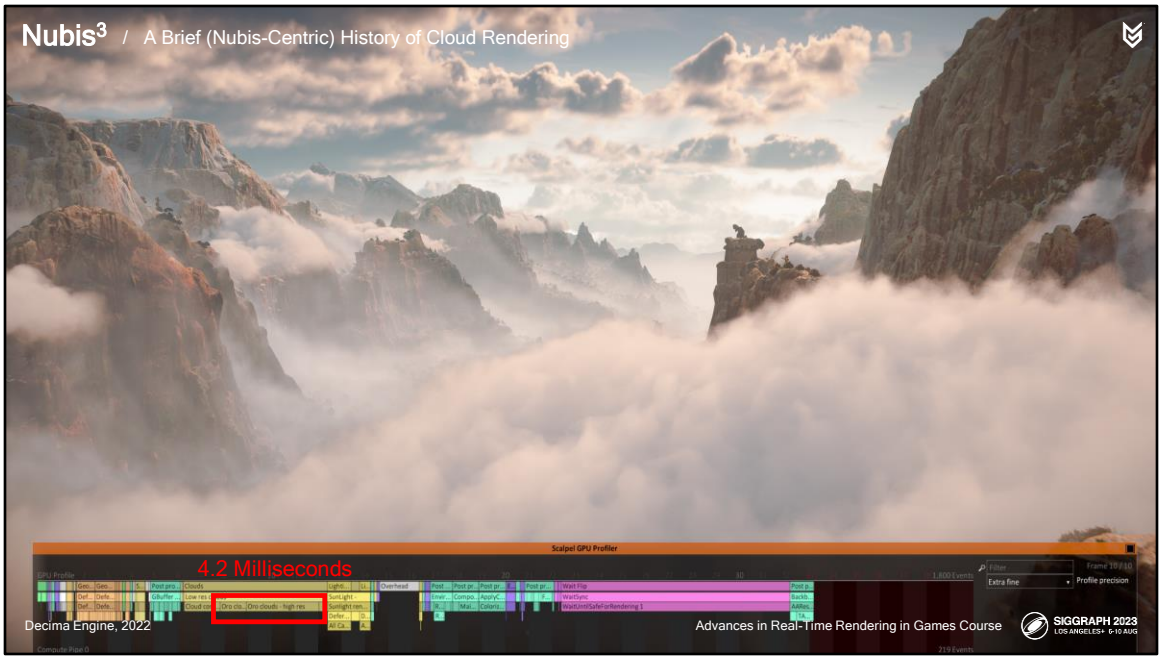
- two methods called Cone Step Mapping and distance step mapping so that we could place samples as efficiently as possible while using only height data.



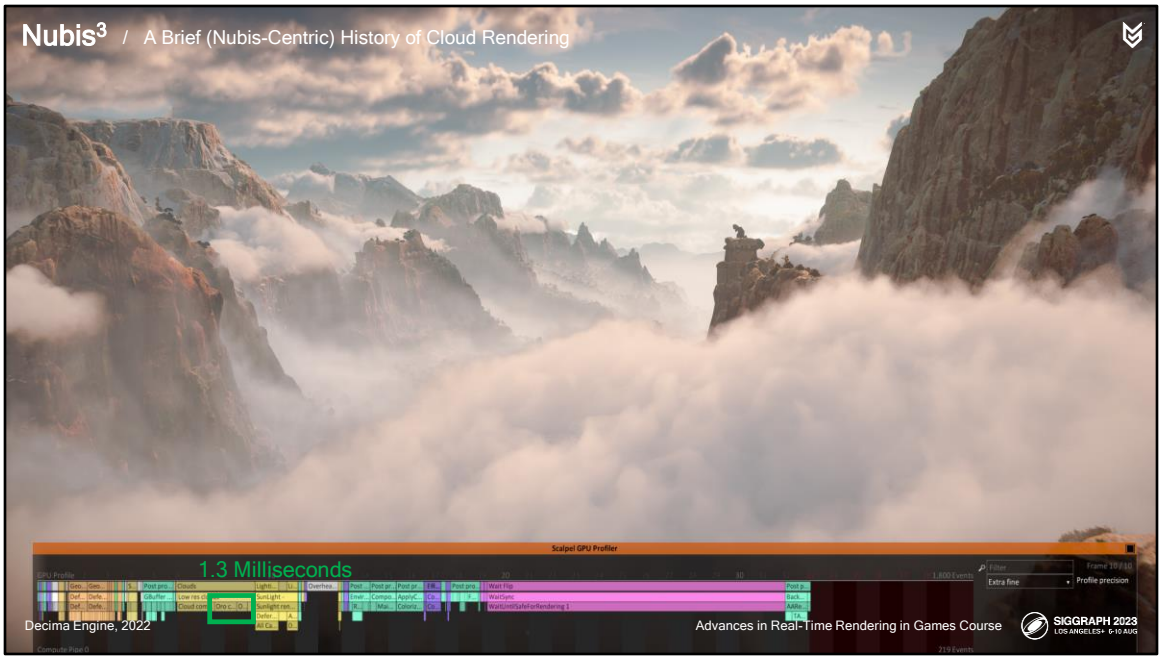
When raymarching we did an intersection test between the ray and cones generated from surface intersections to determine the largest step we can take before hitting any clouds. The distance step mapping was used to make sure that we did not take too many samples for areas outside of these cones and outside of clouds.

- As we began to sample inside the envelope, we'll start to take smaller ray-march steps and sample the actual cloud density.

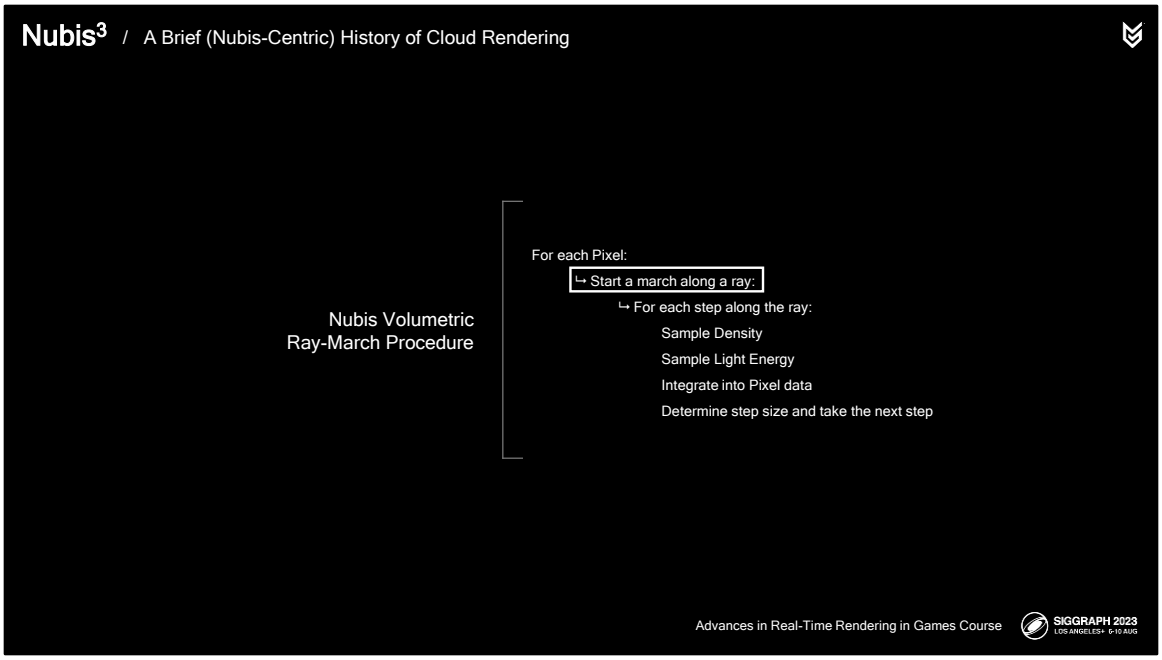
In this image you can see that the samples, as indicated by the red dots, are focused around clouds and not as frequent in empty space, which is a better distribution.



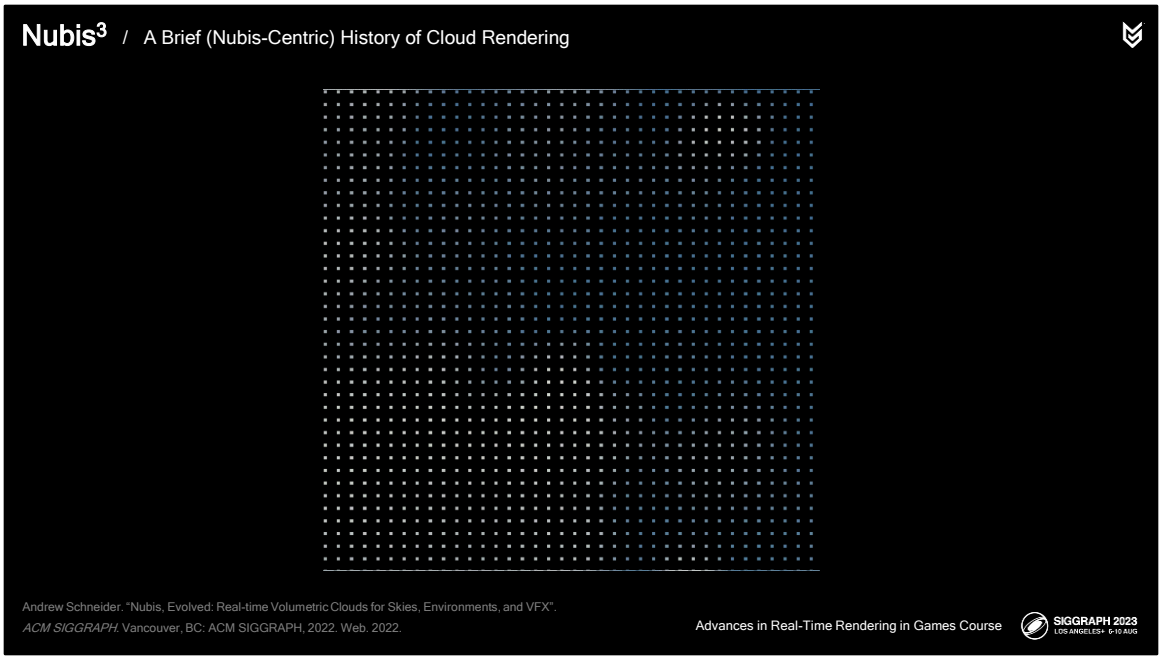
But how much could this save? Here is the cost before… 4.2ms



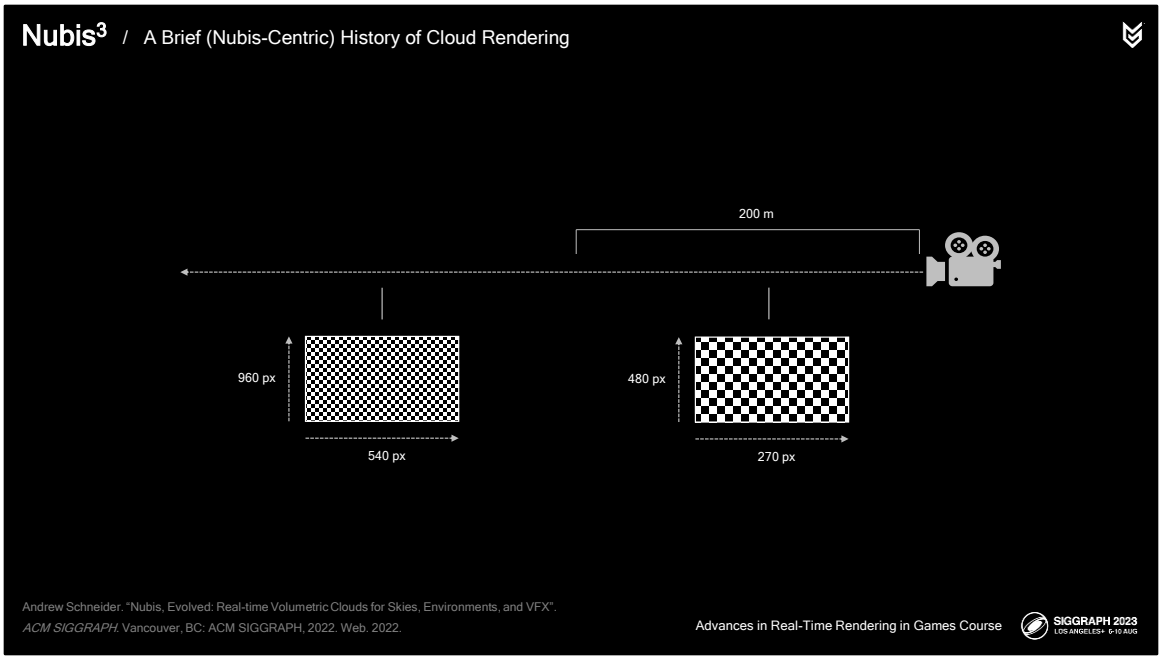
And a more comfortable 1,3MS after



Another way we made these systems as performant as possible was to do more than just fire off a ray march for every pixel. Both approaches to clouds employed some rendering accelerations.

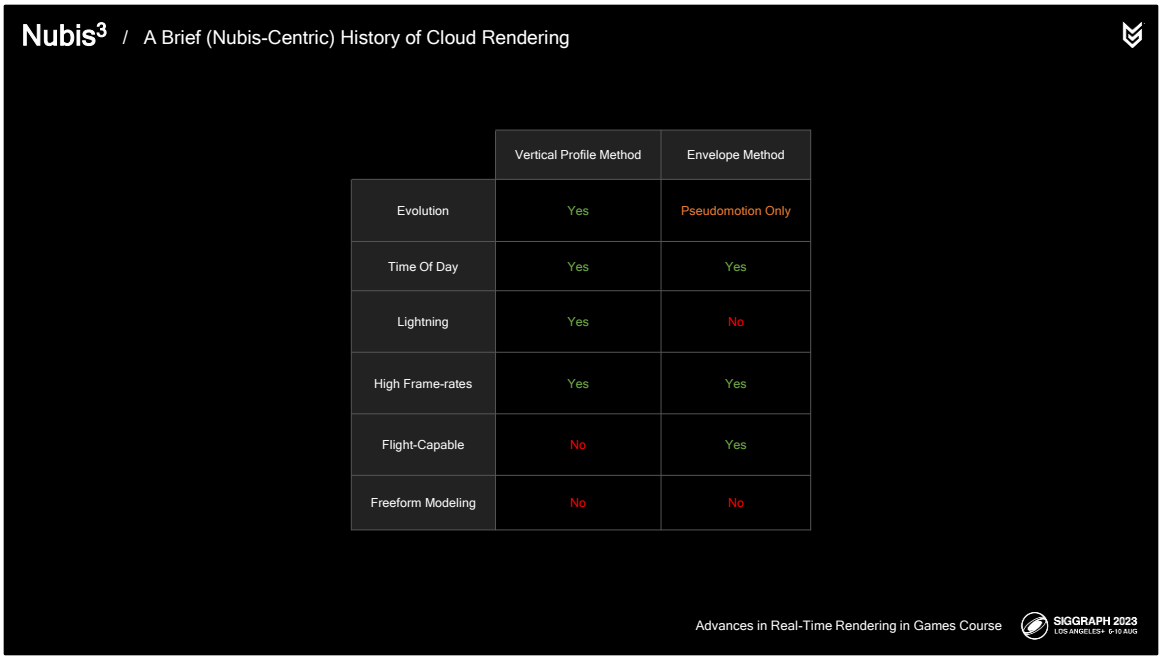


Vertical profile clouds used temporal upscaling which Amortizes the cost of the render over 16 frames.
This had the effect of making a 20ms render a 2ms render and worked for distant clouds but not for nearby clouds because the image would not be able to resolve in time when the camera moves quickly.



So, for immersive Envelope clouds, we instead

- split the render into two passes: Lower resolution near camera where rendering is more expensive, and higher farther away where we want to prevent aliasing.



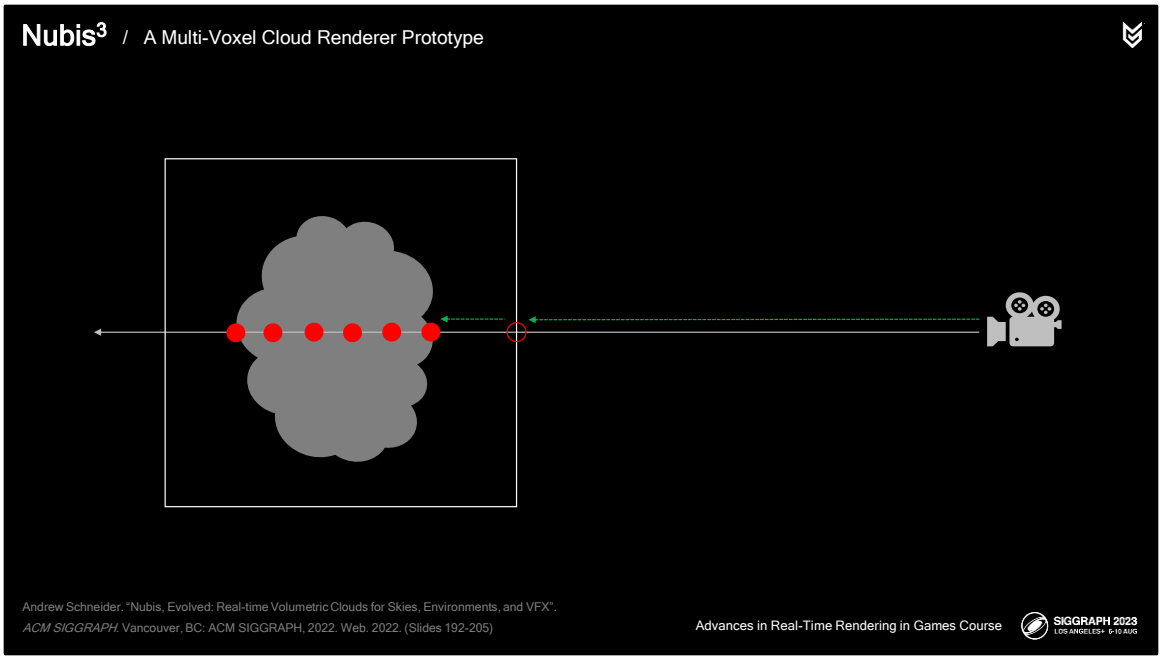
Ok, so we have now covered our previous methods for cloud modeling and rendering. Lets see how they compare:

Evolution isn't really supported by the Envelope Method because we didn't have the budget to create the cones and distance fields on the fly.
Both work for time of day.
Lightning was only implemented for the Vertical profile Method.
Both support high framerates.
Only envelope clouds are flight capable.
And unfortunately, none of these methods are very visually intuitive to author and work with. Let's be honest. Those of you who have worked with these methods know what I mean.

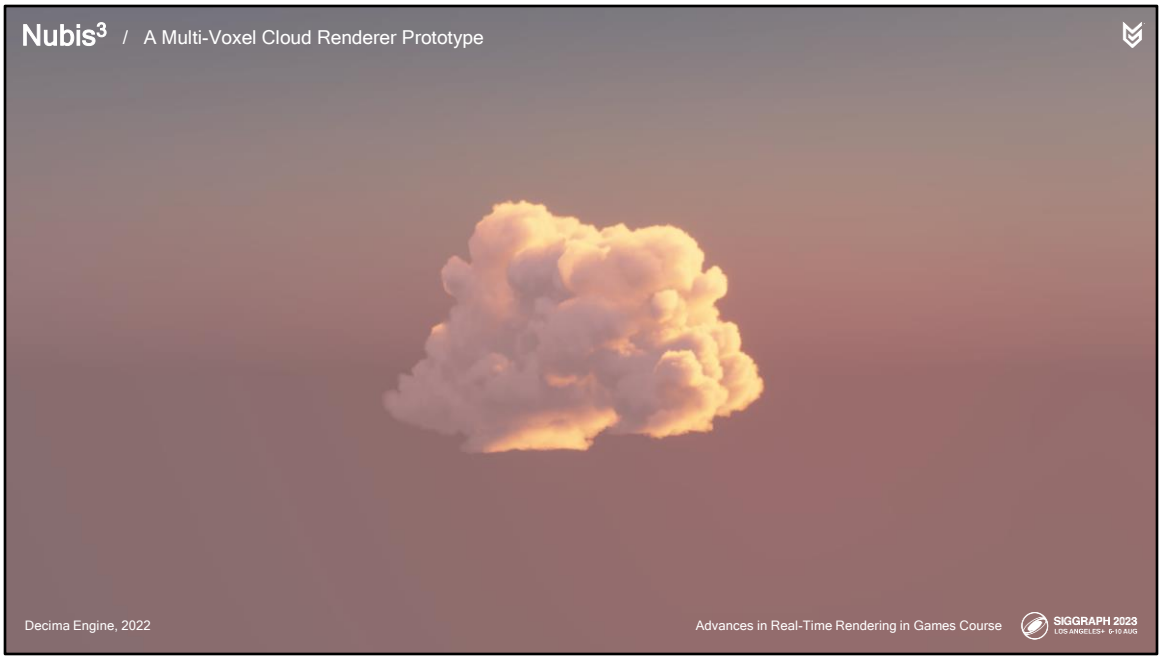
Both of these solutions held us back from achieving a goal that Managing Studio Art Director JB gave me back before I even worked at Guerrilla: flight through (what he called) proper clouds.

So, after we finished Forbidden west, frustrated and eager to prove what we could do, I started developing a prototype. I ended my talk in this course last year with a

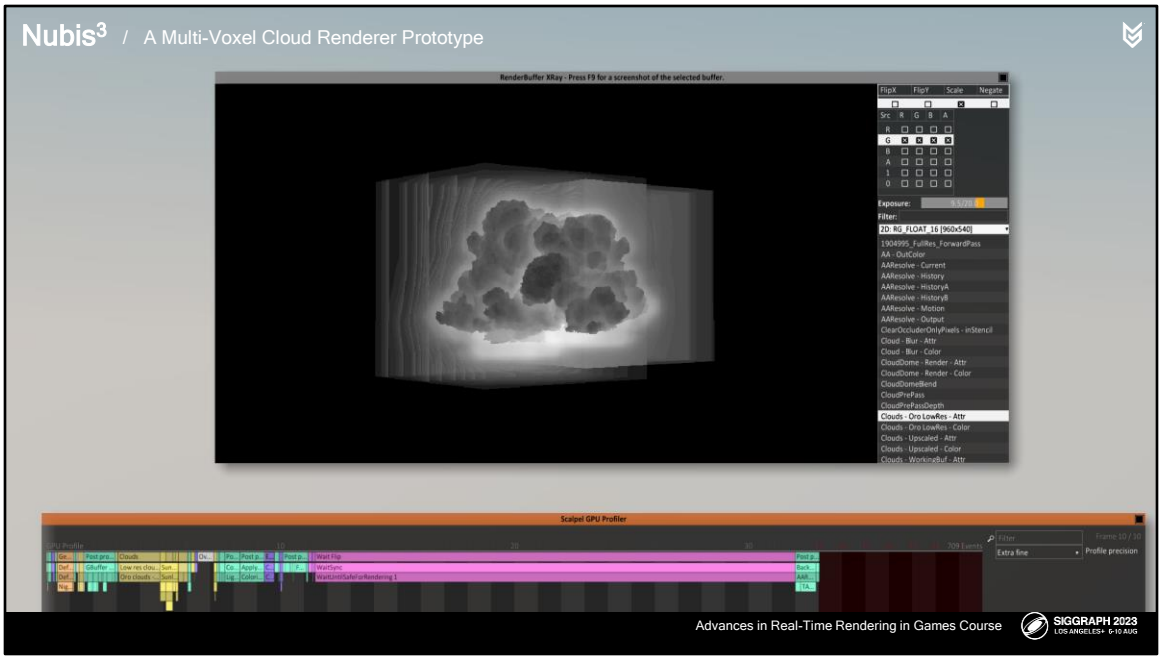
glimpse at this prototype real-time voxel-cloud renderer.



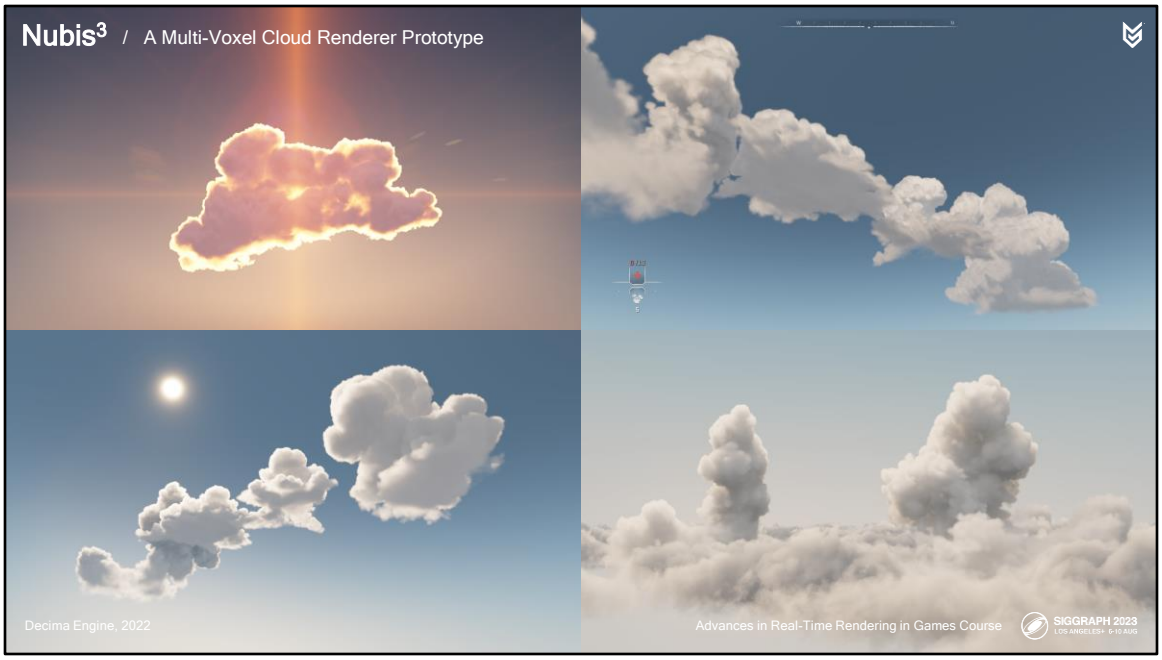
Instead of placing ray-march samples with no idea of where clouds are, it combined bounding box and internal signed distance fields into a source-agnostic step placement method that supported…



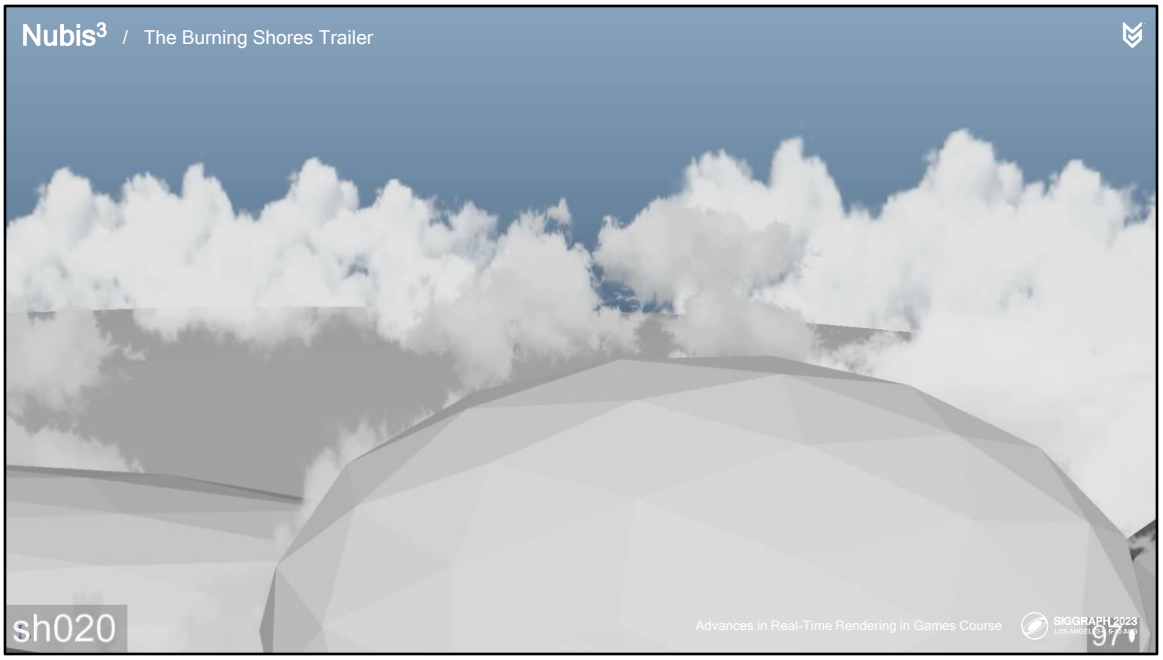
…overlapping volumes in both sample placement and density and light sampling. Here you can see a rendered result of 3 overlapping voxel clouds….



And here you can see the samples taken in white using this hybrid sample placement method.
I iterated and iterated on this until nothing more could be done to make it faster by myself alone.

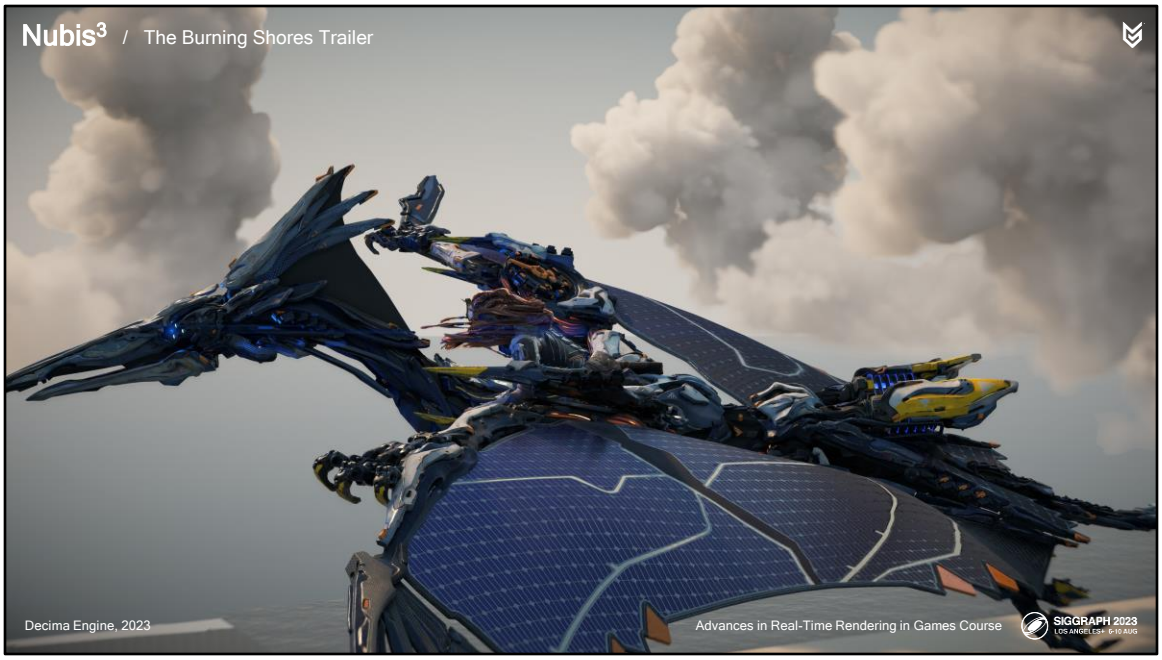


The goodish news was that we could combine a few clouds, fly into them, and the result would cost 4ms at 960x540 resolution on the Playstation 5.
The bad-ish, but ultimately fortuitous news was that I made the mistake of showing this to Jan-Bart van Beek , the managing studio art director. Now, recall that Jan-Bart mentioned this type of thing as a goal to me back before I even joined Guerrilla.
Soon after, I got called into a meeting about the trailer for our upcoming DLC.
I had heard that there would be some clouds in it.



Here is the animatic video that was shown to us. As you can see it is nearly all clouds as indicated by all the white ellipsoids - and not just clouds but high velocity flight through billowy detailed clouds spread between foreground and background as far as the eye can see.

After the meeting I ran this test to see how the prototype would perform in theses conditions and it was a bit concerning. Performance was terrible and the detail we would need just wasn't there.
Fortunately, at the same time, we learned that the Burning Shores DLC would be a PS5 only release, freeing us of a lot of performance restrictions.



I then sat down with Lead Tech Programmer Jeroen Krebbers and Principal Tech Programmer Nathan Vos to discuss how we could move this from a prototype into a shippable solution for the trailer. Let's just say that this was a tense discussion. And for good reason.

One of the things we left the meeting with was an agreement that if we did this for a trailer, then we had to do it in the game itself.

- We also agreed that if we couldn't pull it off, then the whole venture would have to be scrapped. With the help of my collages in Tech Code, Nathan Vos, James McLaren and Hugh Malan we made sure that whatever we showed in the trailer could be optimized further into what we shipped for the actual DLC without a loss of quality. Here is what we came up with:


Nubis3 / The Burning Shores Trailer
16
TM
www.pegi.info
Decima Engine, 2023
Advances in Real-Time Rendering in Games Course
SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

trailer

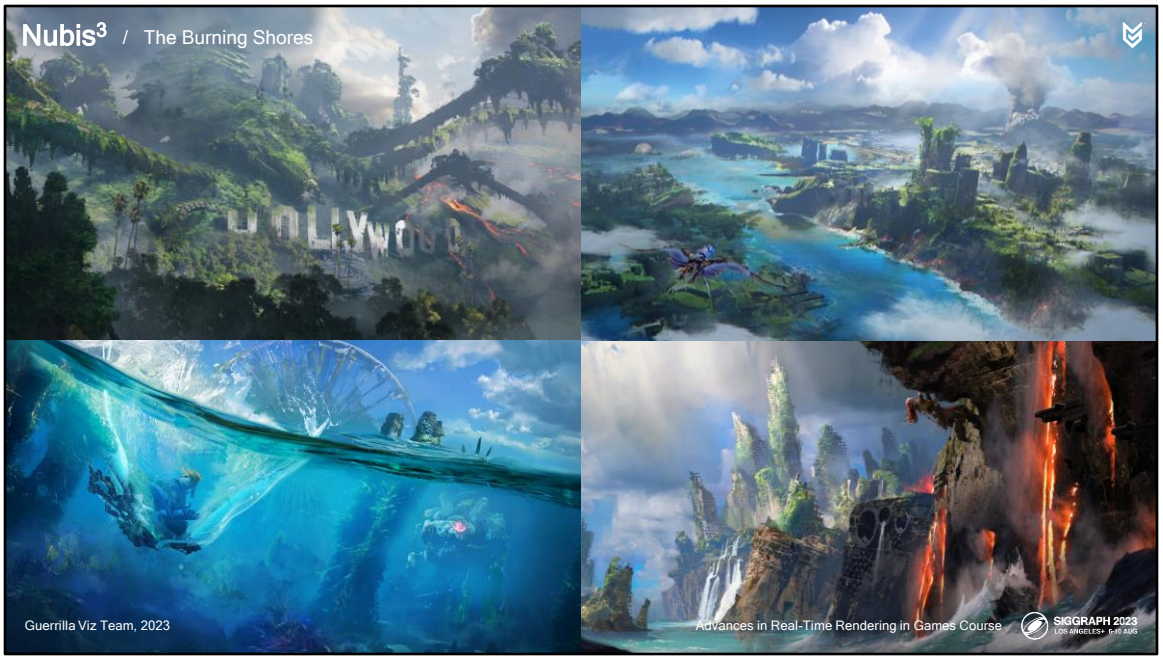
A few takeaways from that experience: I think it was when I was able to produce an expansive cloudscape with this much variety for this shot, that I realized we had a system that was going to work one way or another.

And when we were reviewing this shot, which was supposed to take place a kilometer in the sky, someone asked if I should hide

- that 200 meter mountain top with clouds. The art director said “No. Leave it” I realized then, that Our original intentions didn’t matter as much as producing something that made a statement. I realized that we had a reached a point where Art direction was getting comfortable with using clouds as one of those foreground tools for building depth.

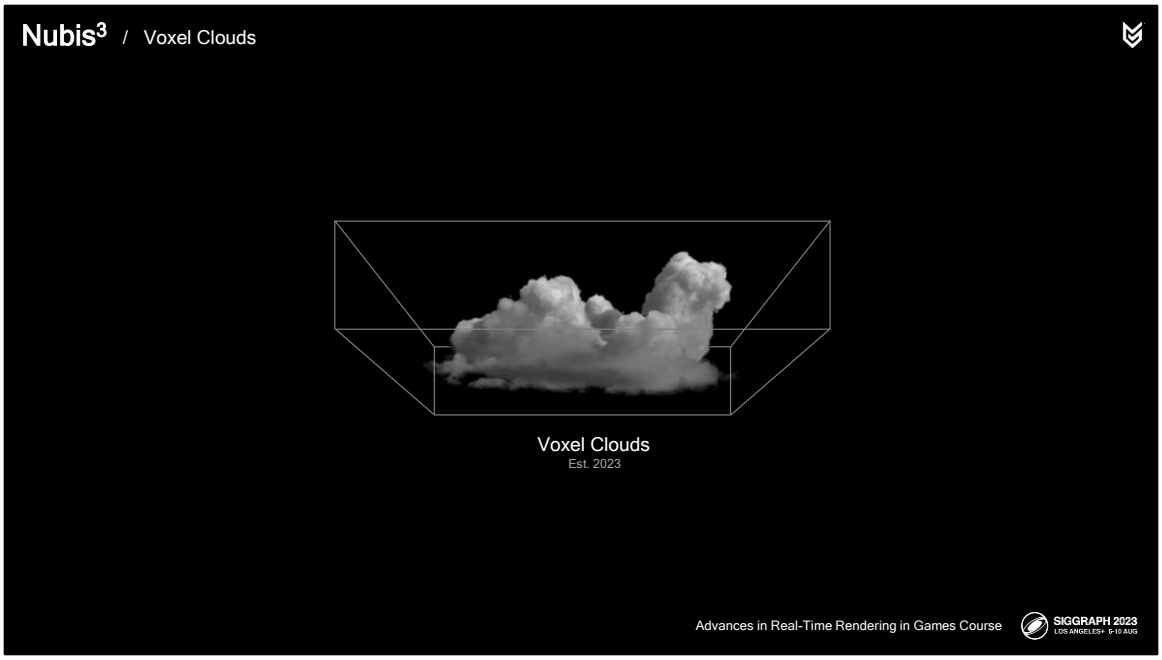
Also, When we looked at reaction videos to the trailer, People mentioned the clouds but there was not any talk at all of what it would mean for gameplay – meaning that a lot of people who viewed this expected that this was just a cinematic trailer.

That was exciting and also a bit terrifying ☺



The Burning Shores DLC itself was set in the ruined volcanic archipelago of Los Angeles.
True to Hollywood, the goal of the project was to deliver bigger than life experiences.
So naturally, with this new tech in development, I wanted to open

the skies for exploration and not limit that experience to a transitional cinematic between the base game and the DLC as was originally planned.
This meant that not only would the clouds need to serve in their traditional capacity as detailed volumetric skyboxes, but the player would need to be able to fly from the ground into the highly detailed billowy clouds and then dive into the water – all without any seams.
And, as it turned out later, there would be something even more challenging. More on that later.

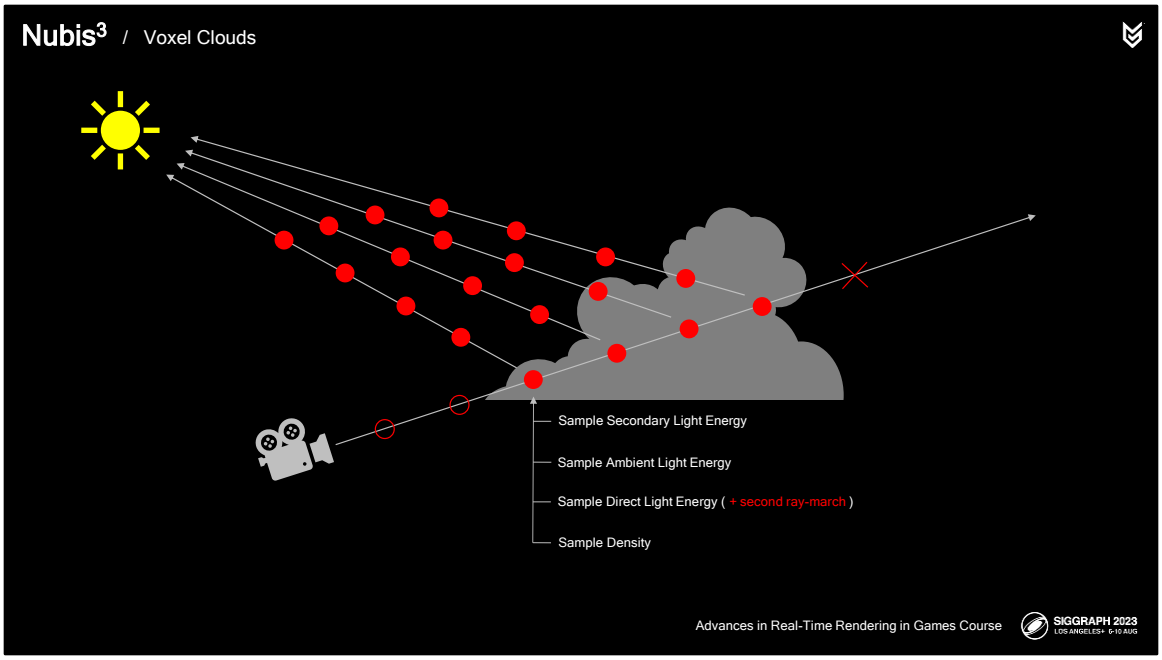


It took a lot of fast paced development in just a few months,

- but we managed to pull it off.

So without further delay.

Here is our solution for immersive real-time voxel clouds.



Recall our ray-march procedure

- We march along without much knowledge of where the clouds are.
- When we hit a cloud we sample density
- direct light energy – which requires an annoyingly

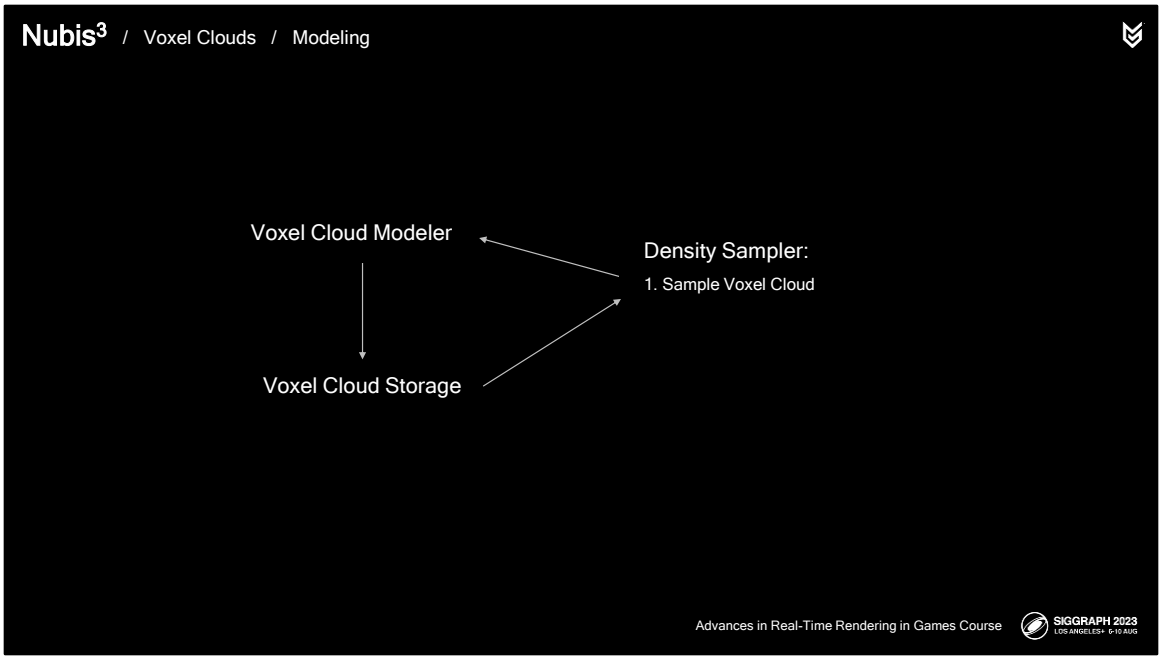
expensive secondary ray-march,

- Then ambient light energy,
- And finally light energy from secondary sources like lightning.
- Then we repeat this for every other uninformed step until we reach full opacity and stop the march.

None of these operations are terribly efficient or were designed to work efficiently with voxels, but our approach to voxel cloud rendering replaces and optimizes almost every one of them with faster better voxel–based methodologies. Let’s start this

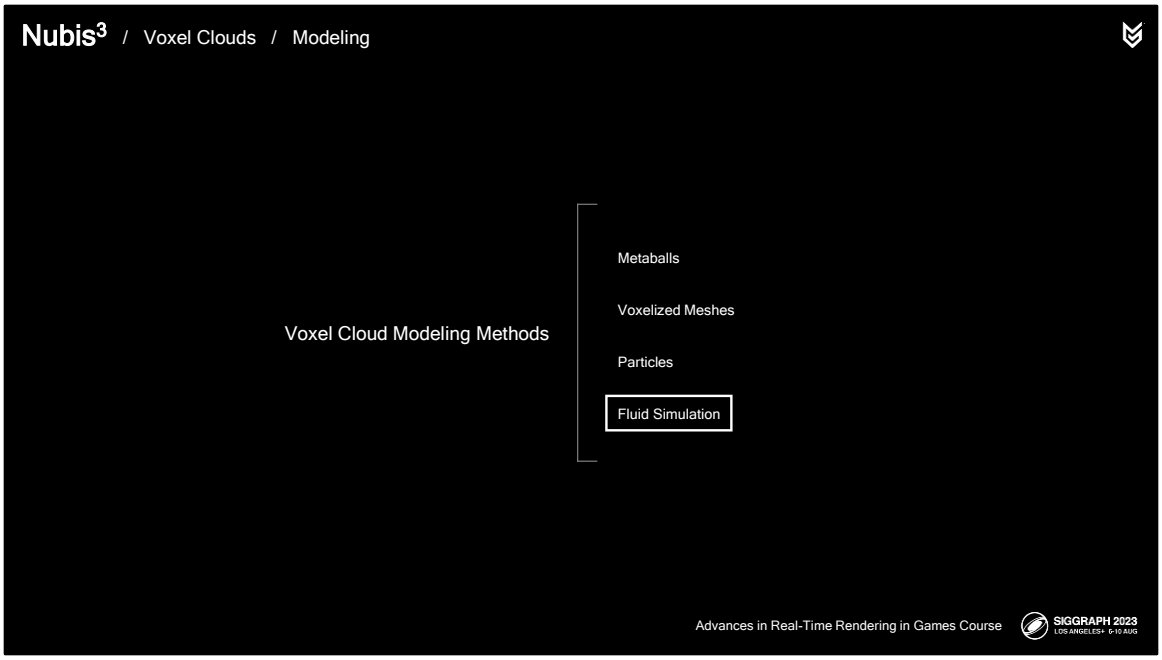
overhaul at the beginning.

Developing an efficient rendering and lighting solution depends on solid modeling foundations because for AAA real-time graphics so much of what you decide to do here and how much memory it all takes can either help or hurt the performance.



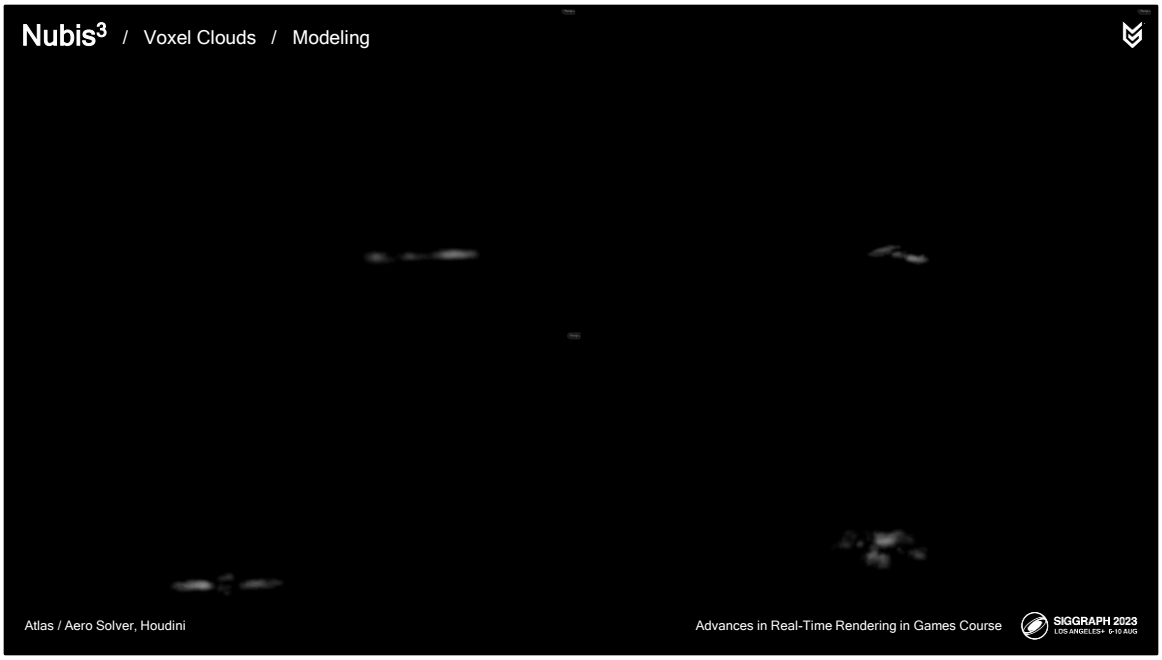
Recall that the density sampler for the 2.5D clouds first constructed a 3D cloud volume from 2D modeling data before up-rezing it. In a voxel-based approach,

- this entire expensive operation can be moved out of the shader, but this means you need to
- develop a way to model voxel clouds,
- store them, and then
- access them in the sampler efficiently.

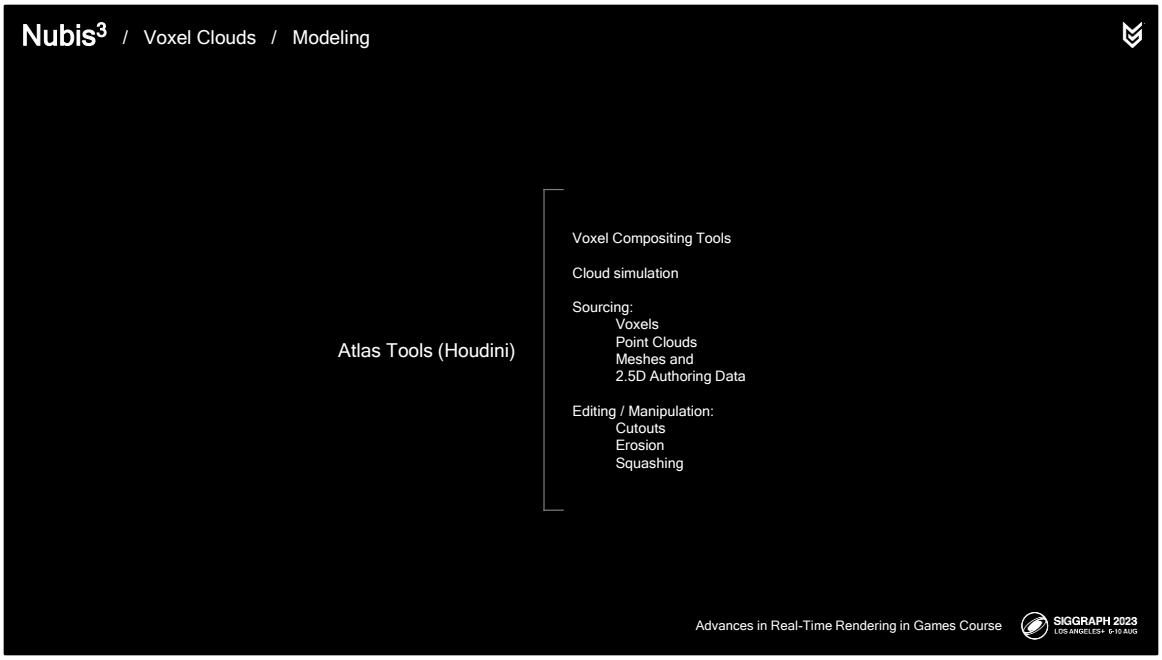


There are many tedious and unrealistic ways to model voxel clouds by hand but the most promising approach that I found the most success with in

- the past was using fluid simulation to “grow” clouds.



It seemed like un-shelving our 2014 fluid simulation experiments, which I kept because I’m a data hoarder, and doing some “frankenclouding” would be the best place to start modeling voxel cloudscapes for the Burning Shores. But we needed a way to work with this data to build cloudscapes.

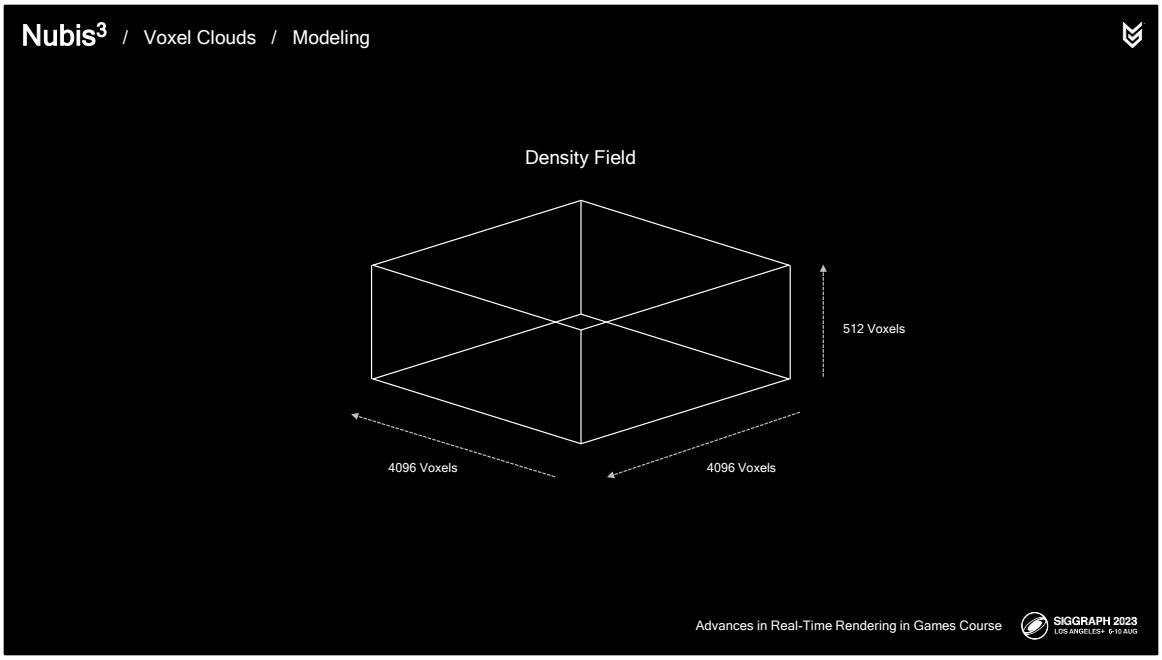


We developed a set of authoring tools in Houdini, that we call Atlas. At its heart, Atlas is a compositing pipeline for voxel data with various ways to generate or source and manipulate the data.

As an example, one of our Atmospherics Artists, named Bryan Adams, created a tall neck cloud from the game model.

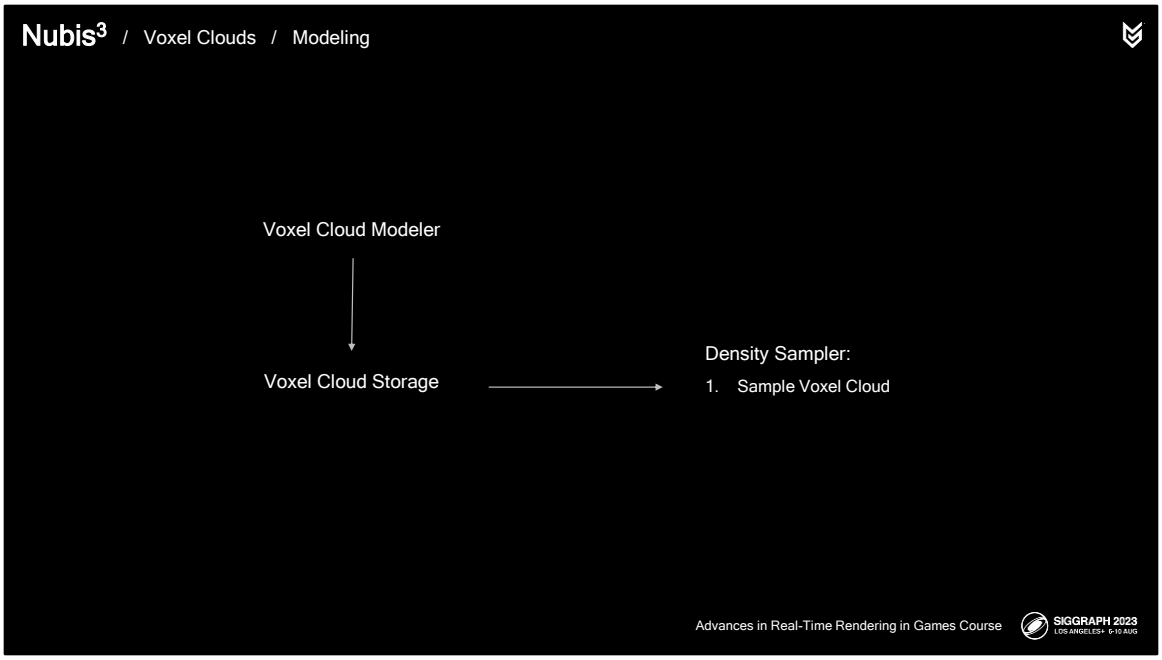
One of the things we learned early on was that multiple voxel grid boxes bounded us both figuratively and literally.

Keeping them in memory and switching between them in the inner loop of a ray-march was not ideal and lead to slow performance.

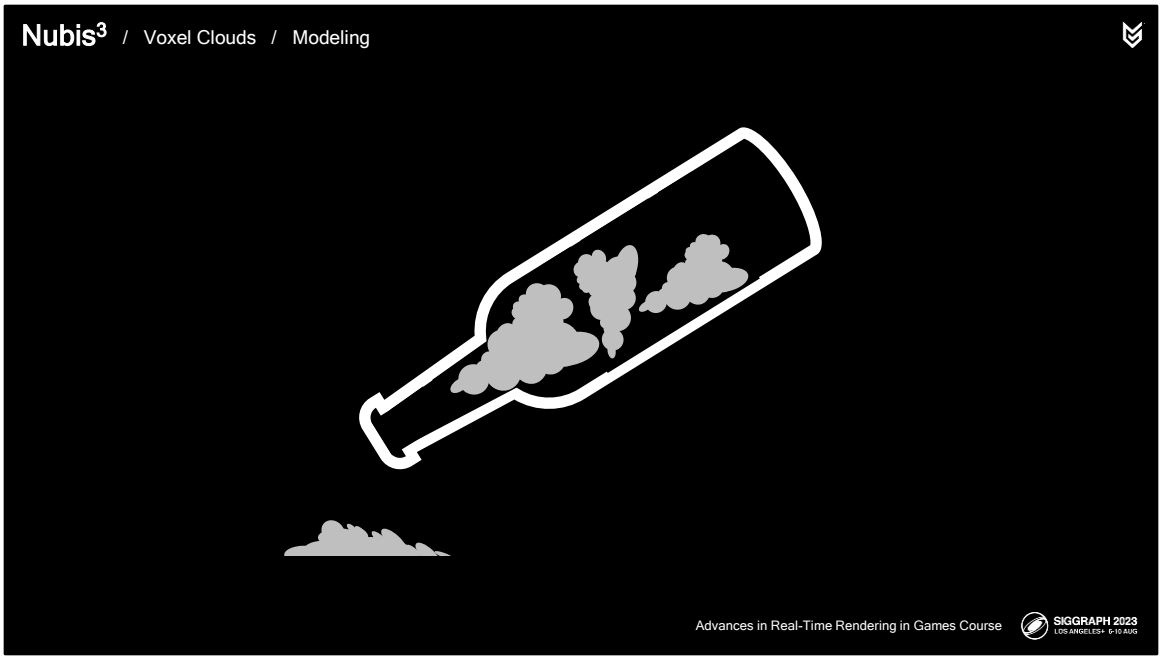


So, the next idea was that after the entire cloudscape was constructed, we write out a dense voxel grid

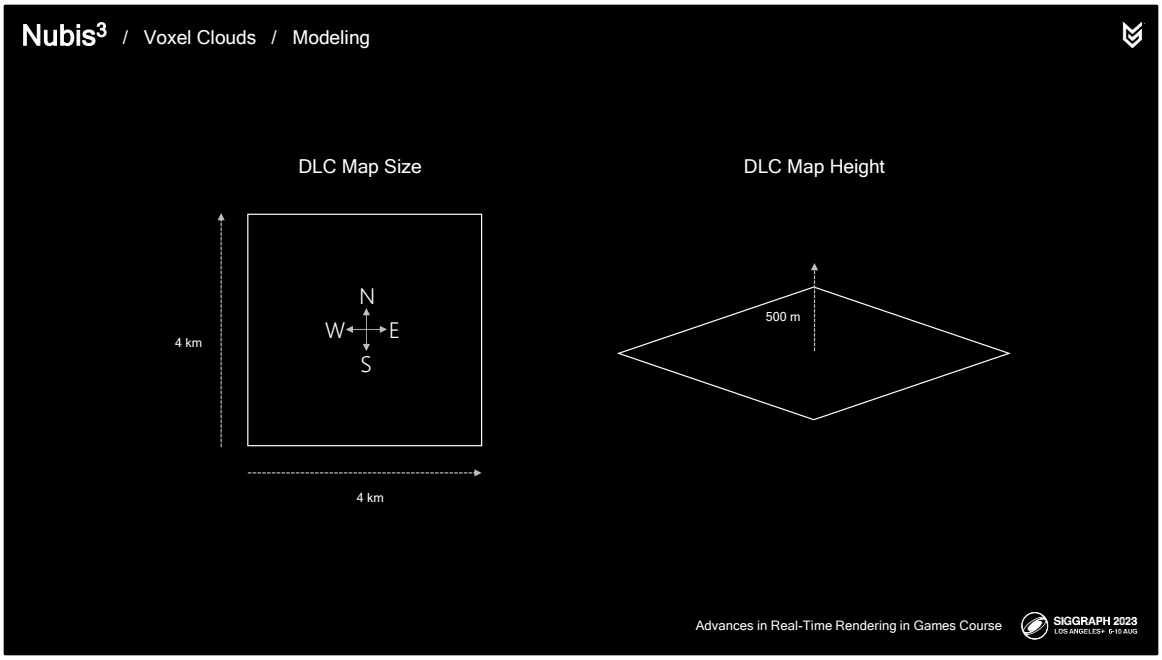
containing cloud density information for every cubic meter space in the world.



This would solve the first challenge of modeling voxel clouds and completely replace an expensive part of the 2.5D Cloud density sampler – that is if it didn't send us crashing into the..

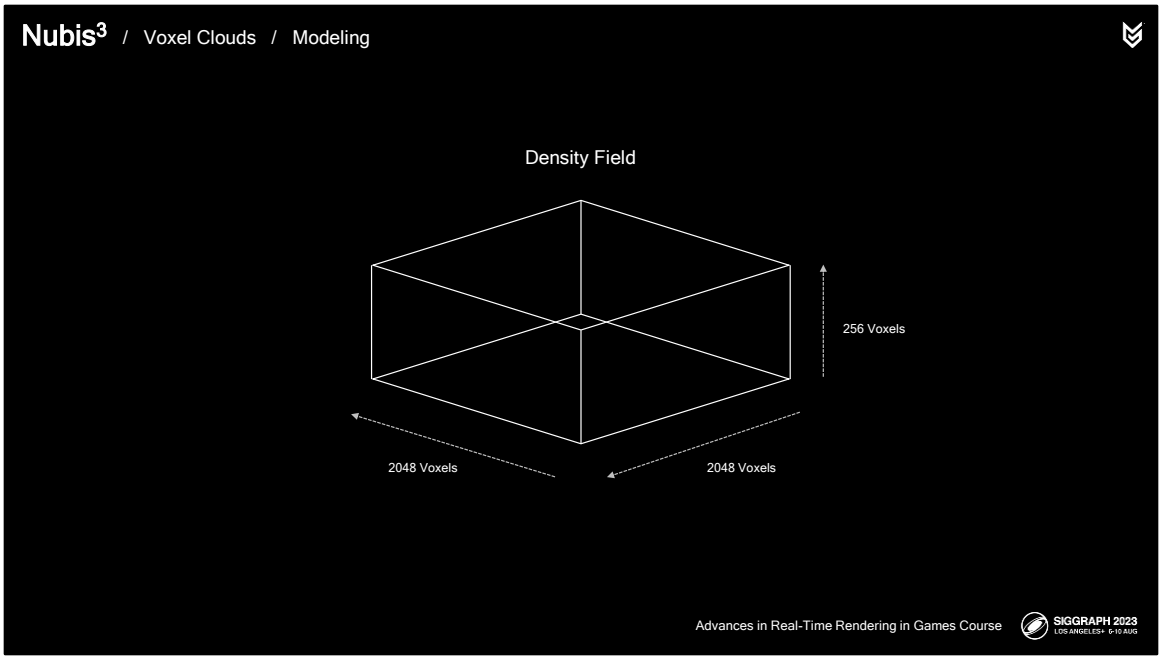


first problem that we knew we would face with sky scale Voxel grids. Memory bottlenecks.



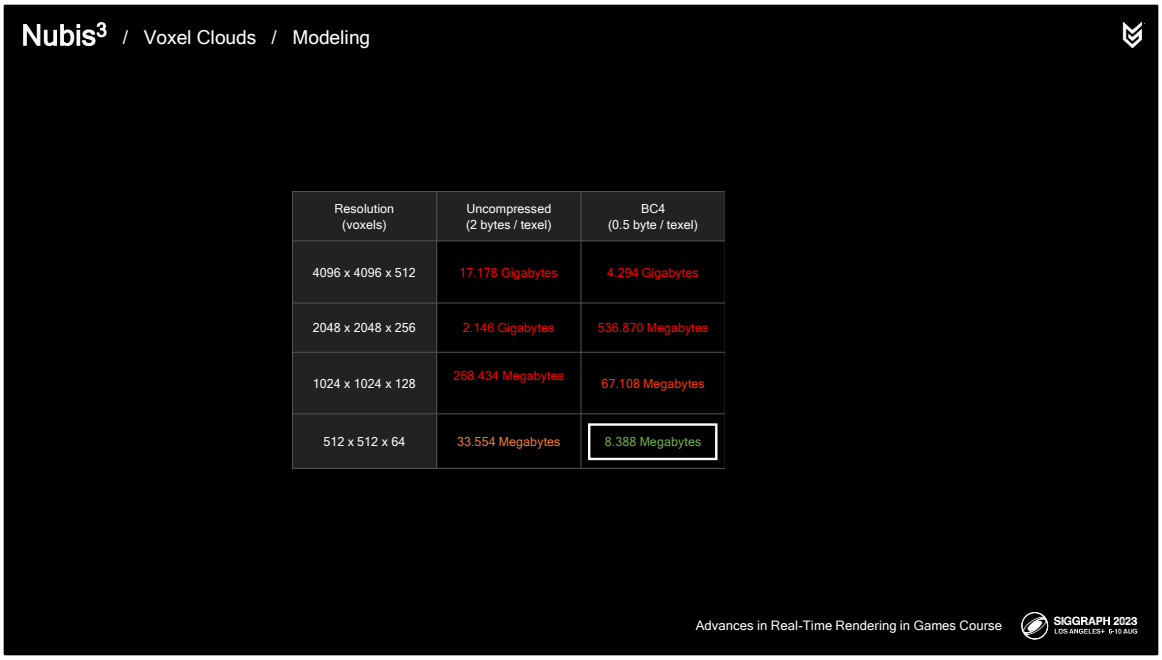
I'll explain. The DLC area was about 4km by 4km and we

- needed about 500 meters of vertical space to place clouds that would work from the ground and in the air. We also needed enough resolution in the voxel grid to capture fine details as the player would fly through clouds.



If you took expectations down a notch… and then do the math… you would need a voxel grid of

- 2048x2048x256 in order to get 2 meter precision. The Flying mounts have a wingspan of about 2 meters so this was barely enough by our estimation.



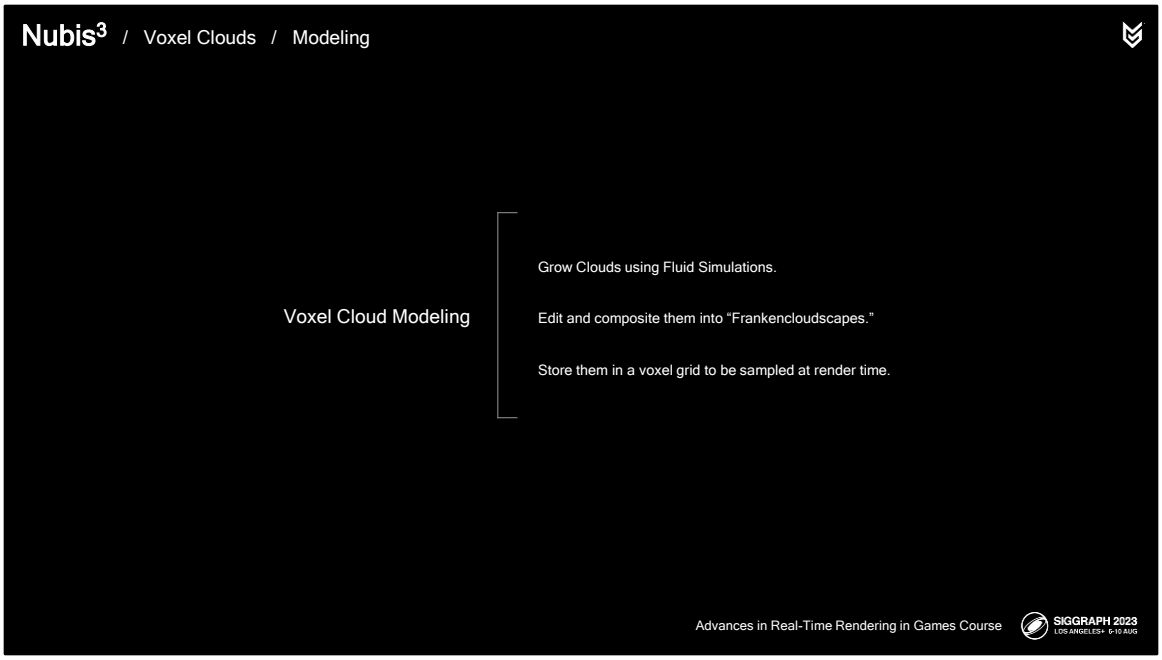
Grids of this density take a lot of memory and, consequently take longer to access for samples in a ray-march because the bandwidth requirements increase exponentially. Early experiments showed that this was simply not going to work for us.

- There are various sparse voxel formats that hold higher precision data only in areas of the grid that have something in them, but this requires an indirection step, which raises the cost of taking each a sample.

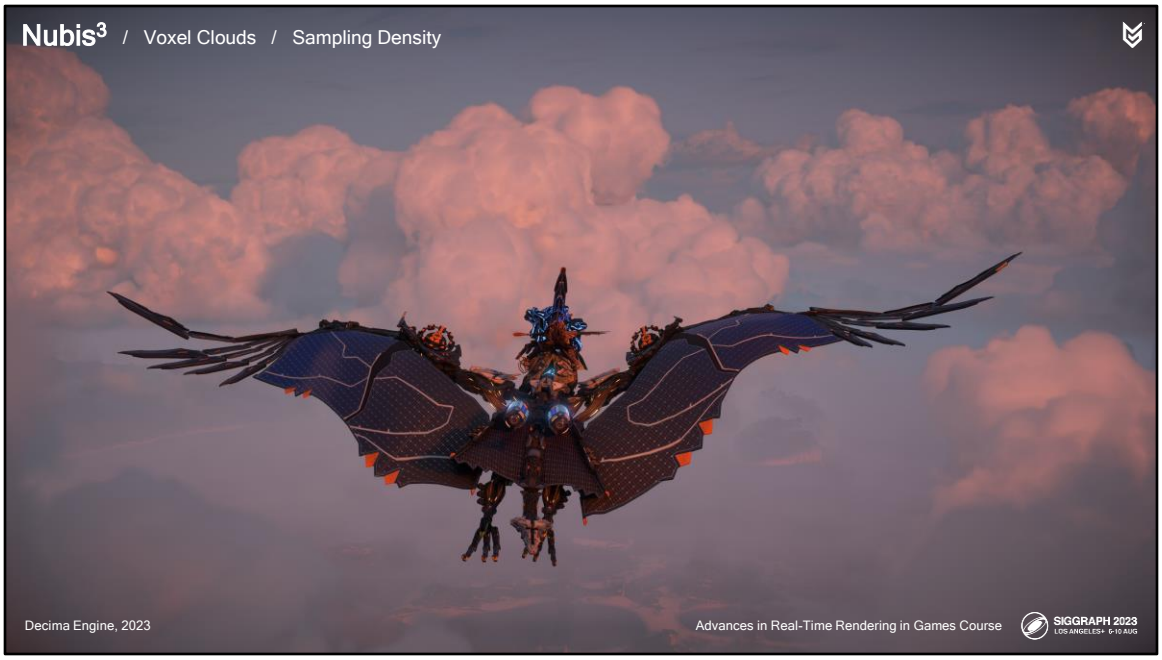
For a ray-march that's this intensive performance is critical..

Given the mountain of known problems we already had to face in an insanely small time window, we decided not to add this unknown to the pile. This could be a good area to explore in the future though.

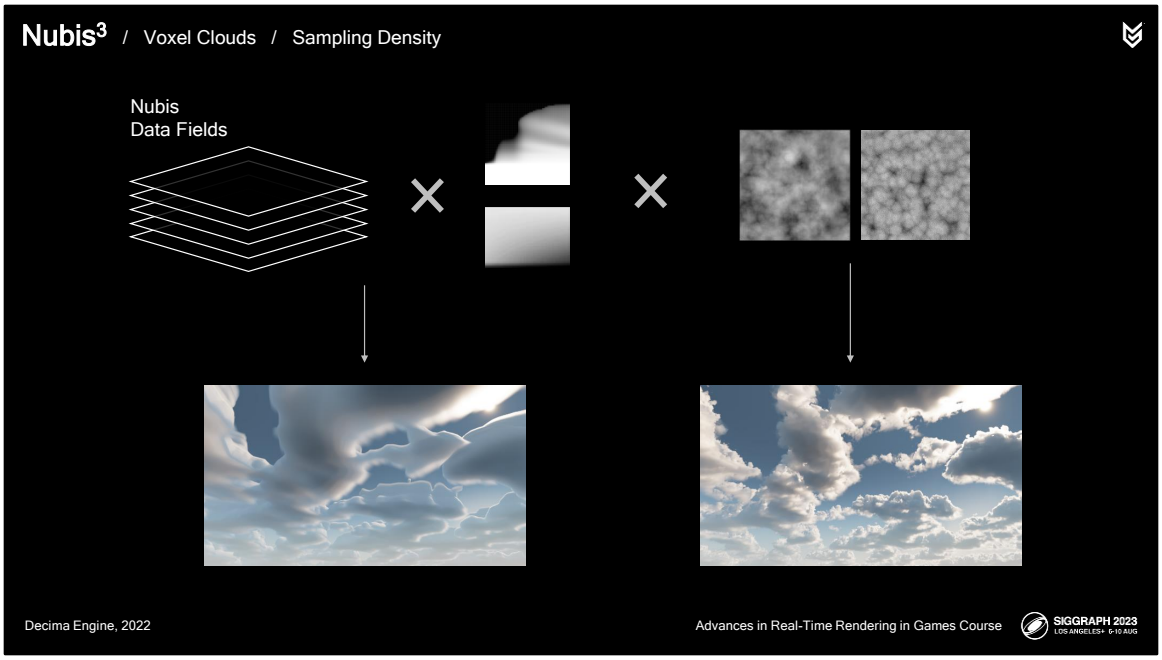
- We decided to start with BC4 compressed, 8 meter precision.



Let’s sum up what we know about our approach to voxel cloud modeling.
We Simulate clouds using a custom Cloud solver called Aero in Houdini
We then edit and composite them into frankencloudscapes and
Then store them in obscenely low resolution voxel grids to be sampled at render time. It sounds like its hardly a solution to our goals, but…



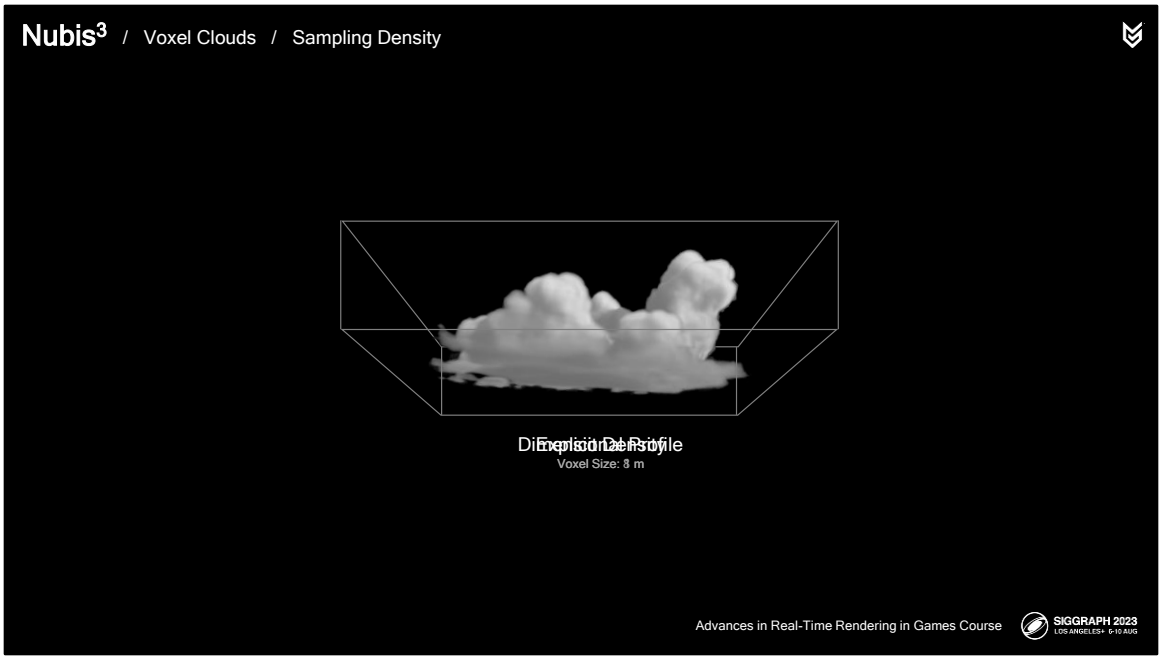
Past experience had taught us that balancing work between memory accessing and instructions in the compute shader can yield better performance on the GPU, so we chose to solve this in the density sampler itself as we have done in the past.



Recall that the previous approaches

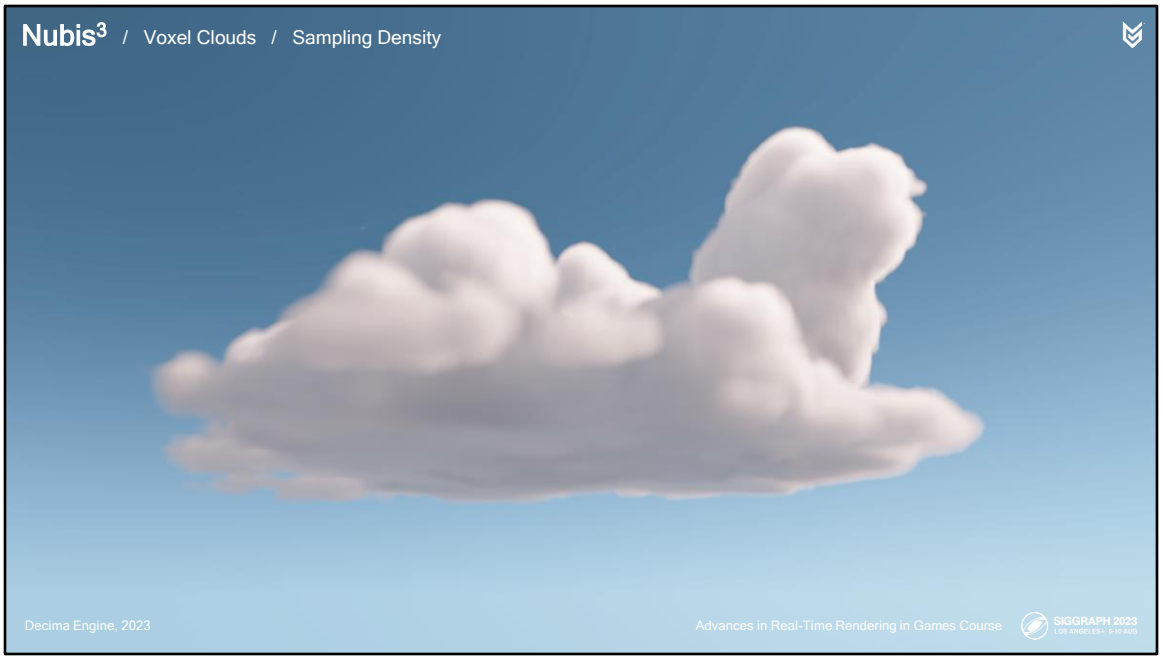
- decompressed their 2D NDF modeling data into the dimensional profile. Also recall that
- we then up-rezed the dimensional profile by eroding it with higher frequency noise. The question is obvious. Could the up-res approach work when sampling voxel

data in a volumetric ray-march? The answer is yes.



Rather than defining cloud density explicitly in every voxel at high resolution,

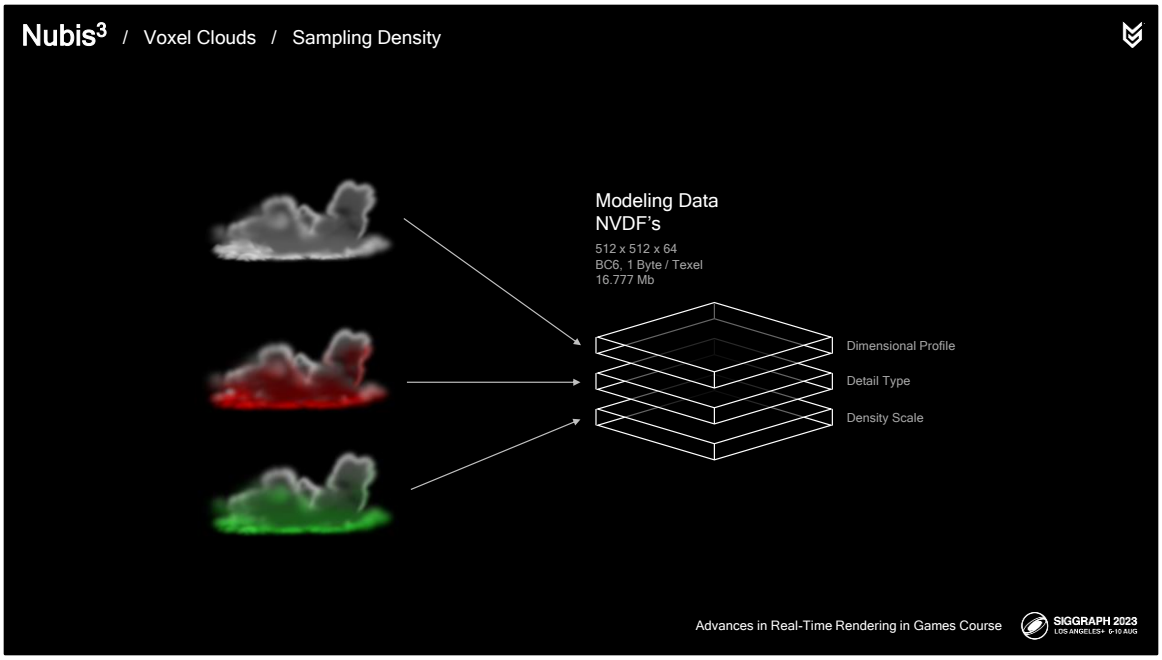
- our approach is to up-rez a lower resolution voxel-based dimensional profile using new detail noises. This allows us to avoid memory bottlenecks by shifting the work to up-res instructions as we did with the 2.5D model clouds.



Just to give a preview of what I mean: Here is what the dimensional profile looks like when we render it

- And here is the result of the up-res.

The Dimensional Profile is generated from a signed-distance field of the cloud to ensure that we get a gradient from outside to inside.

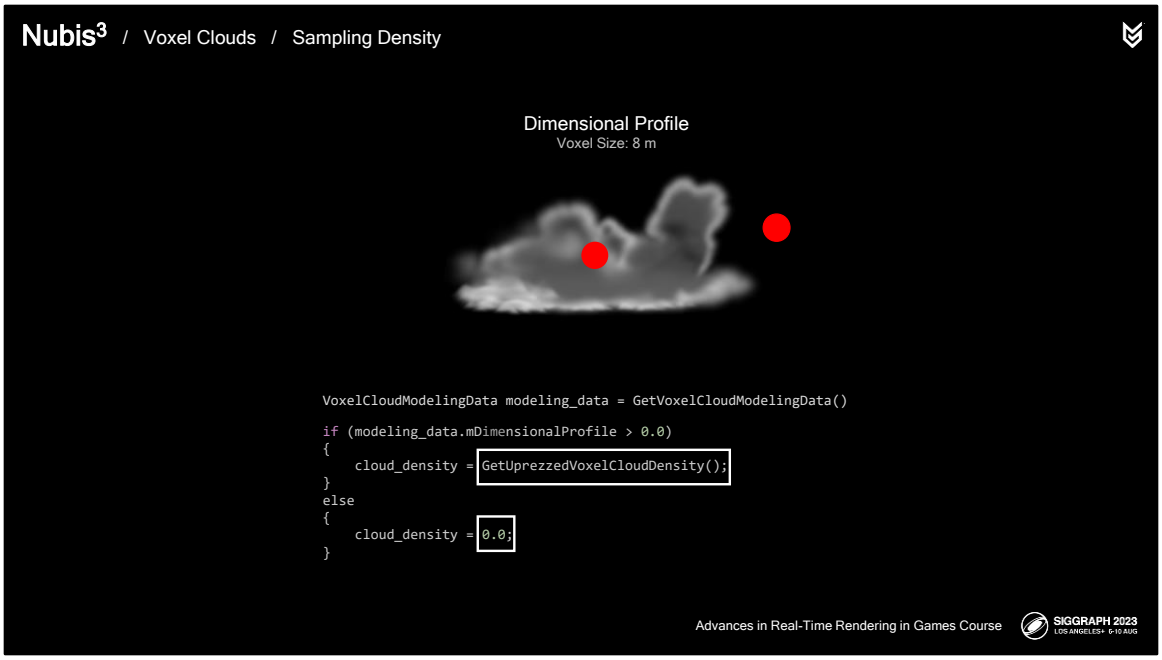


Our Atlas tools allow the user to model What we call Nubis Volume Data Fields, or NVDF's. In addition to the dimensional profile, the user can create additional NVDF's like:

- Detail Type, which is used to blend from wispy to billowy noise details
- and Density Scale, which is used to decrease density after the noise was applied in the up-rez.

These were compressed using BC6 Compression, allowing 3 channels with 1 byte per texel.

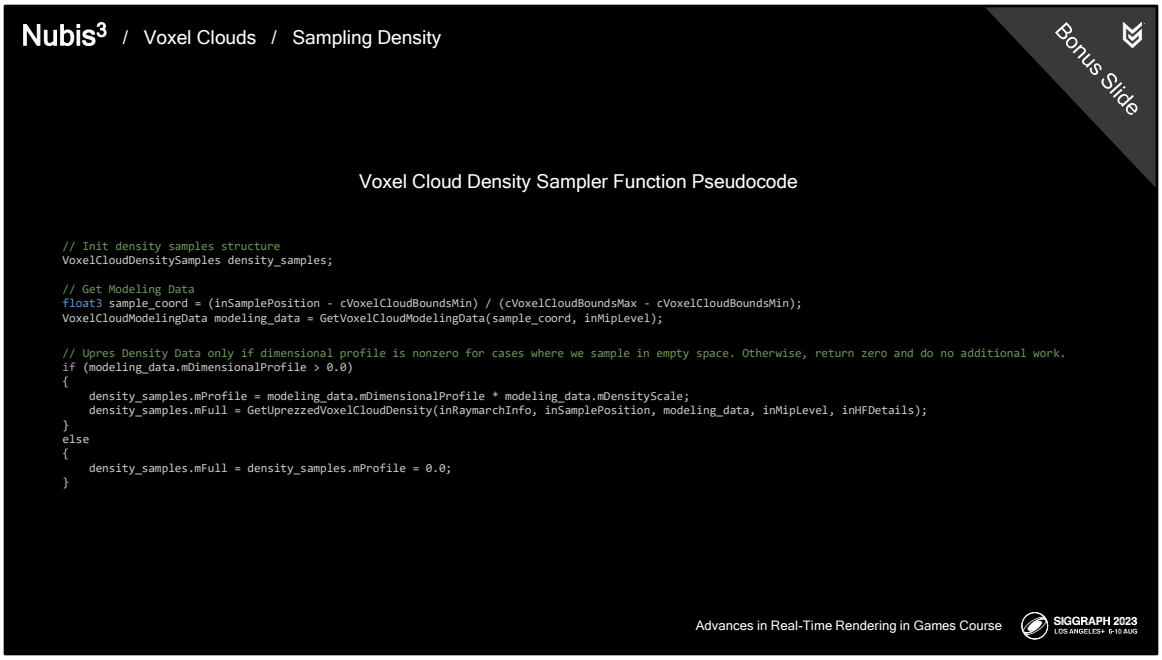
Let’s take a look at how the voxel cloud sampler function is implemented using our up-rez method, and lets look at our new flavors of 3D Noise.



The first thing we do in our density sampler is

- sample the dimensional profile NVDF. We do this first because if the result is zero,
- then we are not sampling a cloud and we can prevent any further work for this sample.
- If the dimensional profile is nonzero, then we pass the modeling data into the up-rez function.

Before we examine the up-rez function we need to take a look at the new 3D cloud detail noises that it uses.

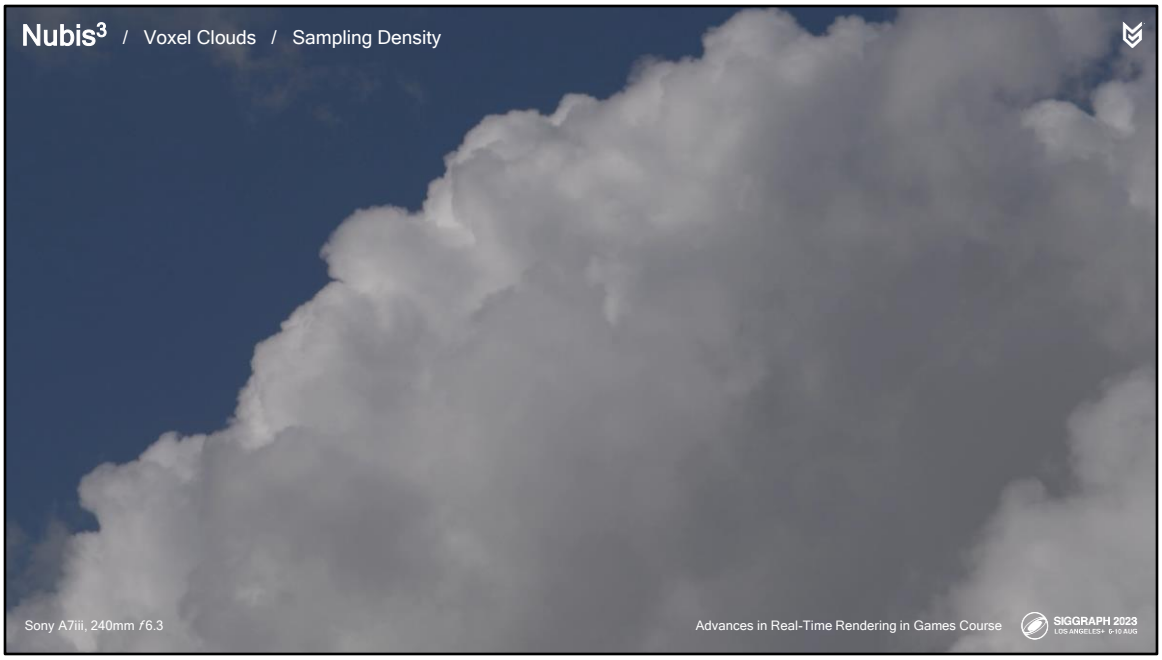


Notes:

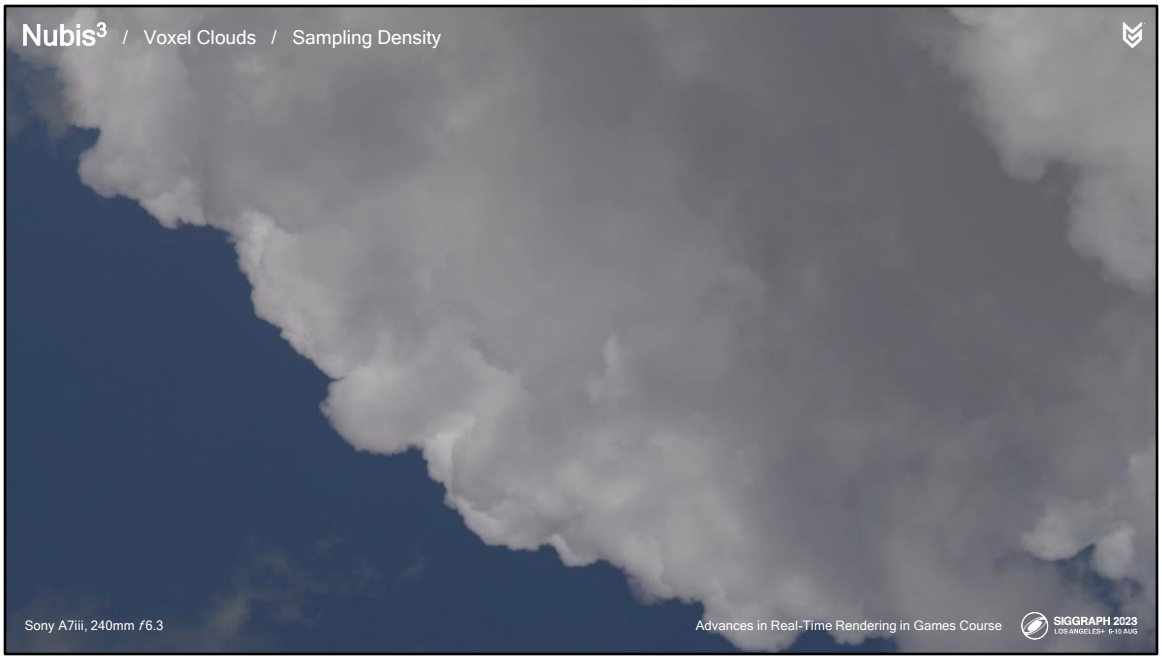
We define structures to hold the NVDF modeling data because they have a special ability: You can pass the populated structure into a function and members that are not accessed do not

contribute to VGPR usage. (if you are careful)

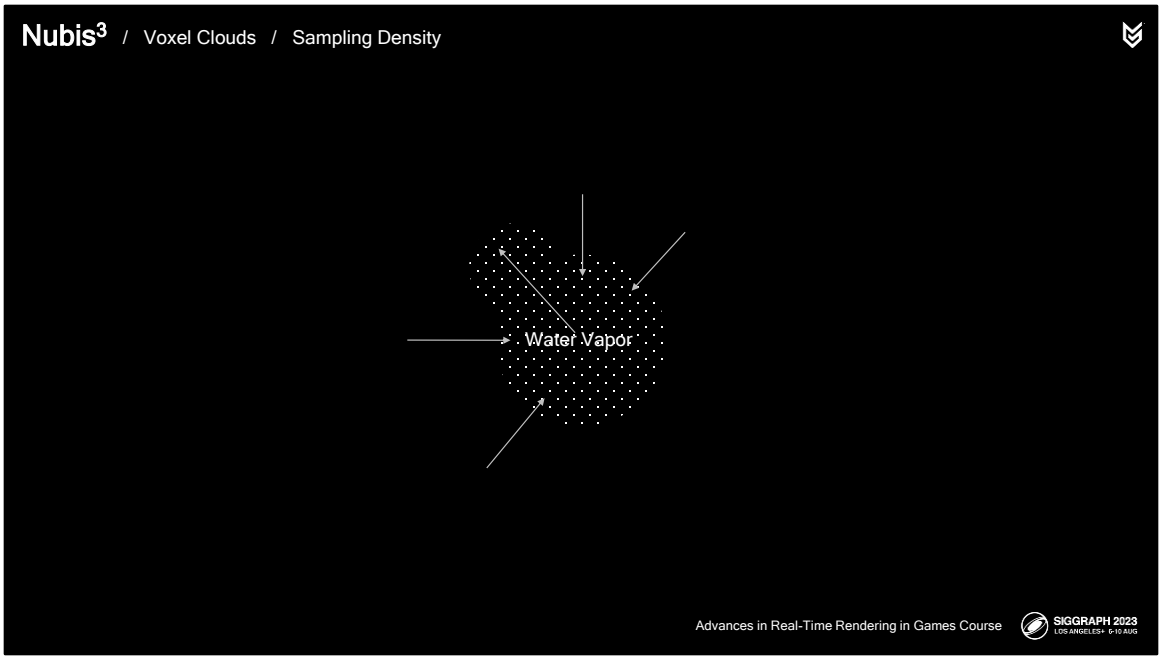
We store two density samples in a density sample structure. Full samples are for density and lighting calculations that require precision and Profile samples are for lighting calculations where we attempt to construct a light scattering volume. More on that later.



First, lets look at this timelapse. I should really have used a Tripod, but it gets the point across. You have all seen footage like this before. However, I find that to really understand Fluids, it is best to look at them from different perspectives.

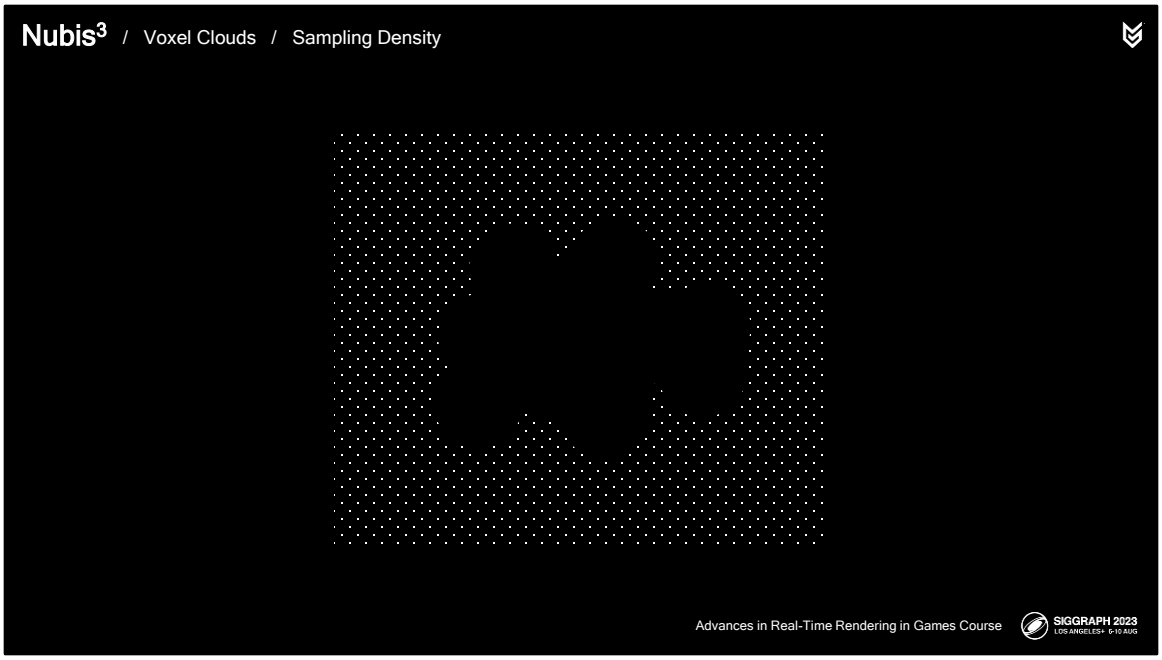


Especially upside down. If you walked by my desk at work you might see me looking at my monitor in an odd angle. From this point of view, it looks more like milk being poured into a tank of water with a blue background than actual clouds. From this perspective its easy to imagine the pressure that water vapor is fighting in order to get where it wants to go. That struggle is key to understanding how to model cloud details.



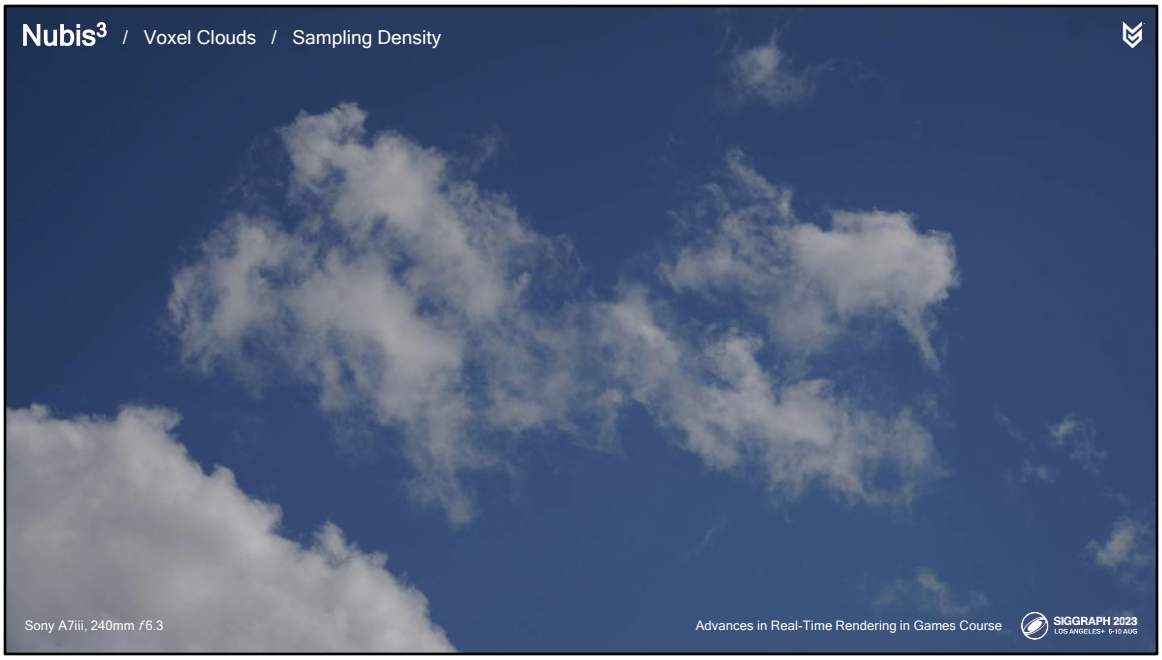
Recall that we classify cloud details as billows and wisps.

- As vapor moves into an area and encounters colder air it is effectively pushed back where it came from.
- The billows that we see are the result of some of that expanding vapor punching through weak spots in this squeezing force.

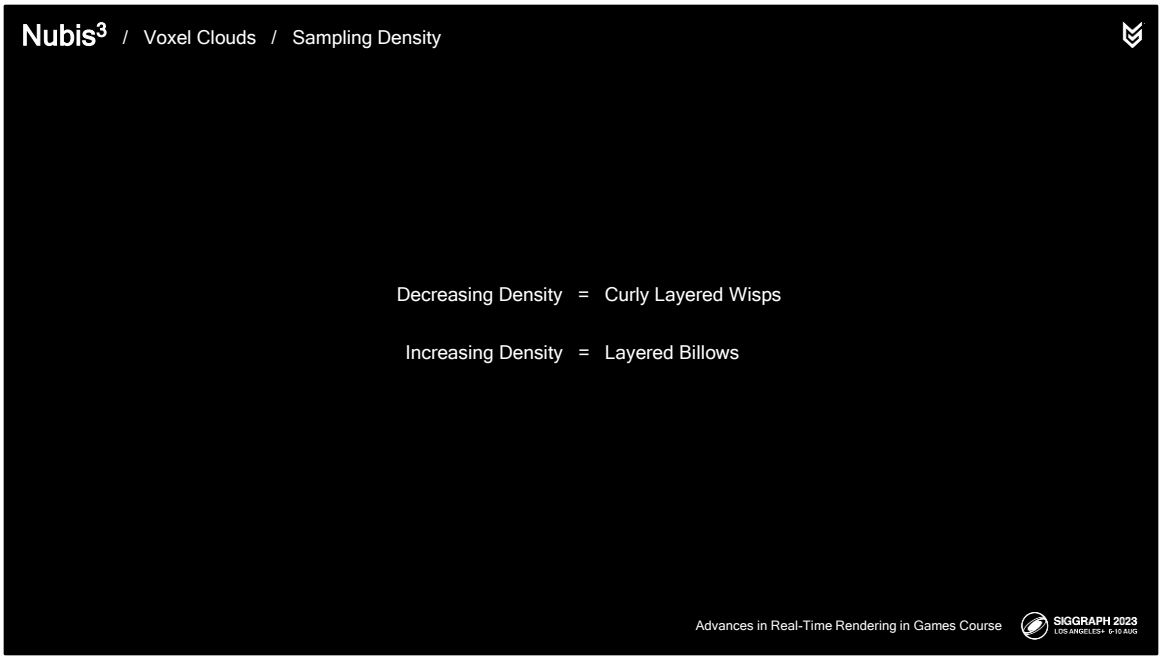


Now, invert this in your mind and you get a more wispy scalloped type of shape.

- This is what happens as the surrounding air itself billows into the water vapor as the vapor… E-vaporates.

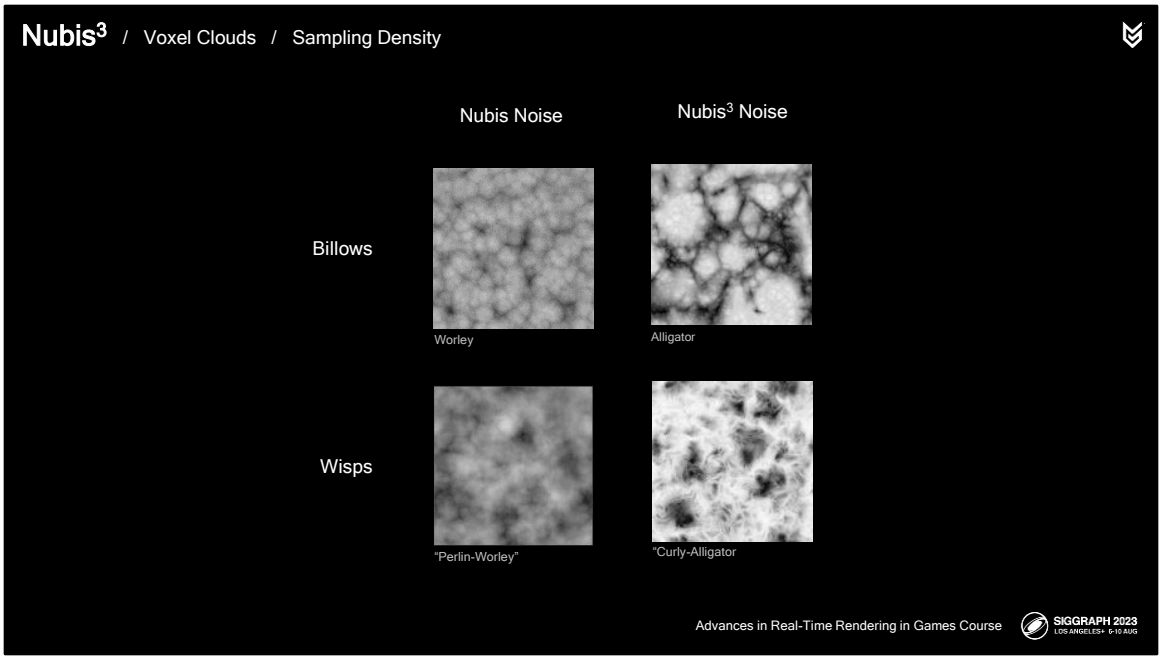


Here’s another timelapse. It looks almost web-like. In this situation, turbulence in the air plays a larger role in adding additional distortions to the shape.



To sum it up,

- layered Wisps and curly distortions form where density is decreasing
- and layered billows occur where density is increasing.

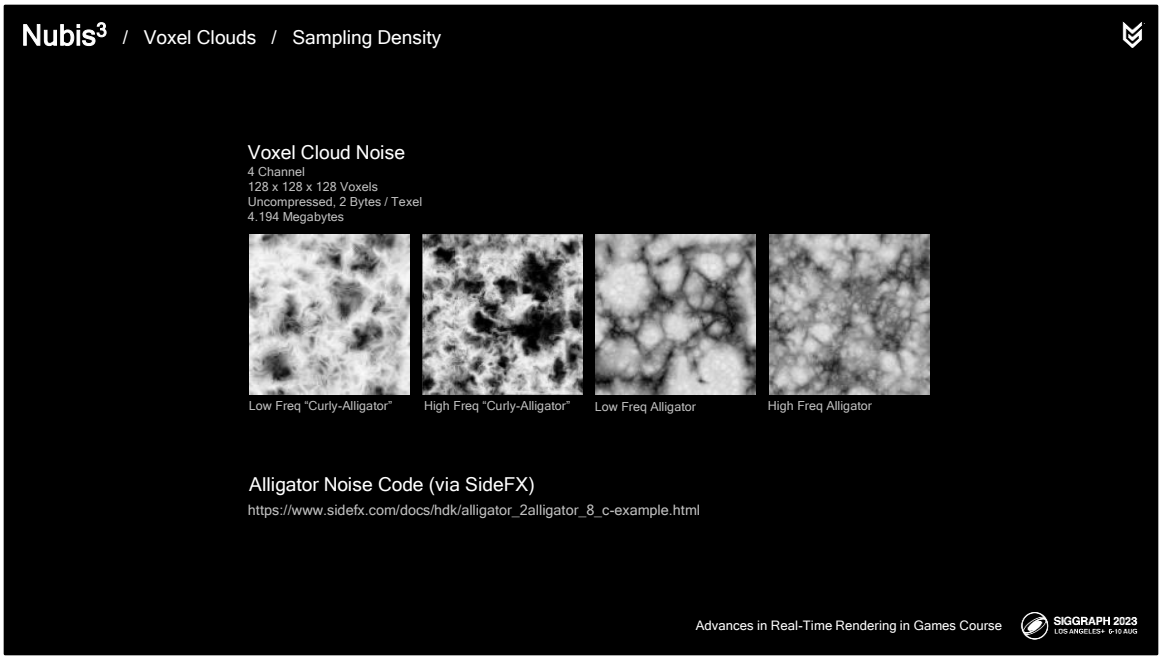


In the past,

- we used inverted Worley noise to create billowy details, but we had to do quite a lot of layering and work in the sampler to get something that looked like cloud billows as opposed to packed spheres.
- This time we decided to use Houdini's Alligator noise, which looks very much like our layered Worley noise but with more appropriate cloud-like lacunarity to these structures.
- For wispy details we previously used something we called Perlin-Worley noise. But, this was not

purely wispy enough to be used in this up-rez procedure.

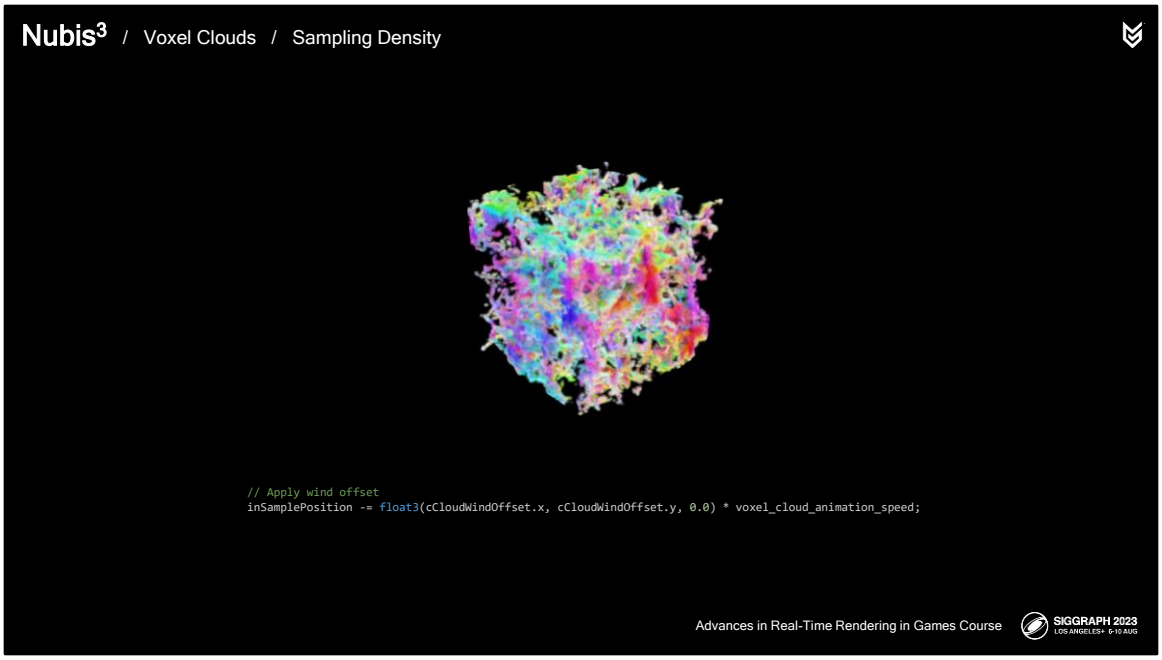
- Instead, we started with inverted alligator noise, which creates nice web like shapes, and then distorted it using curl noise.
- We call this Curly-Alligator noise.



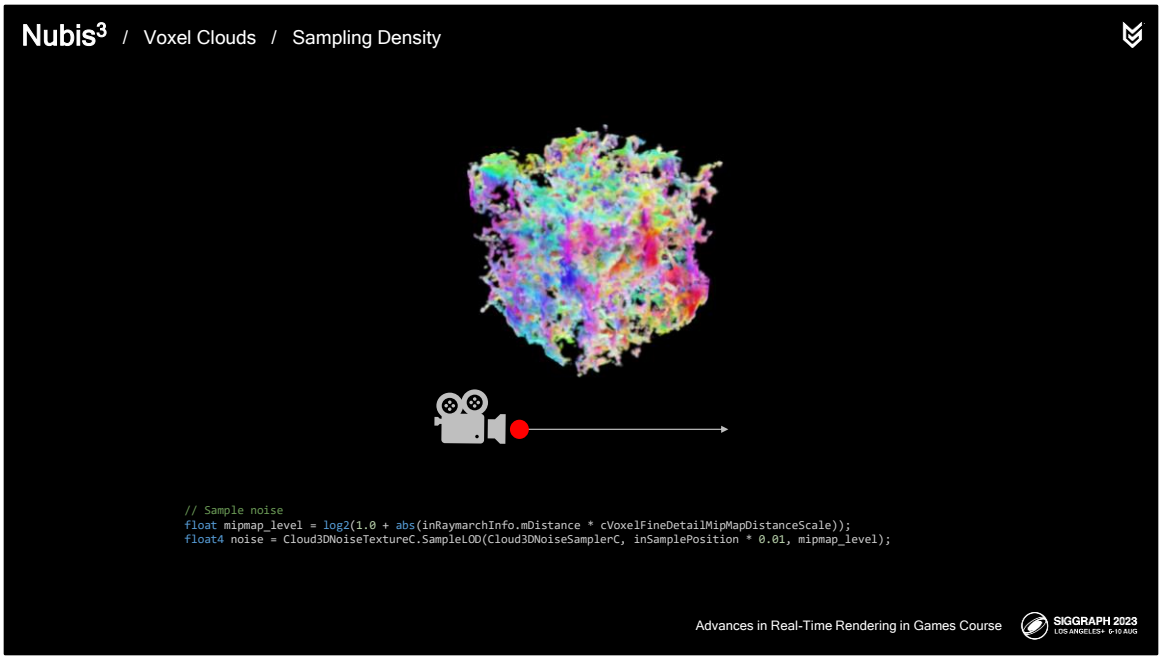
The 3d texture we generate has 4 channels of 128x128x128 voxels.

- The first two channels are low and high frequency Curly-Alligator noise
- and the last two are Low and High frequency Alligator Noise. Stay tuned until the end of the talk for something to help generate these 3D noises.
- I am also happy to share a direct link to the source code for Alligator noise, kindly provided by SideFX Software.

Now that we understand the new 3D noises that we use to up-rez the dimensional profile, we can look at the implementation of the Up-rez function itself.



The first step is to make sure that our detail noises scroll as they did in the 2.5D clouds. We do this by adding a wind offset to the noise sample position.



Next we

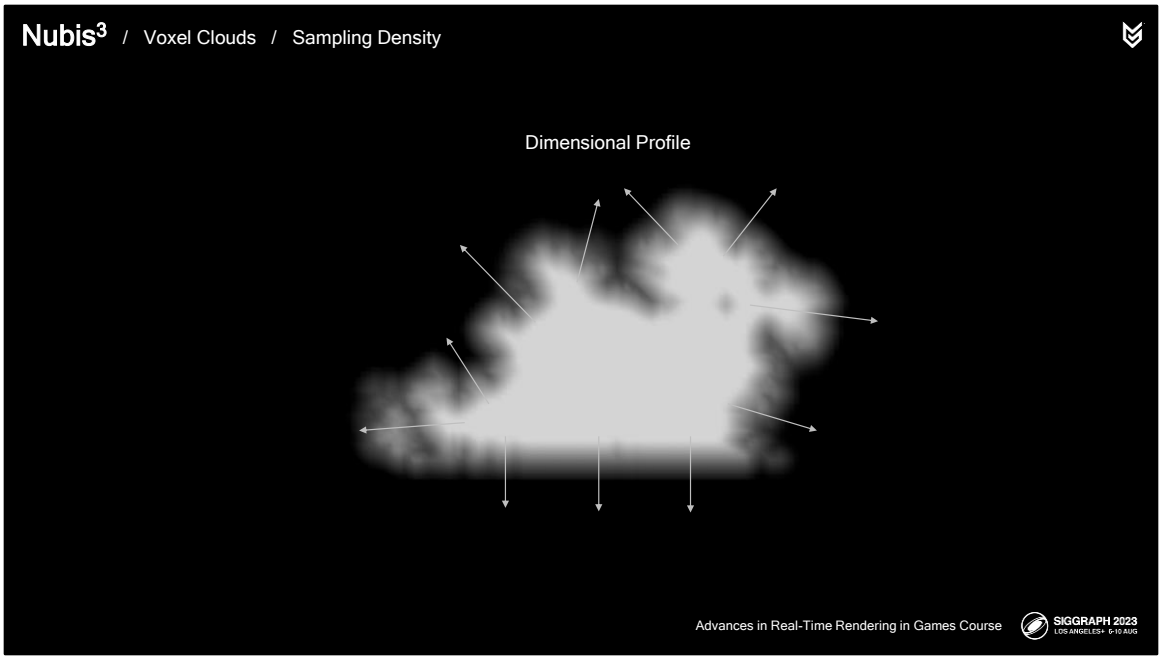
- sample the 3d noise texture while increasing the MIP level over the distance from camera. This improves our performance by about 15% over a fixed MIP level and doesn't noticeably affect the results. Remember that the sampler is called a lot, so every little bit helps.

Now let's look at how we define wispy and billowy detail noise inside of the up-rez function.

First, take a look at this photograph of wispy clouds.

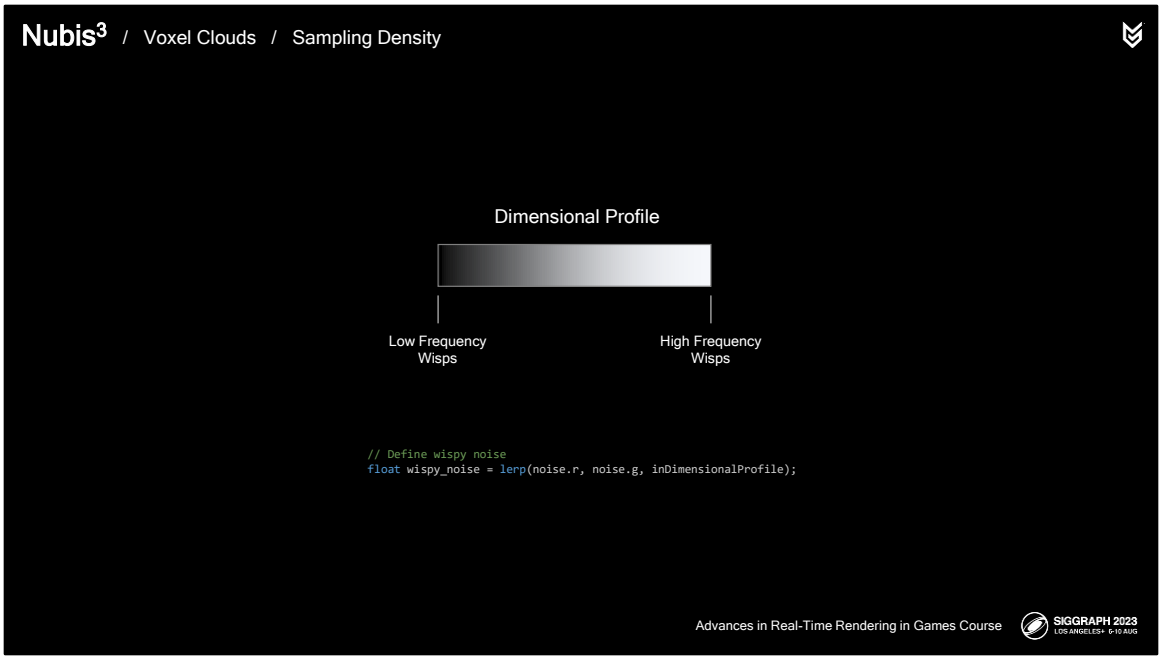
- Notice how there are very high frequency details in the less dense regions
- and lower frequency wispy structures in the denser regions. It's as if we need to provide structure to the high frequency shapes at the edges by using the low frequency shapes.

We want to mimic this relationship with our up-rez noise composites.

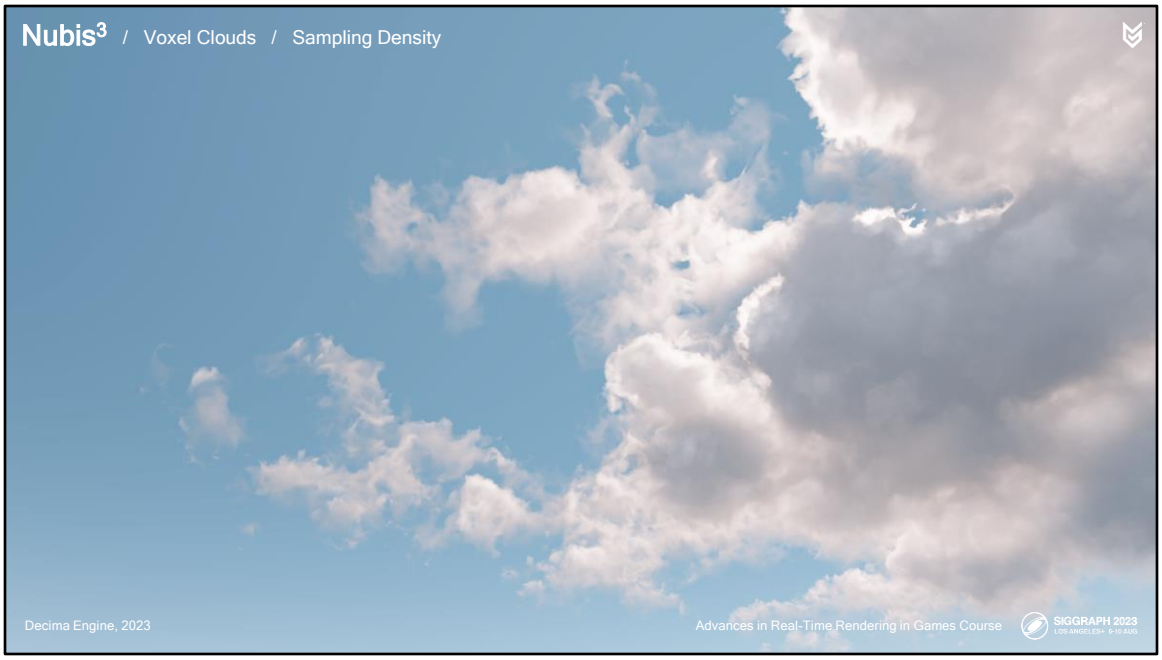


Recall our dimensional profile NVDF. As you can see,

- this already provides us with a gradient from the outside to the inside of the cloud.

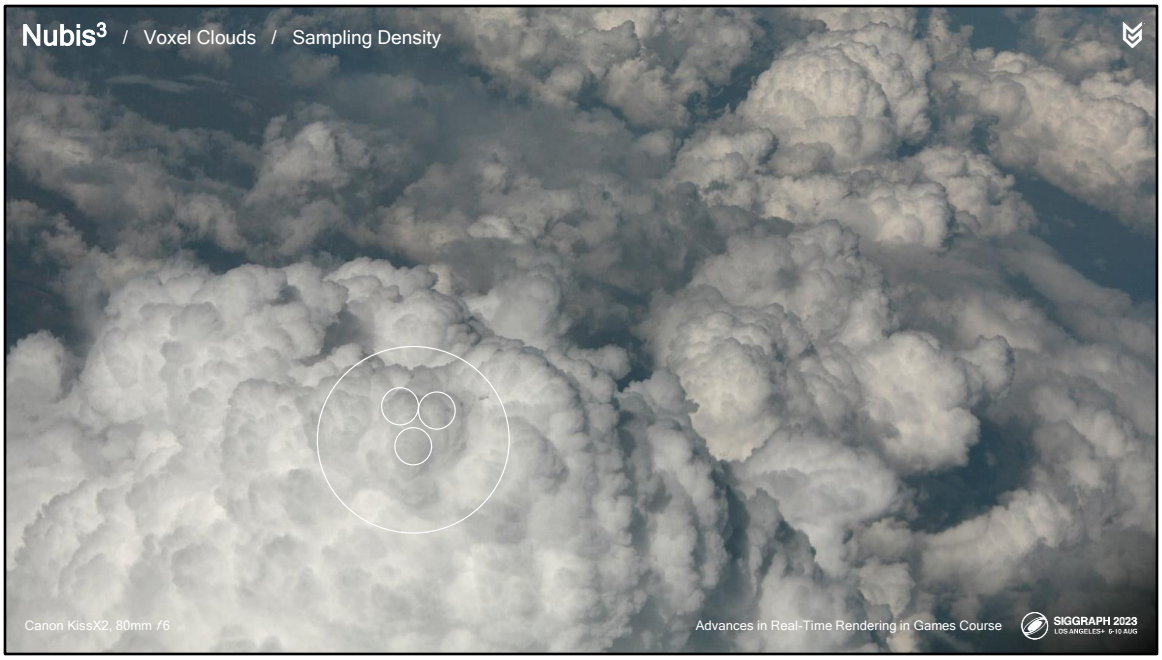


To mimic this, we can simply blend from low frequency wispy noise to high frequency wispy noise over the dimensional profile.



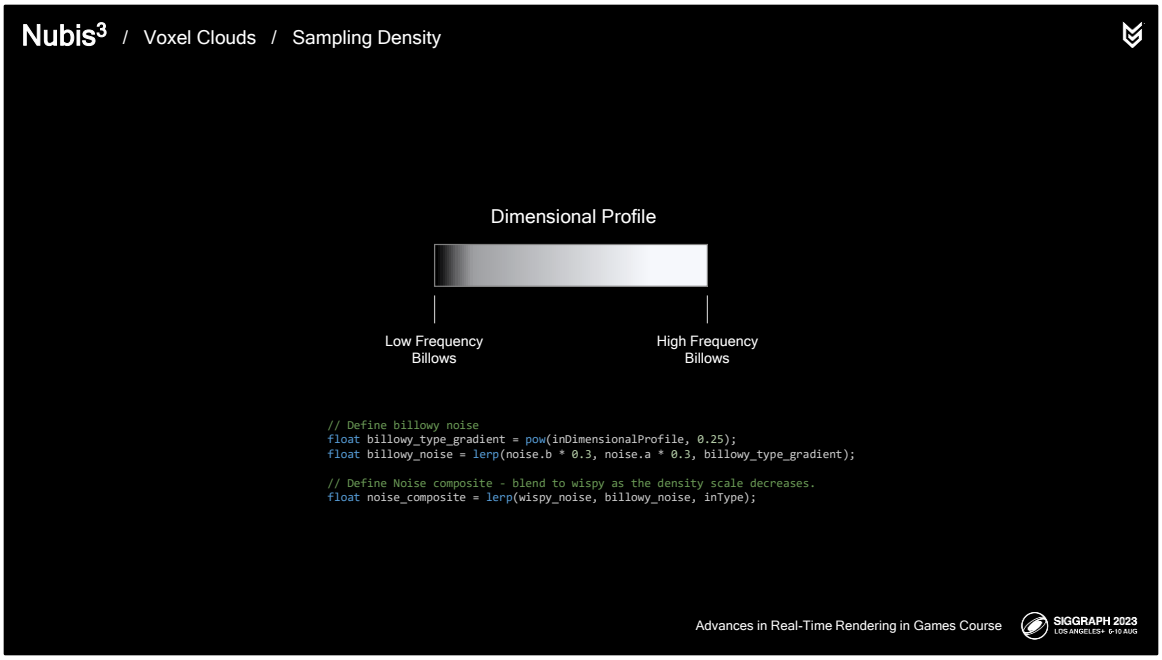
Here is what this looks like when it is applied to the dimensional profile. What you see here is Only High Frequency noise,

- And now Low Frequency noise
- And Now both with the blend over the dimensional profile.

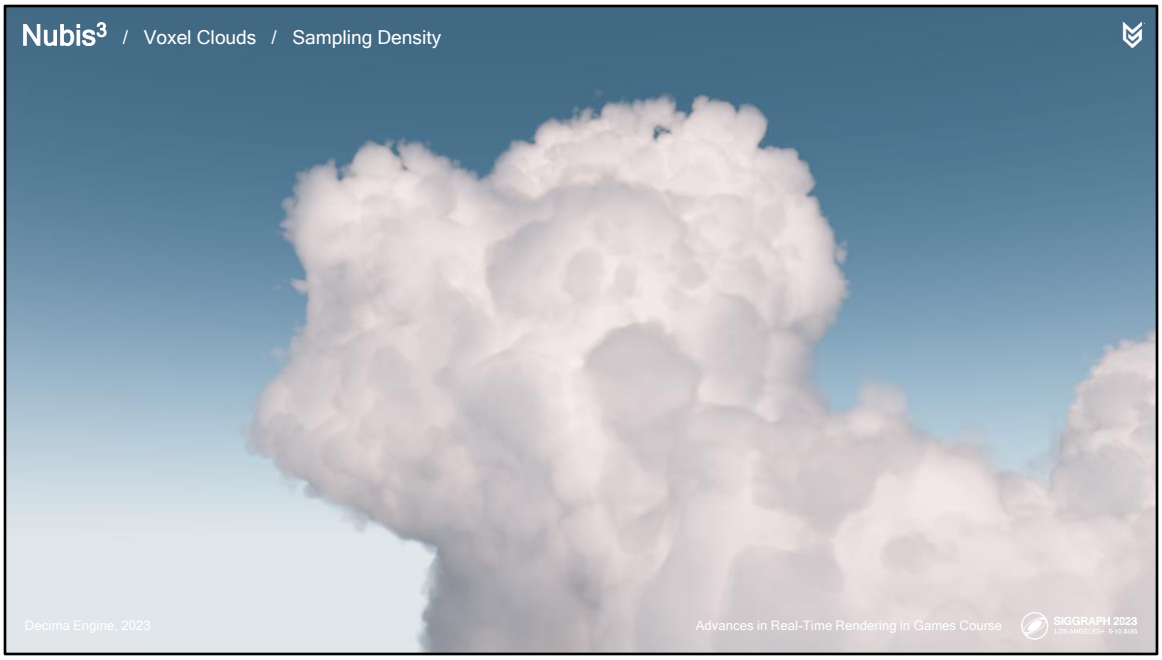


Now look at this photo of some billowy cloud details.

- Notice how we have low frequency billowy shapes near the core
- and higher frequency billowy shapes near the- for lack of a better word – surface.

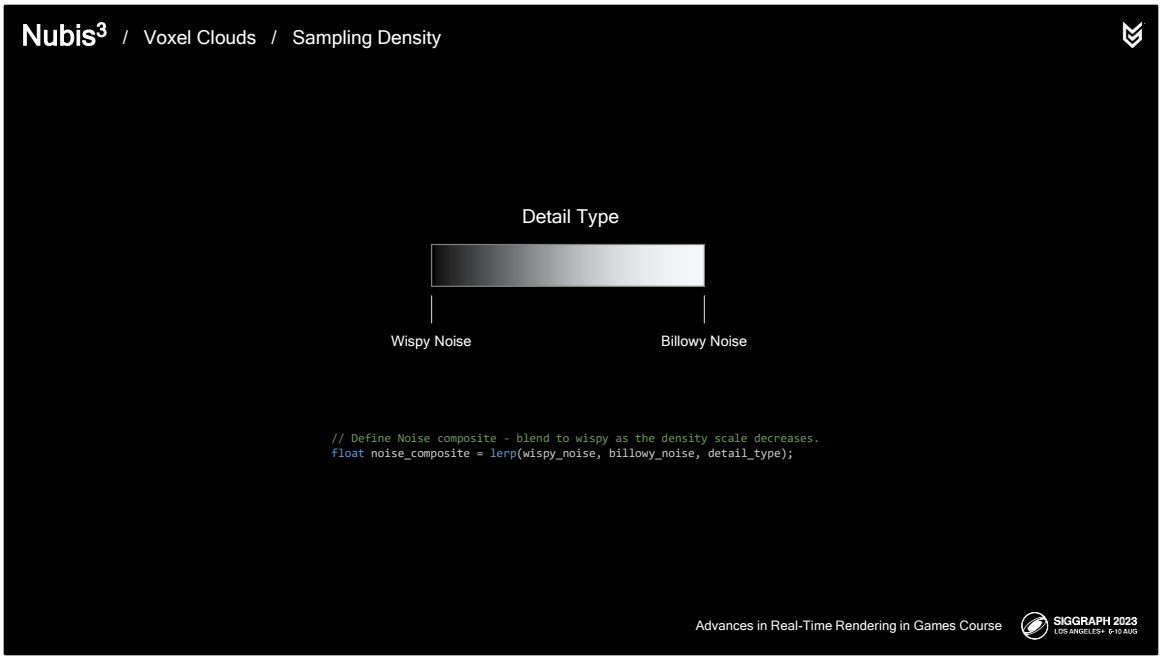


In the case of billows, we want the edges to have more rounded structure than the core – otherwise we will just get high frequency noise everywhere on the edges, so we blend from low frequency to high frequency over the dimensional profile.

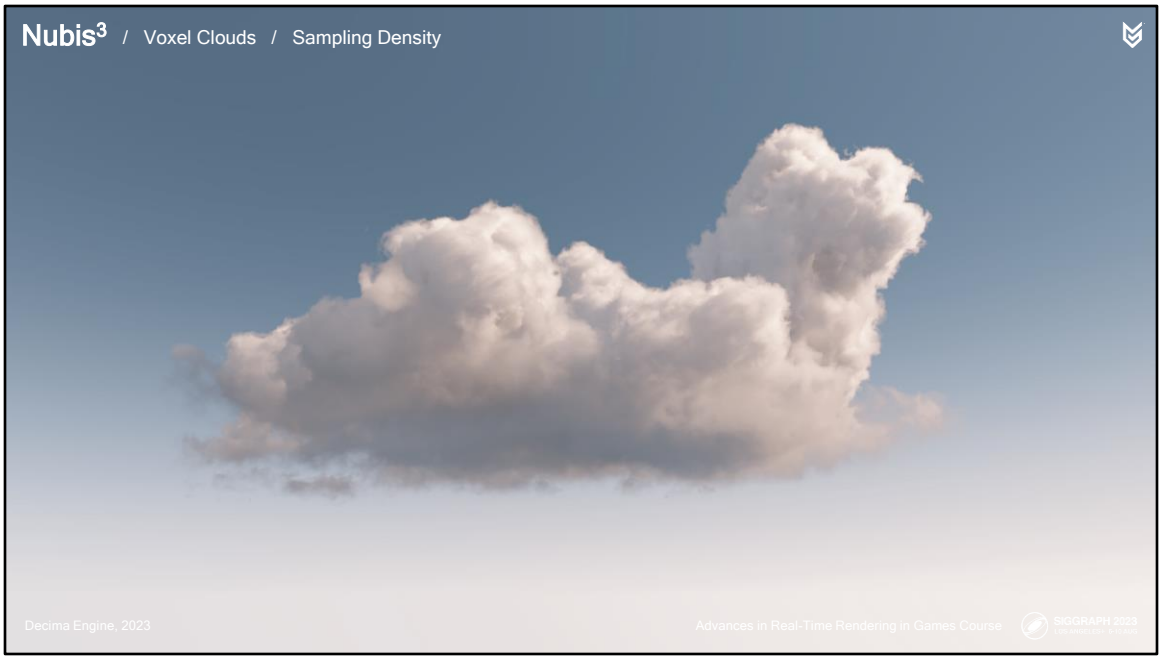


Here is an example of this noise composite starting with Only High Frequency,

- Low Frequency and
- Now both with the blend over the dimensional profile.

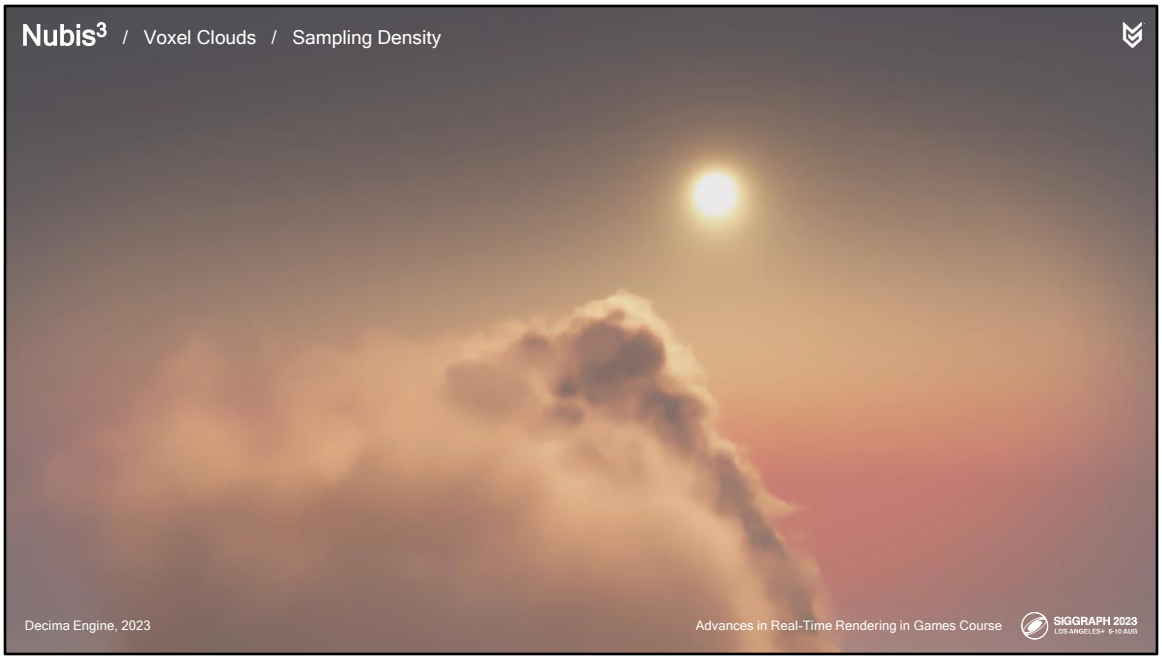


Once we have our two noise composites for the sample position, we blend from wispy noise to billowy noise over the type data as we did with the 2.5D clouds.

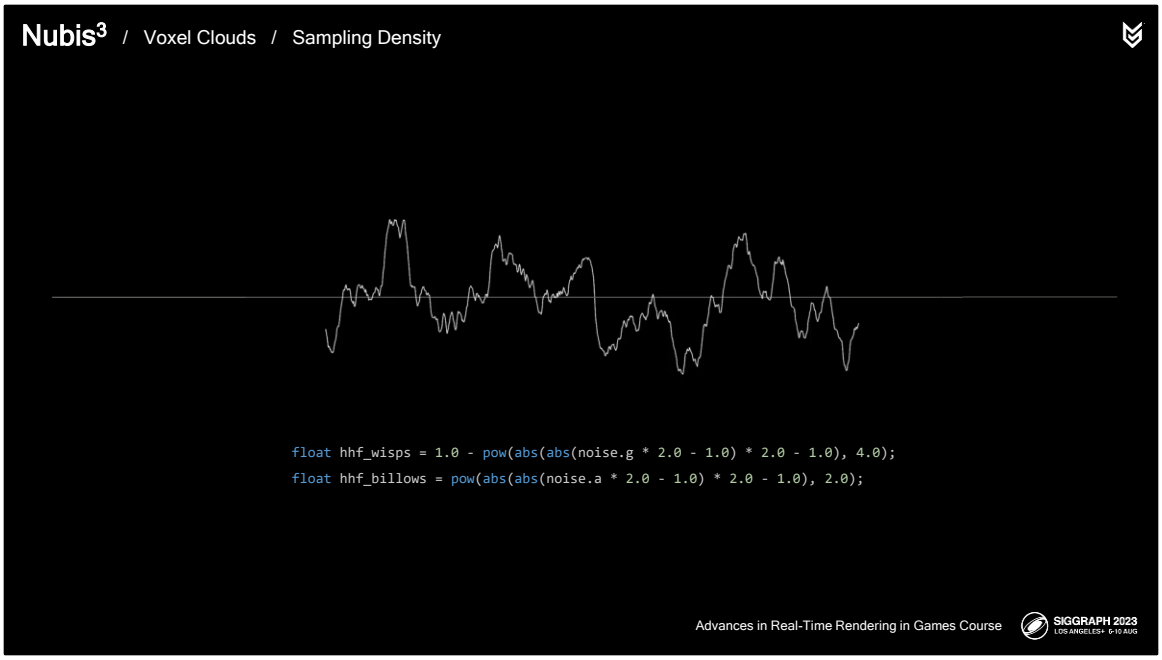


So here is a cloud with just wispy noise,

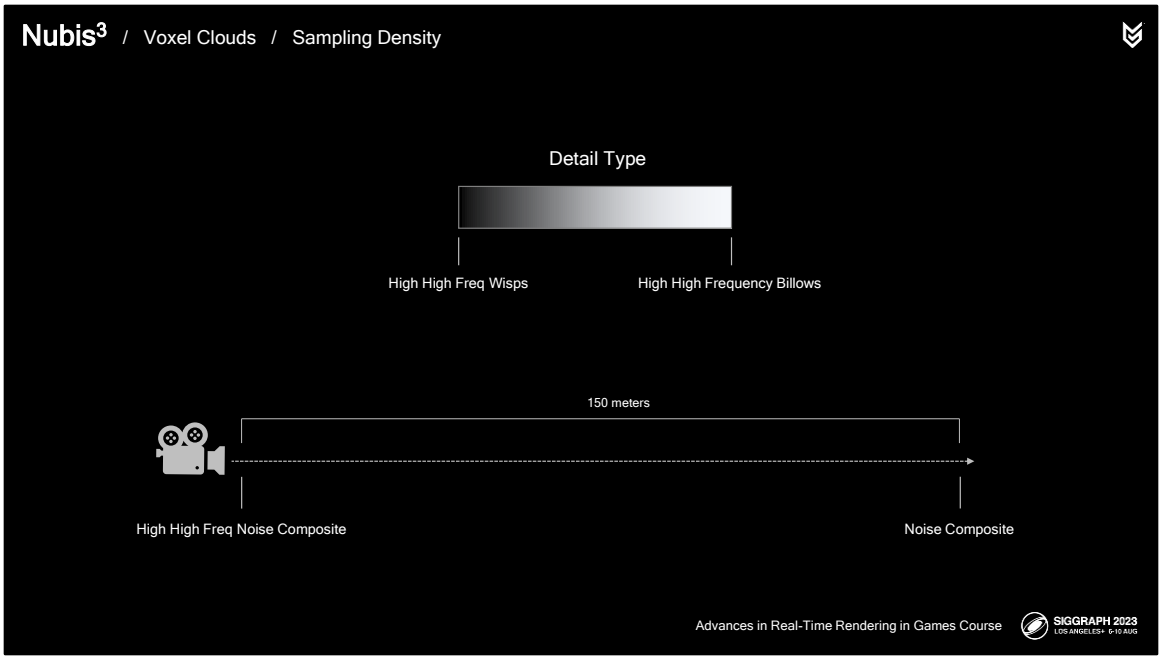
- Just billowy noise
- And now both, blended over the detail type data.



This works well from a distance, but this still didn't scale the way we wanted when flying close to clouds. There's a dramatic lack of detail up close. So, we devised a way to reuse some of our high frequency noises to create an even higher frequency noise and avoid sampling 3d noise again which can add a lot of cost in a sampler.

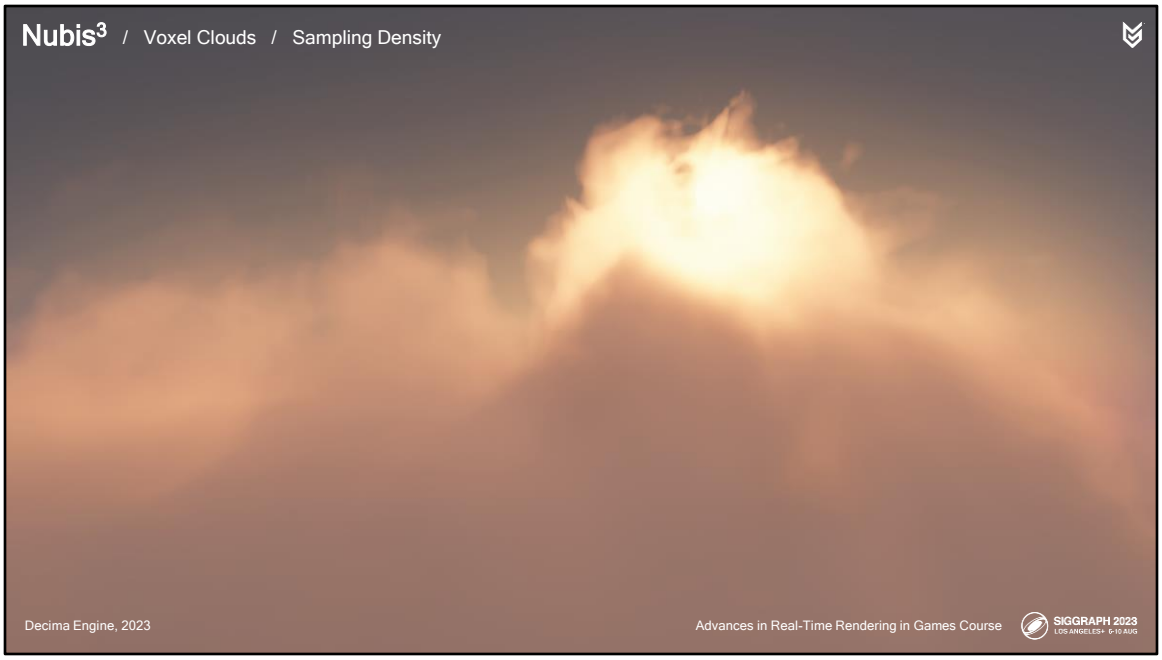


We created what I call twice folded noise by expanding a noise to the -1 to 1 range and folding it over zero using an abs function two times. As you can see it appears higher frequency. We do this for each noise type.



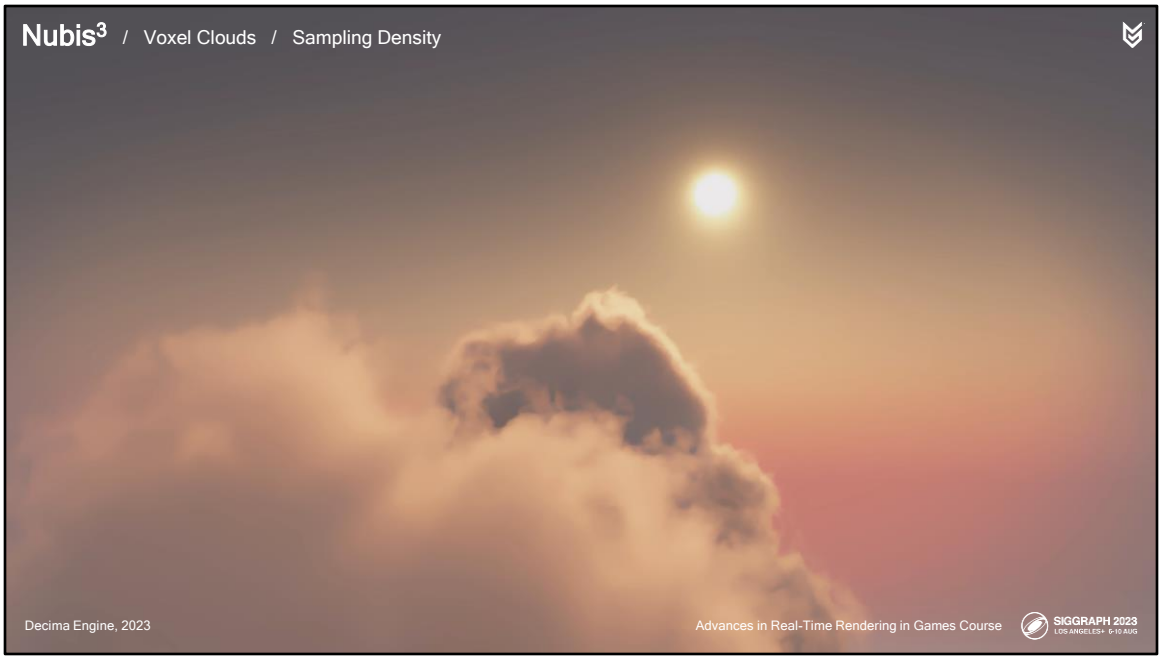
And then blend between them using the detail type data as with he lower frequency noise composites.

- Finally, we blended from the original noise composite to this higher frequency noise nearby camera.

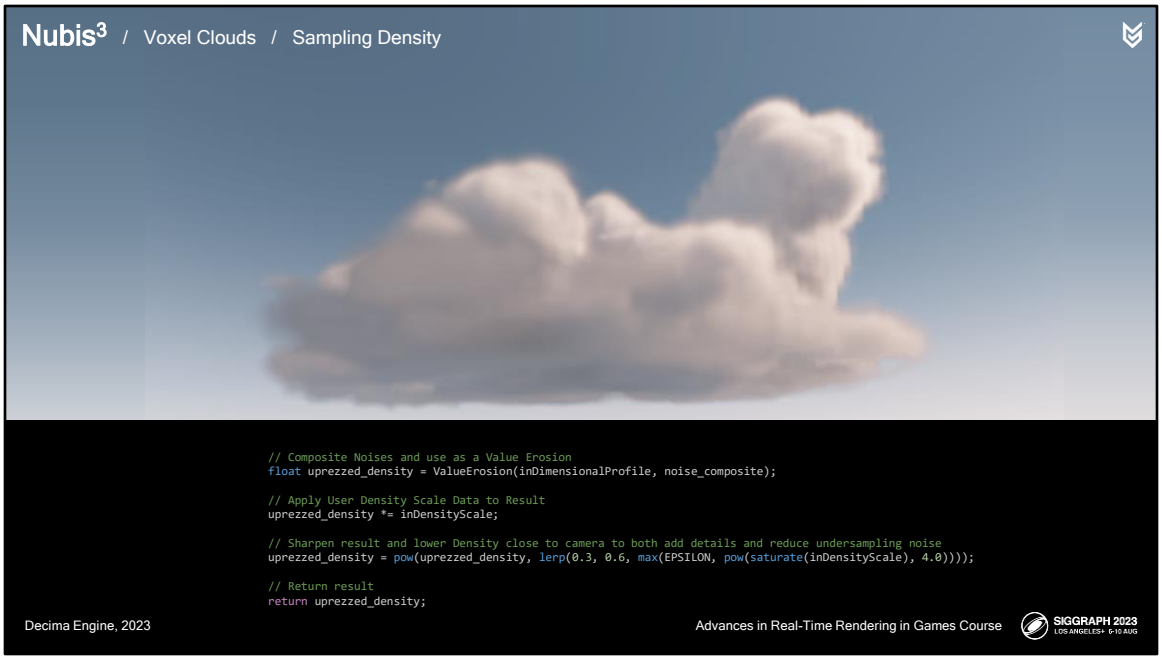


Here is what it looks like close by without this enhancement

- and then with the enhancement.



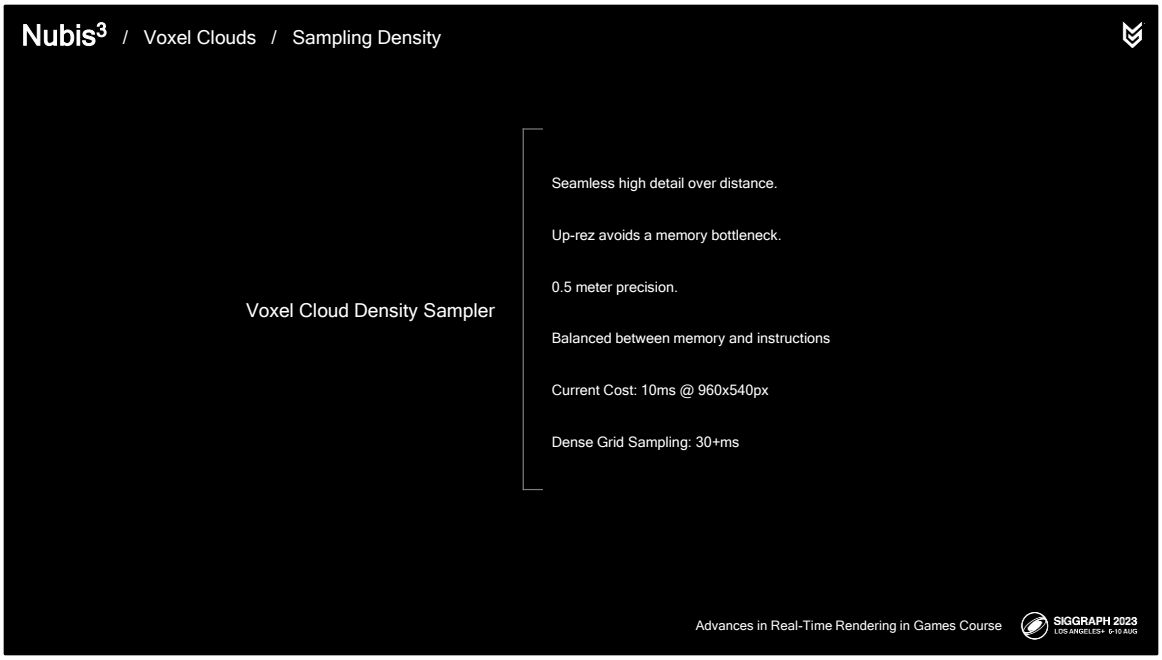
And here is what it looks like as we fly into a cloud and the detail noise takes effect. As you can see the effect is seamless.



Now that we understand how the noises work, lets apply them…

- We erode the Dimensional profile using a remap function and our noise composite as we would for 2.5D clouds.
- Then we multiply the density scale data by the up-rezed density.
- One final trick we do is to sharpen the result with a power function, doing this more in areas of low density to bring out definition in the low density areas.

- Then we return the up-rezed result.



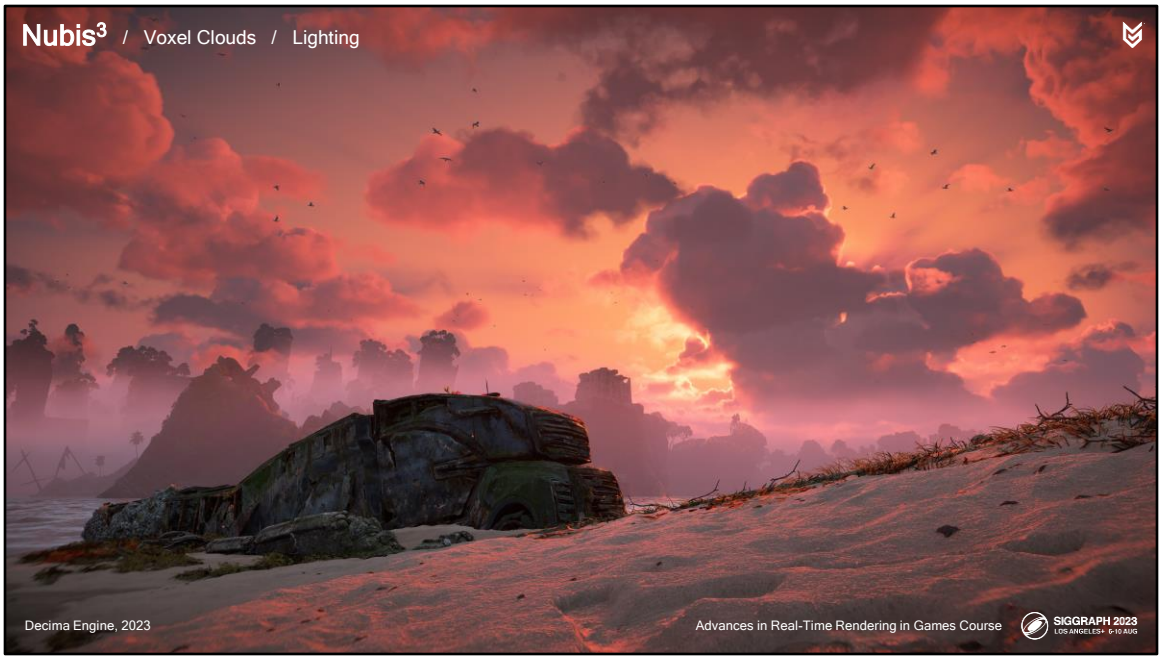
To summarize:
The result is a seamless way to render clouds from the ground and up close that simplifies the sampler function and
sidesteps memory and access time bottlenecks while still delivering a high degree of detail.
The Up-rez procedure delivers us .5meter precision from the 8 meter precision of the low-resolution Dimensional profile.
The key here was balancing memory usage with instructions to give us the best performance possible.

We are far from done with this overhaul though because in our worst case, a 960x540 pixel render costs 10ms when viewed from the ground where we share the budget with a lot of other content. To those of you joining us from feature film VFX or Animation, this might seem good, but is not AAA Game Real-time.
However, up until this point we have been struggling just to get anything with the detail we need to run in the physical memory in the console, so let’s keep going.

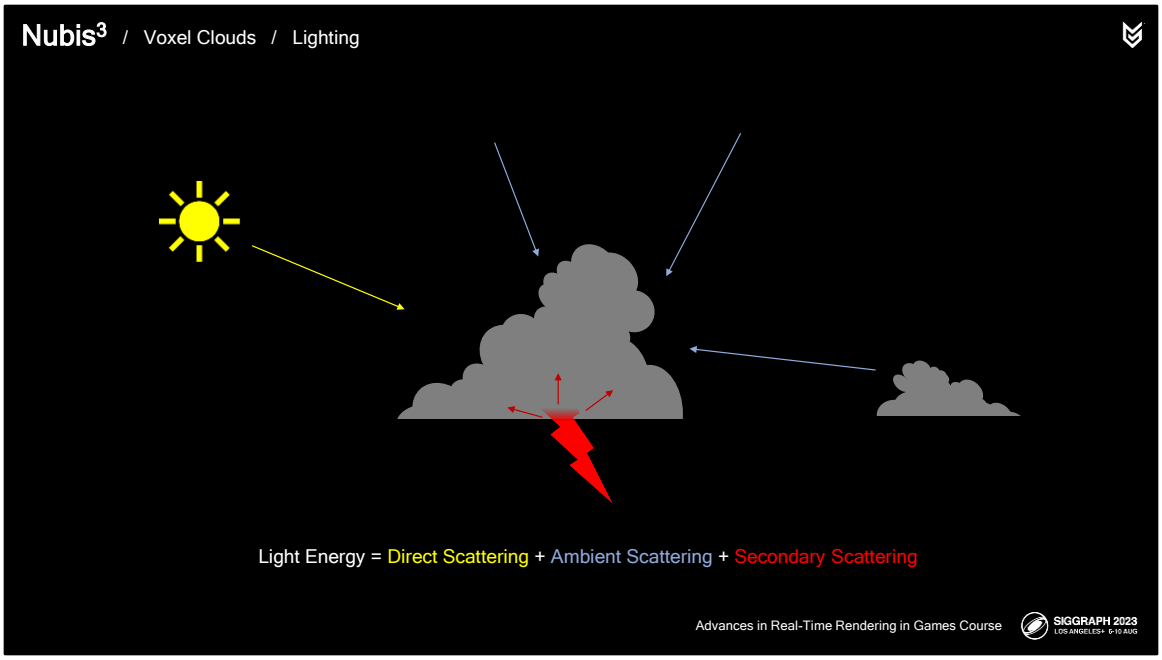
Bonus Slide

Voxel Cloud Density Sampler Up-Rez Function HLSL Code

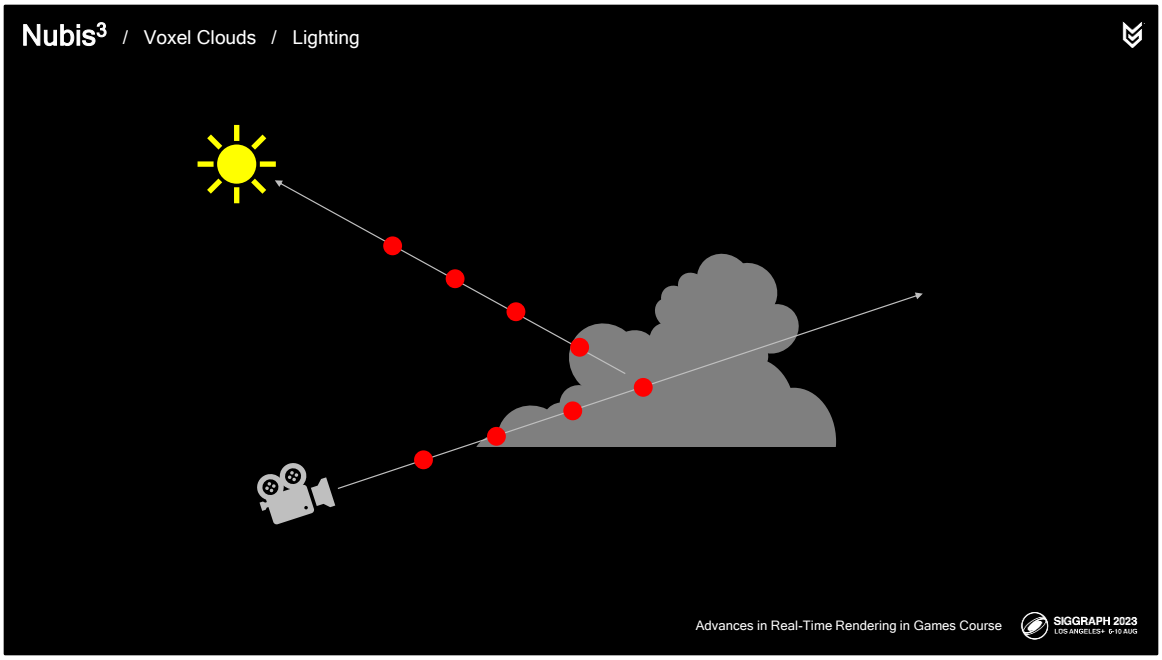
```
//
// Function to up-rez a Dimensional Profile NVDF using noise Detail Type and Density Scale NVDF's
//
float GetUprezzedVoxelCloudDensity(CloudRenderingRaymarchInfo inRaymarchInfo, float3 inSamplePosition, float inDimensionalProfile, float inType, float inDensityScale, float inMipLevel, bool inHFDetails)
{
    // Apply wind offset
    inSamplePosition -= float3(cCloudWindOffset.x, cCloudWindOffset.y, 0.0) * voxel_cloud_animation_speed;

    // Sample noise
    float mipmap_level = GetVoxelCloudMipLevel(inRaymarchInfo, inMipLevel);
    float4 noise = Cloud3DNoiseTextureC.SampleLOD(Cloud3DNoiseSamplerC, inSamplePosition * 0.01, mipmap_level);

    // Define wispy noise
    float wispy_noise = lerp(noise.r, noise.g, inDimensionalProfile);

    // Define billowy noise
    float billowy_type_gradient = pow(inDimensionalProfile, 0.25);
    float billowy_noise = lerp(noise.b * 0.3, noise.a * 0.3, billowy_type_gradient);

    // Define Noise composite - blend to wispy as the density scale decreases.
    float noise_composite = lerp(wispy_noise, billowy_noise, inType);

    // Get the hf noise which is to be applied nearby - First, get the distance from the sample to camera and only do the work within a distance of 150 meters.
    if (inHFDetails)
    {
        // Get the hf noise by folding the highest frequency billowy noise.
        float hhf_noise = saturate(lerp(1.0 - pow(abs(abs(noise.g * 2.0 - 1.0) * 2.0 - 1.0), 4.0), pow(abs(abs(noise.a * 2.0 - 1.0) * 2.0 - 1.0), 2.0), inType));

        // Apply the HF noise near camera.
        float hhf_noise_distance_range_blender = ValueRemap(inRaymarchInfo.mDistance, 50.0, 150.0, 0.9, 1.0);
        noise_composite = lerp(hhf_noise, noise_composite, hhf_noise_distance_range_blender);
    }

    // Composote Noises and use as a Value Erosion
    float uprezzed_density = ValueErosion(inDimensionalProfile, noise_composite);

    // Modify User density scale
    float powered_density_scale = pow(saturate(inDensityScale), 4.0);

    // Apply User Density Scale Data to Result
    uprezzed_density *= powered_density_scale;

    // Sharpen result and lower Density close to camera to both add details and reduce undersampling noise
    uprezzed_density = pow(uprezzed_density, lerp(0.3, 0.6, max(EPSILON, powered_density_scale)));
    if (inHFDetails)
    {
        float distance_range_blender = GetFractionFromValue(inRaymarchInfo.mDistance, 50.0, 150.0);
        uprezzed_density = pow(uprezzed_density, lerp(0.5, 1.0, distance_range_blender)) * lerp(0.666, 1.0, distance_range_blender);
    }

    // Return result with softened edges
    return uprezzed_density;
}
```



Lighting is, ironically, one of the heaviest parts of a volumetric ray-march. Our task was kind of impossible on its face: find massive performance savings while improving the quality of the results to support flight through clouds.

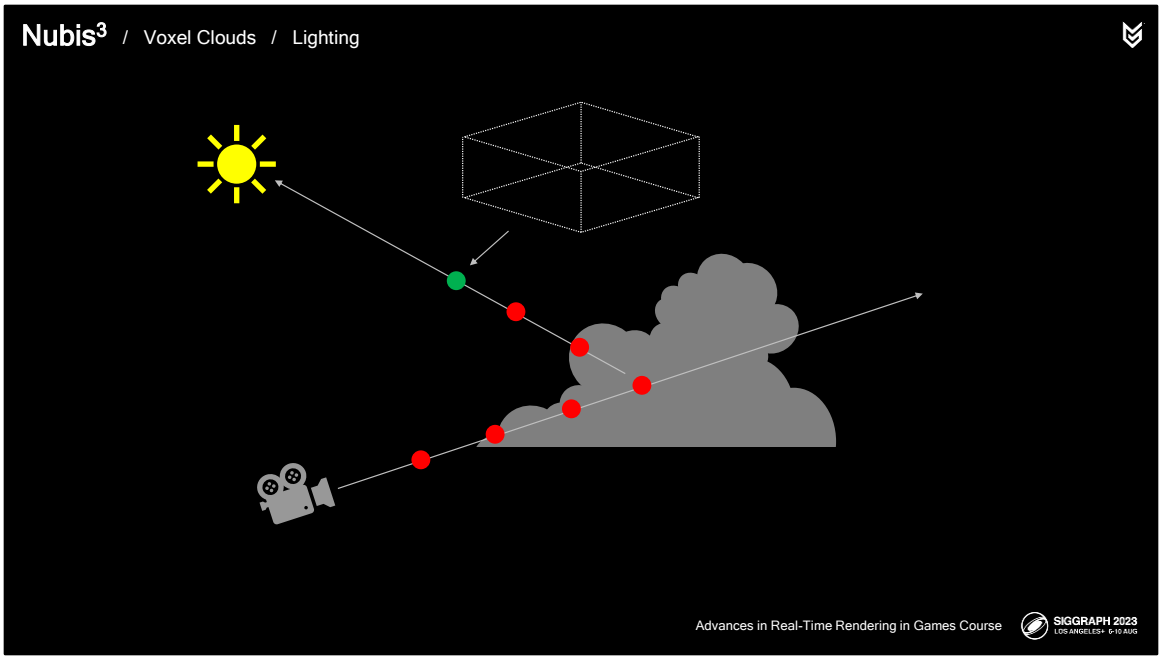


Recall that We calculate light energy components for Direct Scattering, Ambient Scattering and Secondary Sources like lightning, and that the Direct scattering calculation is quite expensive because of the secondary ray-march toward the light that I keep mentioning with contempt. We have been trying all kinds of things over the years to accelerate or replace that secondary ray-march.



It wasn't until we started thinking in voxels that one of the loftier ideas we had several years ago appeared more sane.

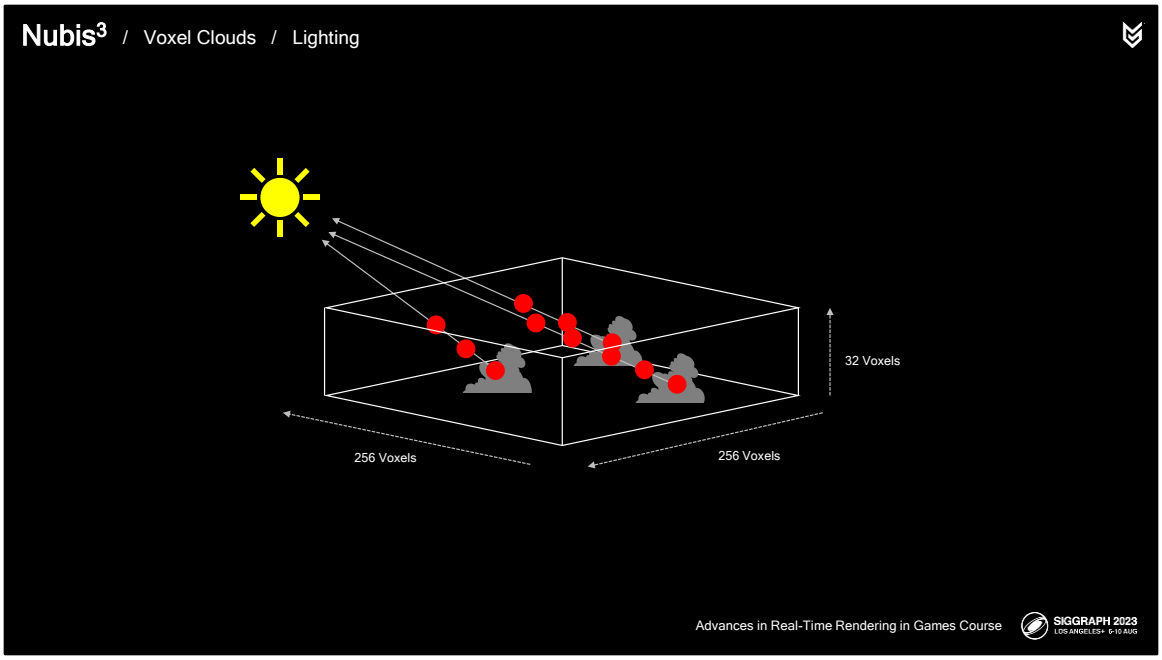
- Principal Tech Programmer Nathan Vos has wanted to decouple the light ray march from the view ray for a long time, but until we were working with voxels, the complexity of doing this just didn't make it worth it.



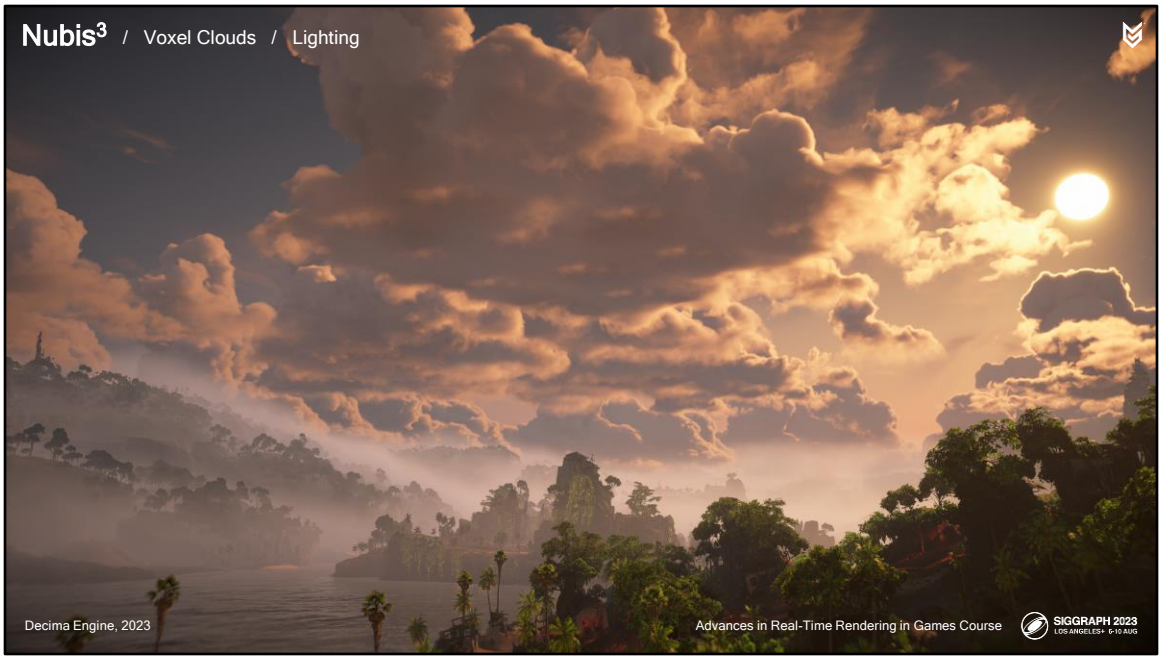
In our new approach,

- The first two samples are taken as we did before
- but the remaining samples are pre-computed in a separate pass and stored in a 256x256x32 voxel grid that we sample after the first two samples.

This preserves detail near the surfaces of clouds and offers a more diffuse result after that. To be clear, this pre-computation happens outside of the ray-march.



Here is how it works: If a voxel has cloud in it, then a ray is marched between that voxel and the boundaries of the voxel grid and density is accumulated at each step.
We found that this reduced the render times by around 40%. The operation itself costs between 0.1 and 0.2ms depending on the time of day and the length of the rays toward the sun. The cost itself is amortized over 8 frames.



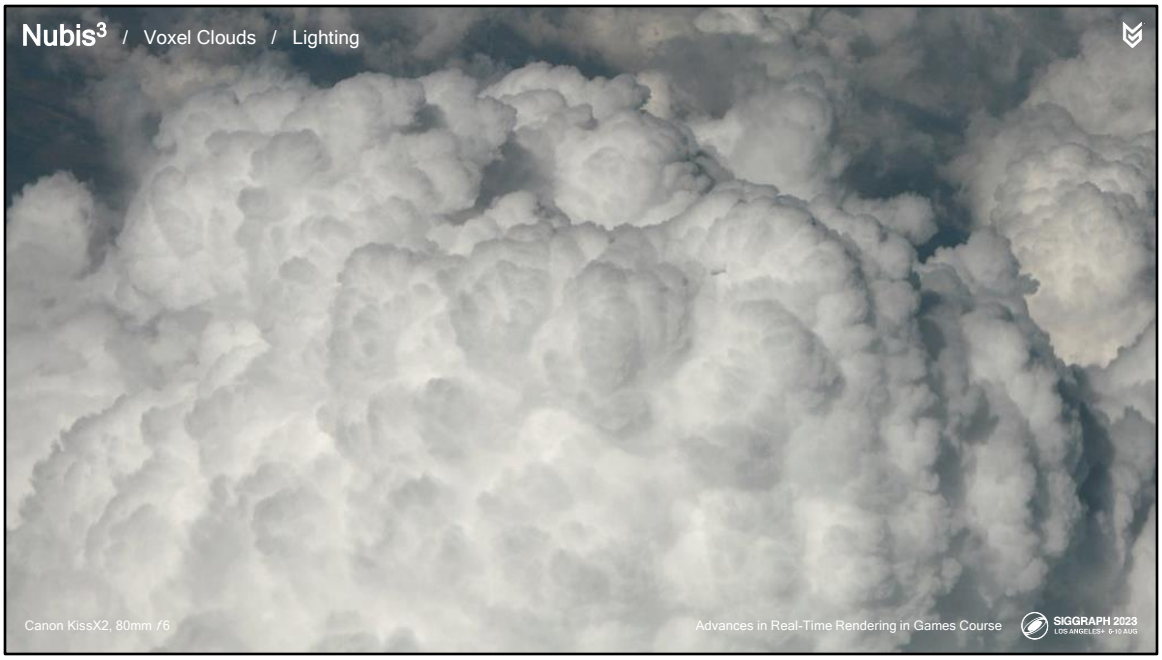
In addition to the savings, We could finally render long distance inter-cloud shadows.
Here is what it would look like with a 256 meter, 10 sample light ray march as we did before.

- And here it is with voxel-based lighting.

It was a rare occasion where an optimization produced a much better result.

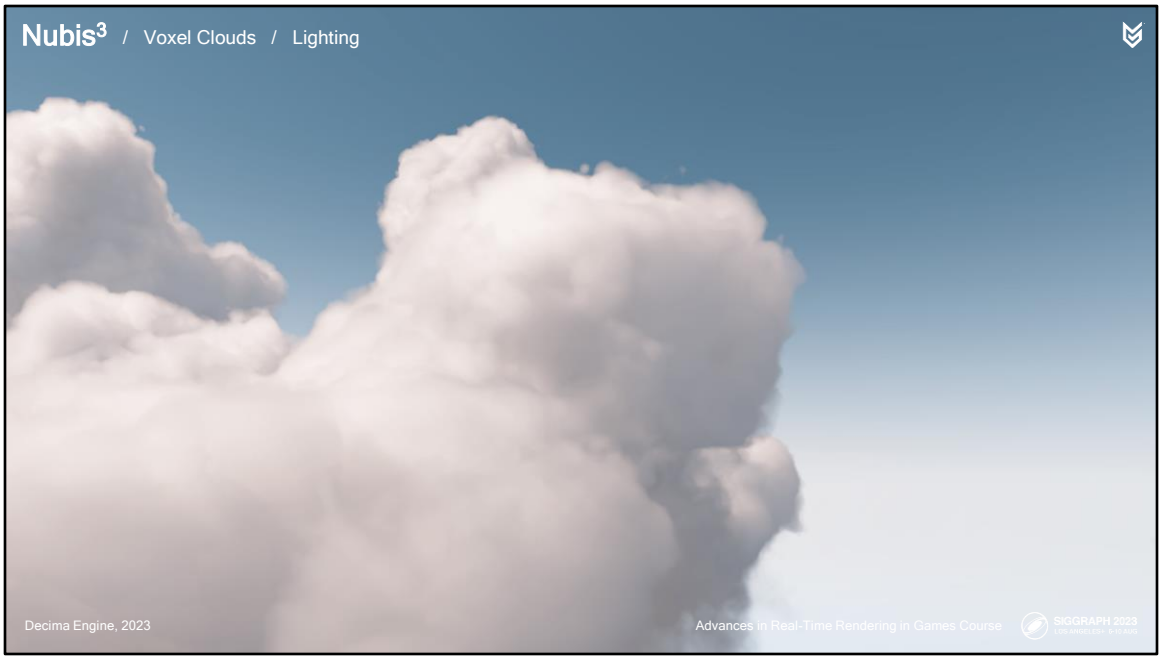
The switch to voxels allowed us a little bit of freedom to solidify how we model multiple scattering effects that are responsible for inner glow and dark edges on clouds. Let’s look a bit deeper.

Take a look at these very fine crystal growths at the Museum of Natural History in Houston, TX.
It's interesting because you can peer deeply into the core through the little gaps between each crystal. If you blur your eyes, you can almost see what looks like a lump of lighter material under the surface.



Its hard not to notice the similarities. Near the surface boundary of a cloud there is less material to make the incoming light scatter in more random directions so the majority of the rays continue on course and don't reach our eyes. You can imagine a probability field inside the cloud where the chances of in-scattering toward the viewer increases the deeper you get as the scattering direction becomes more isotropic or omnidirectional. This light volume shines through to the surface like a flashlight behind a thick sheet of cotton.

Or a glowing volume of light inside these crystal formations. When you look at this effect geometrically as the result scattering potential it is much easier to model for our real-time purposes.



Fortunately, we already have this volume. Let's use this cloud as an example.

- Here is its dimensional profile.

This serves as the basis of our probability field.

Let’s look at this cloud again, but now let’s align it so that the cloud is exactly between us and the sun.

- We are going to look at a slice of this cloud that cuts through the cloud at this angle.

Here is the slice from the same side view that we started with.

- And let’s turn off all of the light energy.

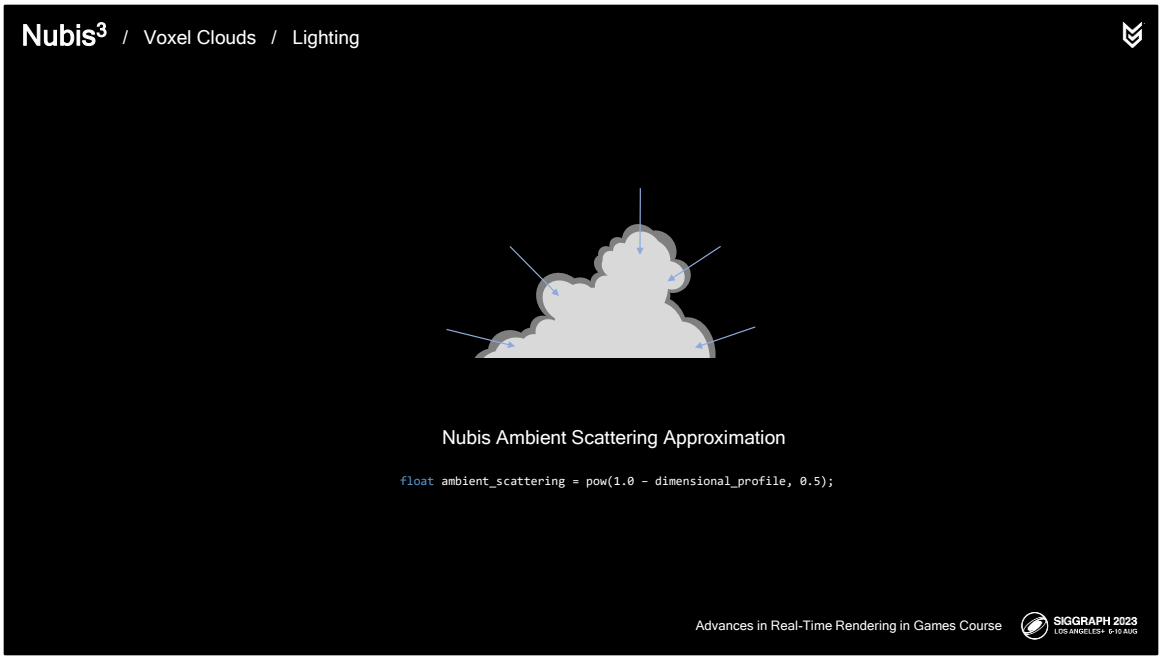
Here is what if looks like if we use the dimensional profile as scattering.
As light scatters in this way it is eventually absorbed , so we apply a

- beer-lambert attenuation curve to the field.

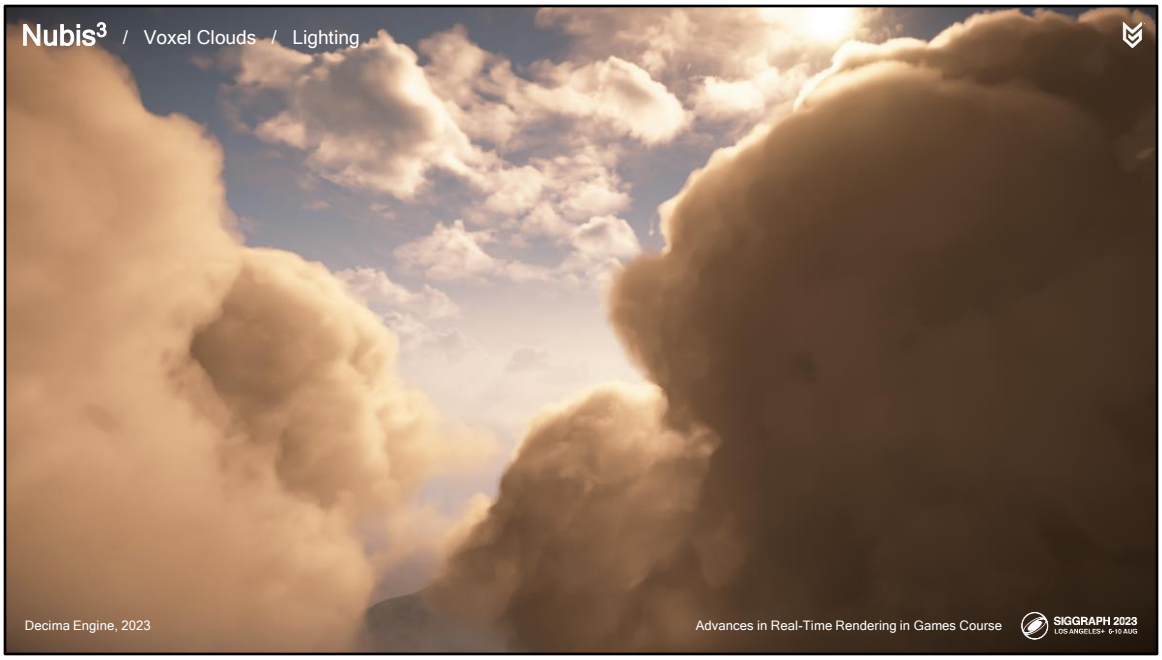
This curve is relaxed for vectors near the sun and near the inside of the cloud to produce the inner glow effect.
Let’s look at clouds we shipped with the game.

Here is a view with the sun in front of us with no multiple scatter approximation.
And then with the approximation.

And here is a view with the sun behind us without the approximation.
And then with the approximation.



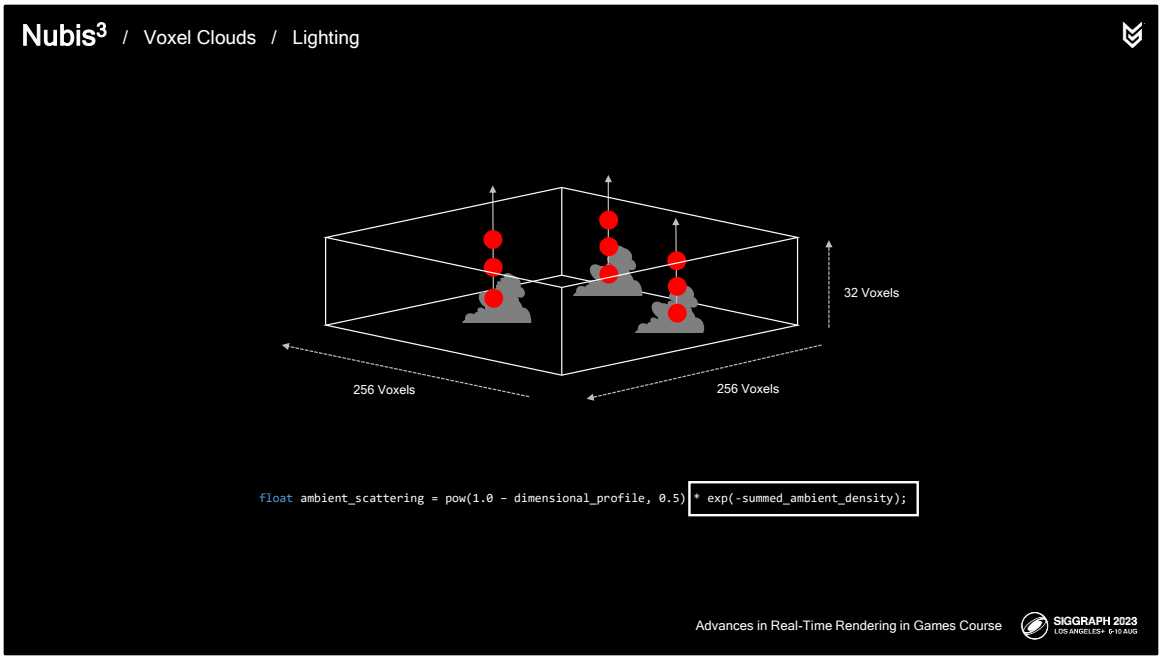
Just like with the 2.5D clouds, We also use the dimensional profile as a probability field to calculate ambient light potential.



Here it is without ambient light contribution,

- and then with ambient light contribution.

Pre-calculating our lighting and sampling it in the light ray march worked so well that we thought the technique might be able to help us finally improve our ambient light contribution by adding directionality.

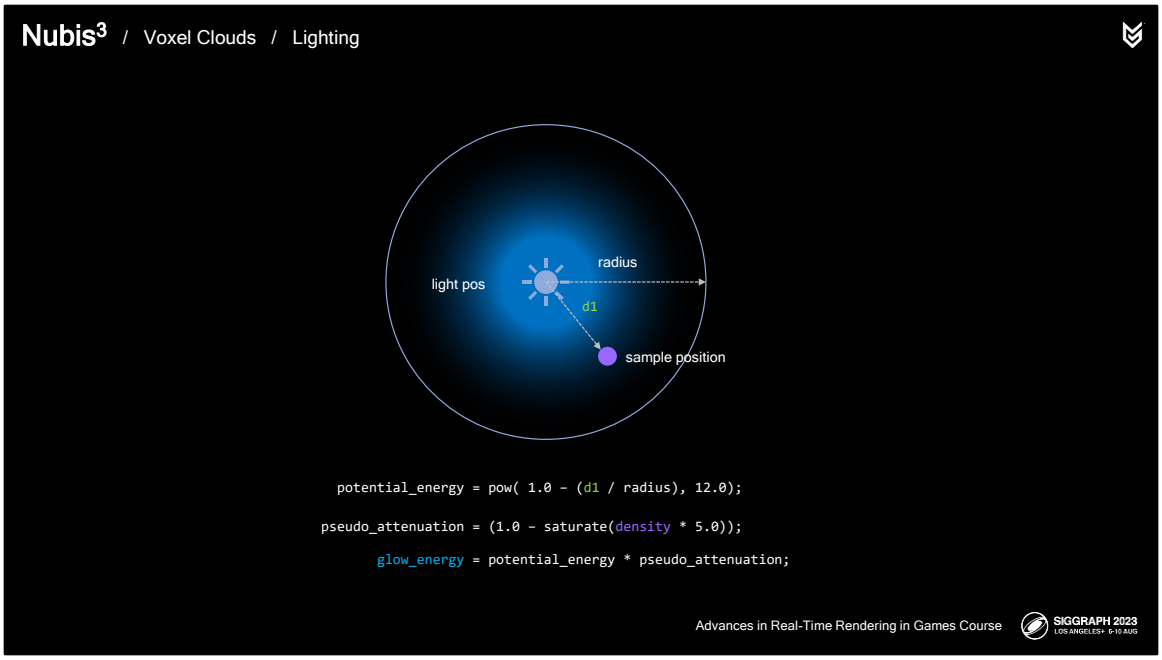


When we pre-calculate summed density along the ray toward the sky, we get a result that can be used to scale this ambient scattering so that it is less intense in areas where the sky is occluded by clouds above the sample position.

- We simply scale our original method by an attenuation signal computed from the summed density toward the sky.

Here is what it looks like without directionality in the ambient scattering component

- And then with some directionality provided by voxel-based lighting.

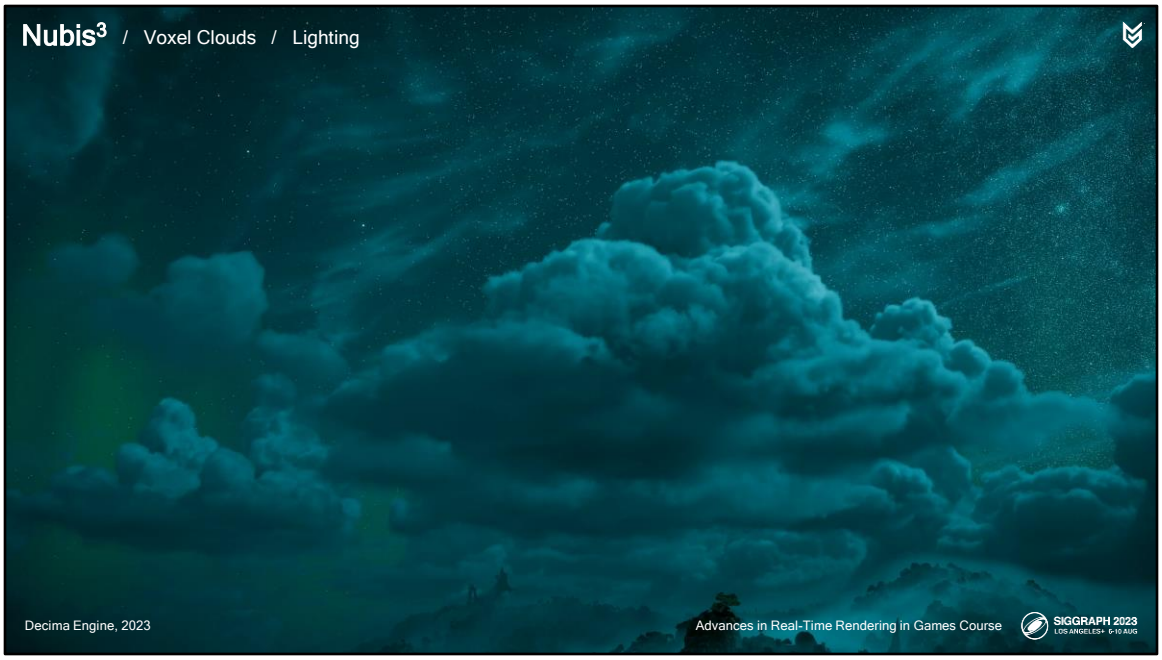


Secondary sources like lighting were once again approximated as a probability field.

- First, we defined a potential energy for the flash using spherical falloff and
- then multiplied this by something that would fake the effect of non-homogeneous density.

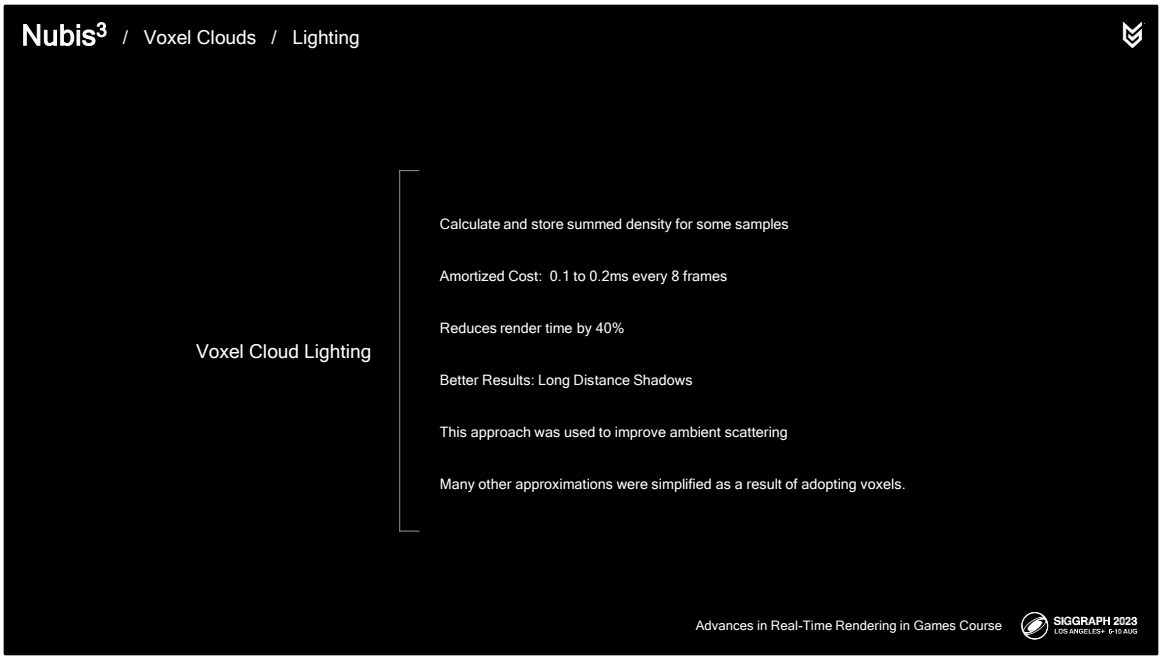
Here is what this looks like in

- game. I guess this is a preview of what's coming.



Because this is such a big overhaul, It is important to make sure that the time-of-day cycles still work as intended. So here is the obligatory TOD cycle.

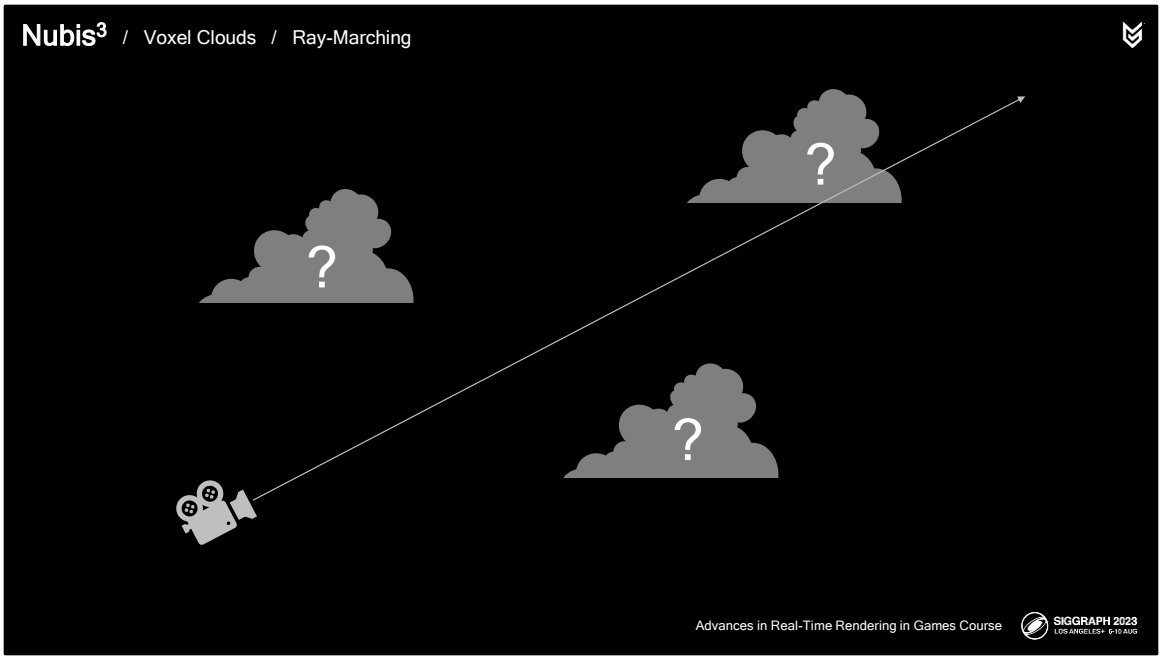
Those long distance shadows make quite a difference.



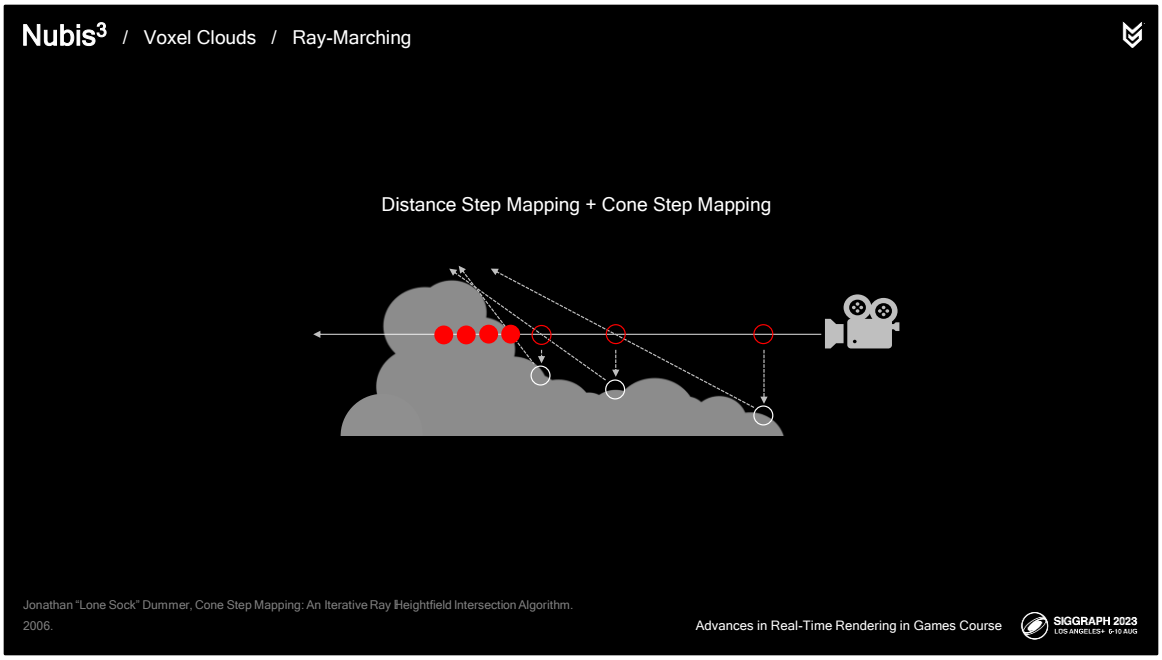
In short: We decoupled the lighting ray march and precomputed and stored summed density in a voxel grid.
we reduced the cost of a frame by 40% while actually improving the quality of the results by adding long distance shadows.
We also used this approach to improve ambient lighting
Additionally, some of our lighting approximations were simpler to calculate by just looking up the dimensional profile
Sometimes adopting a new efficient approach

opens other doors too.

Marching right along, we must also look to the…. ray-march for performance savings if we are going to support flight through clouds.

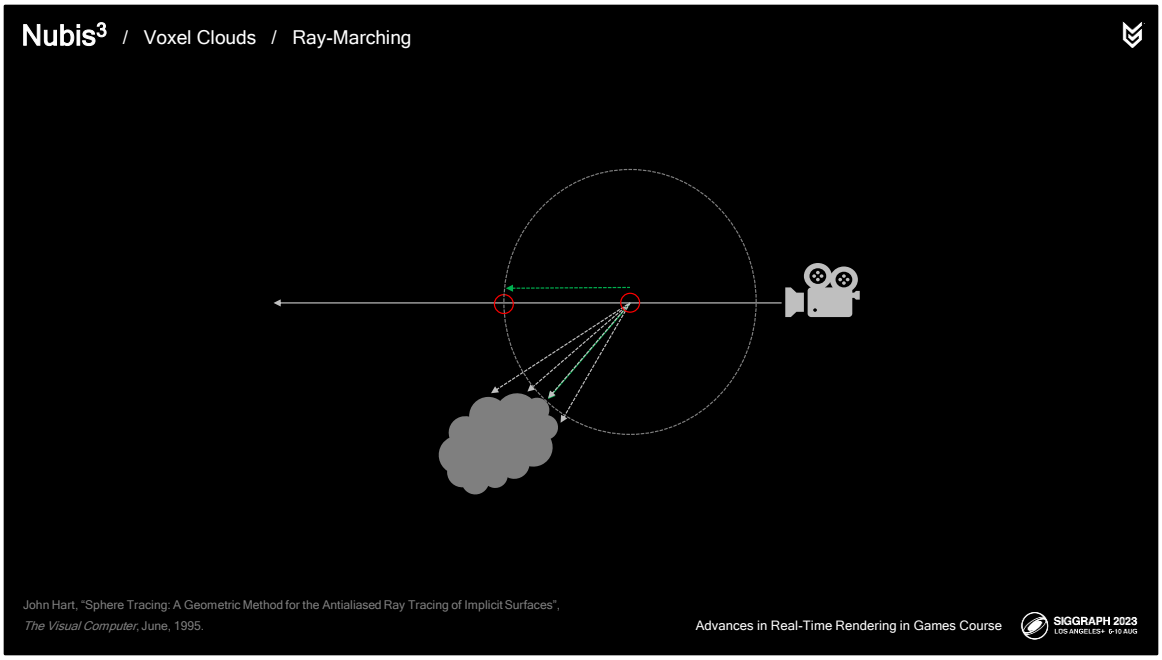


Recall our dilemma. Without some idea of where clouds are relative to each sample position, we cannot efficiently place successive samples along the ray.



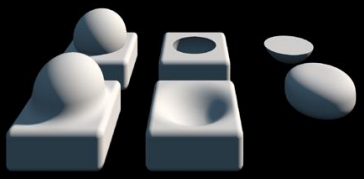
Recall that for the Envelope method clouds we combined cone step mapping and distance step mapping to help us better place samples around clouds generated from 2D authoring data. Well now our problem is truly 3-dimensional to begin with.

Fortunately, there's a technique we have been waiting to use for quite some time.



Sphere tracing is a technique that uses the 3-dimensional distance to the closest implicit surface to determine step size. A signed distance field stores measurements of these distances for every point in space. Positive values are outside of the object and negative values are inside.

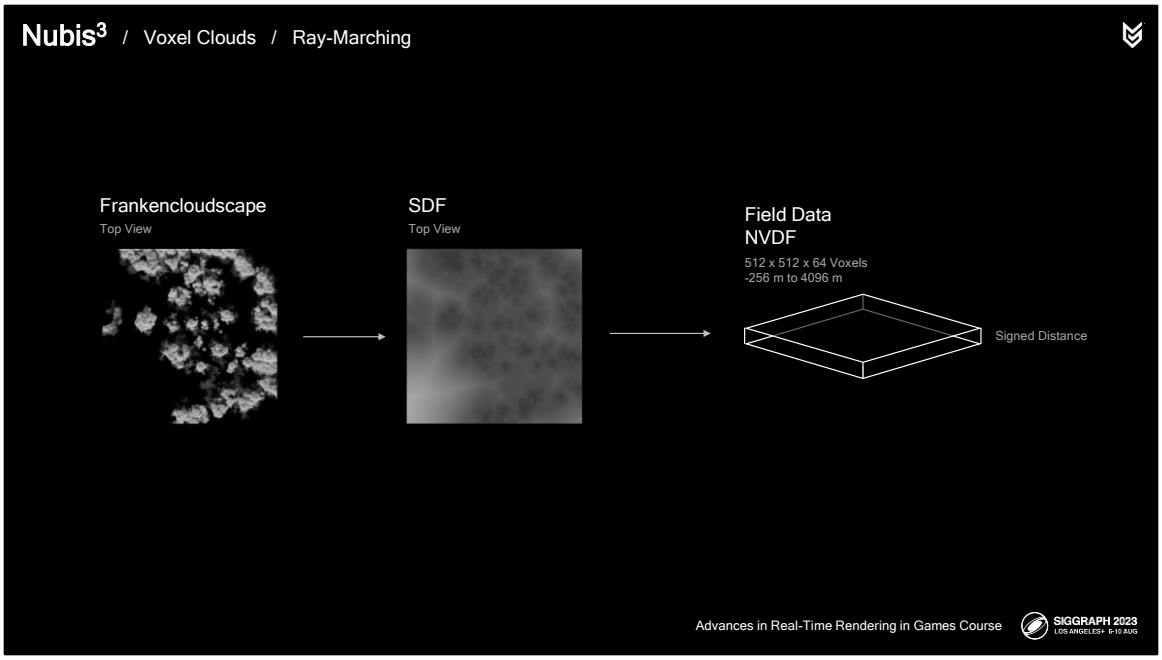
- When you are ray-marching through volumetric clouds, you can use this distance to take the largest step possible and reduce the amount of work along the ray.



https://iquilezles.org/articles/

https://www.secondorder.com/publications

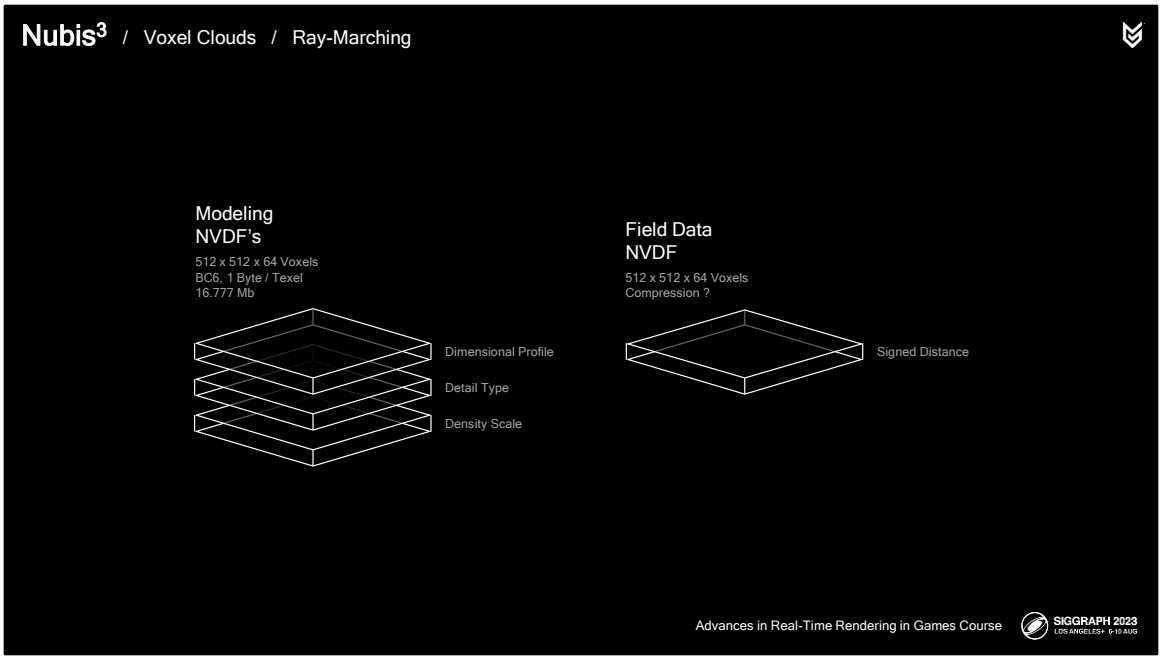
There are two really great sources for information about how to use Signed Distance Fields. The first is Inigo Quilez's website where you will find out how to calculate signed distances from a number of primitive shapes. The second are the Slides for Sebastien Antolnen's Claybook talk. –Yes, the Sebastian who spoke just before me today - Here you will find out how to use them effectively in high performance cases.



Once the artist is done building their frankencloudscapes

- we generate a 3d signed distance field and write it out as a separate NVDF. The data is originally ranged from -256 to 4096 but we scale it to a zero to 1 range. In the center

image you can see that the values increase the farther we are from the clouds in the image on the left.



We keep it separate from the modeling data NVDFs because we will be looking this up for every step and don't want to introduce a memory bottleneck for no reason. Speaking of bottlenecks, compression for this field is important.

Source:
https://www.reedbeta.com/blog/understanding-bcn-texture-compression-formats

Type Of Data	Data Rate	Palette Size	Line Segments	Use For	
BC1	RGB + optional 1-bit alpha	0.5 byte/px	4	1	Color maps Cutout color maps (1-bit alpha) Normal maps, if memory is tight
BC2	RGB + 4-bit alpha	1 byte/px	4	1	n/a
BC3	RGBA	1 byte/px	4 color + 8 alpha	1 color + 1 alpha	Color maps with full alpha Packing color and mono maps together
BC4	Grayscale	0.5 byte/px	8	1	Height maps Gloss maps Font atlases Any grayscale image
BC5	2 × grayscale	1 byte/px	8 per channel	1 per channel	Tangent-space normal maps
BC6	RGB, floating-point	1 byte/px	8-16	1-2	HDR images
BC7	RGB or RGBA	1 byte/px	4-16	1-3	High-quality color maps Color maps with full alpha

Precision Loss Symptoms

- Too low = extra steps
- Too High = rendering artifacts

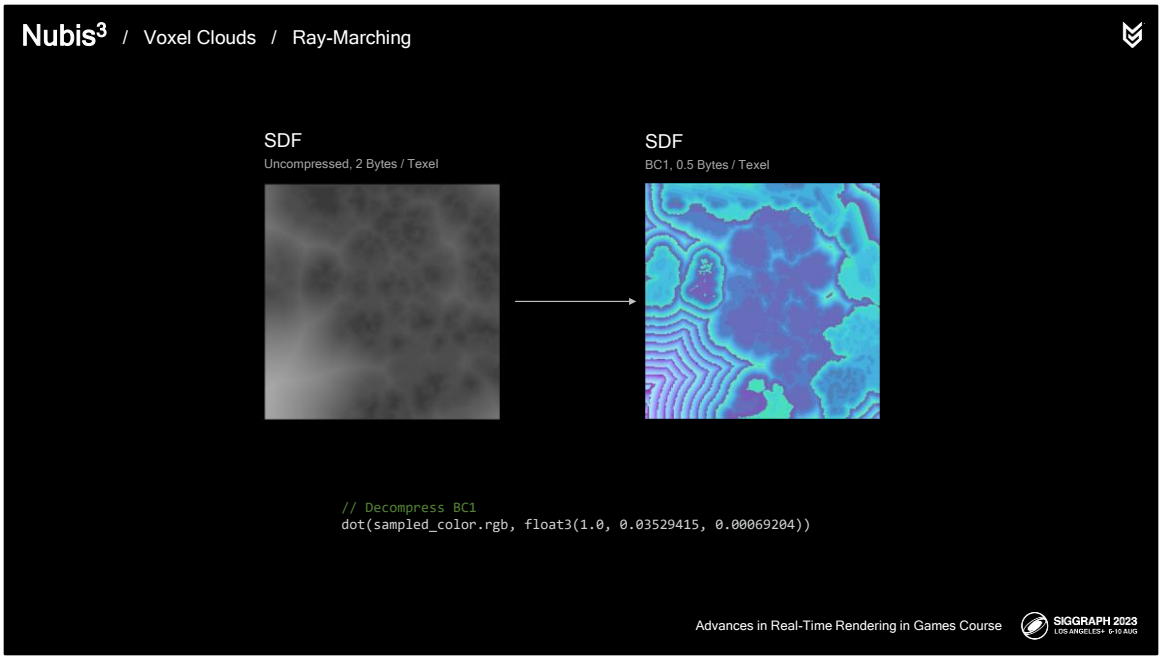
Here is Nathan Reed's world-famous BC Table to help explain this.

This SDF is sampled a lot, so compressing it would reduce the bits per texel and as a result reduce the memory bandwidth needs, which should improve

performance.

- A 1 Channel BC4 would seem ideal. The tradeoff is that fewer bits per texel means lower precision and that is a
- big problem for a SDF based Raymarch. If the decompressed value is too low it will mean the raymarch takes more steps than needed; if it’s too high it will cause rendering errors.

How do we solve this? With what those of us with an art degree call Rendering engineer black magic.



Using a custom BC1 Compressor developed by Senior Principal Tech Programmer Hugh Malan,

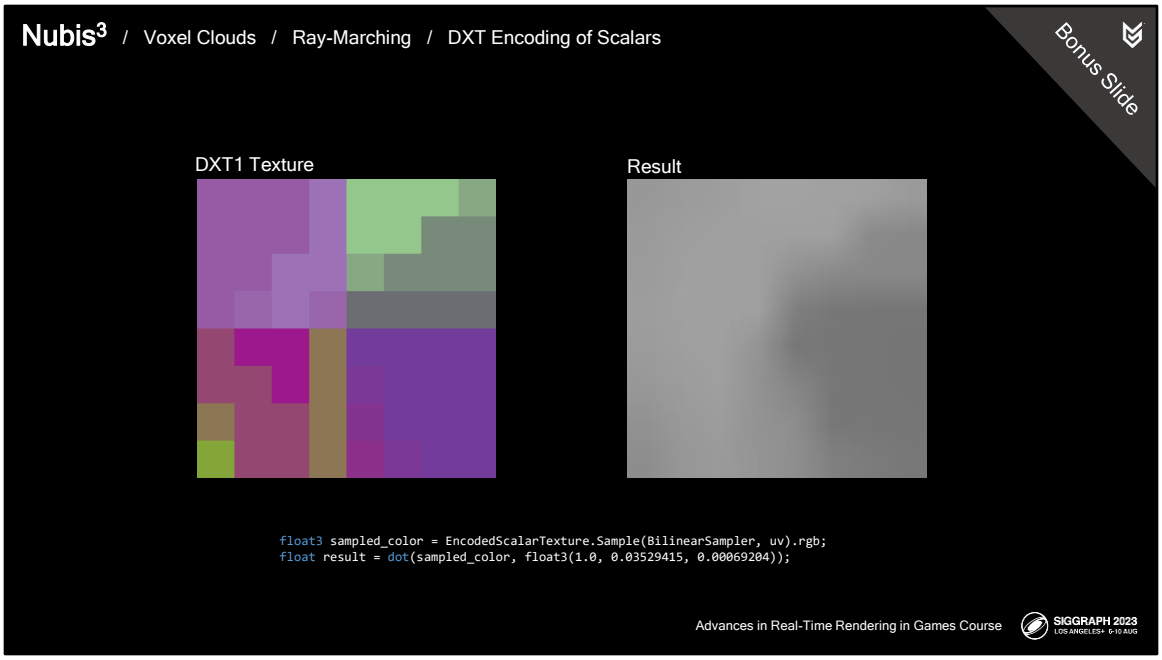
We

- compressed the SDF in a way that still gave close to 16 bits of precision.

There is a bonus slide right after

this one with an in-depth explanation of Hugh's work that you can look at, but to sum it up, we split up the data into three channels and then reassemble them after we sample the SDF in the ray-march shader.

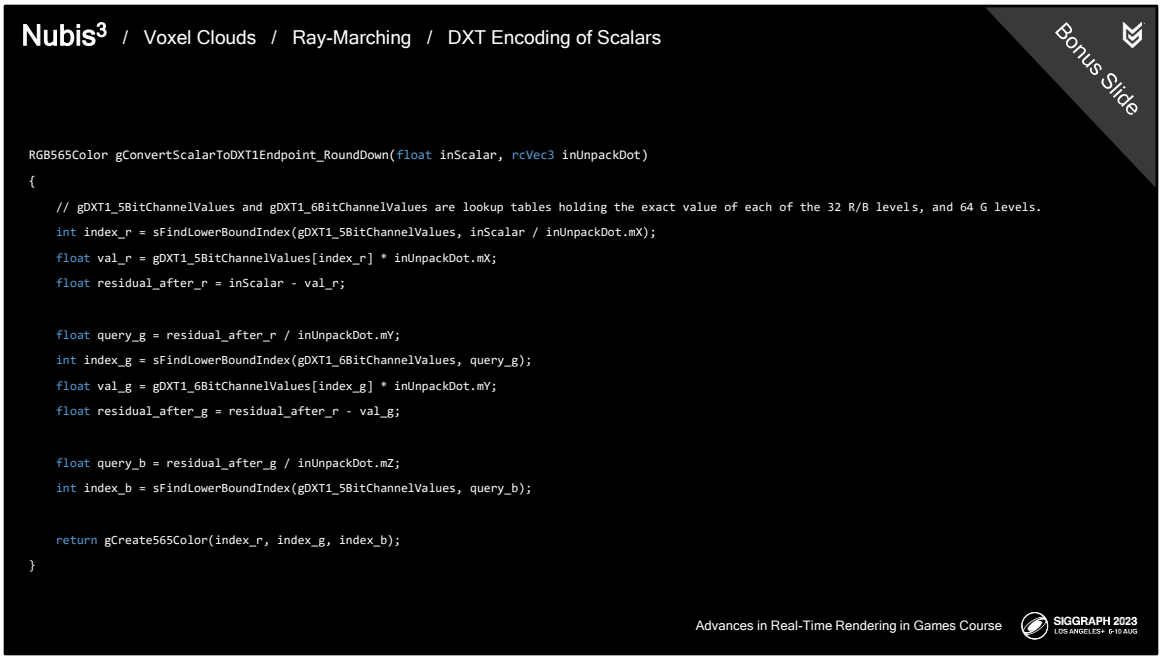
This approach worked because SDF data is smooth.
It turns out that depending on the length of the ray, this saves 10-30% over an uncompressed texture by reducing memory bottlenecks for our most frequently accessed voxel data.



For DXT1 each 4x4 block has two 16-bit endpoint colors; and two bits for each texel has which specify the blend between those two colors. We wrote a custom compressor that minimizes error. Since it's compressing a SDF for sphere tracing, we need to guarantee that the encoded, approximate value is less than the full-precision input value. If the approximation is ever greater than the ideal value, then the spheretrace can step too far and end up inside or beyond the isosurface.

The basic idea is to express the 16-bit value as a 565 RGB color: ie the red channel has the most significant 5 bits, G the next 6 bits, B has the least significant 5 bits. At runtime the shader would do a bilinear sample of the texture, and converts the RGB to scalar with a dotproduct with float3(1.0, 1.0/32, 1.0/(32*64)).

NB: a smoothly increasing gradient will result in a texture where the blue channel increases to maximum, then drops to minimum and the green channel increments. (And likewise for green and red.) Does the dotproduct result behaves correctly over this changeover? It does; here's a brief example. Imagine a texture unpacked by dotproduct with float2(10,1): ie G*10 + B. Two neighbours are 29 (G=2,B=9) and 30 (G=3,B=0). A bilinearly filtered sample midway should give the value 29.5. It does: G will be 2.5, B will be 4.5. The dotproduct is 2.5*10 + 4.5 = 29.5 as expected.



Unfortunately, the 5-6-5 RGB endpoint colors don't decode to [0,1] in a simple linear way implied on the previous slide, and the calculation of the two interpolated colors also has some complications. There's odd rounding, probably for cost or performance on the original GPUs, and that behavior has been preserved. These webpages have more details:

http://www.ludicon.com/castano/blog/2009/03/gpu-dxt-decompression/

https://fgiesen.wordpress.com/2021/10/04/gpu-bcn-decoding/

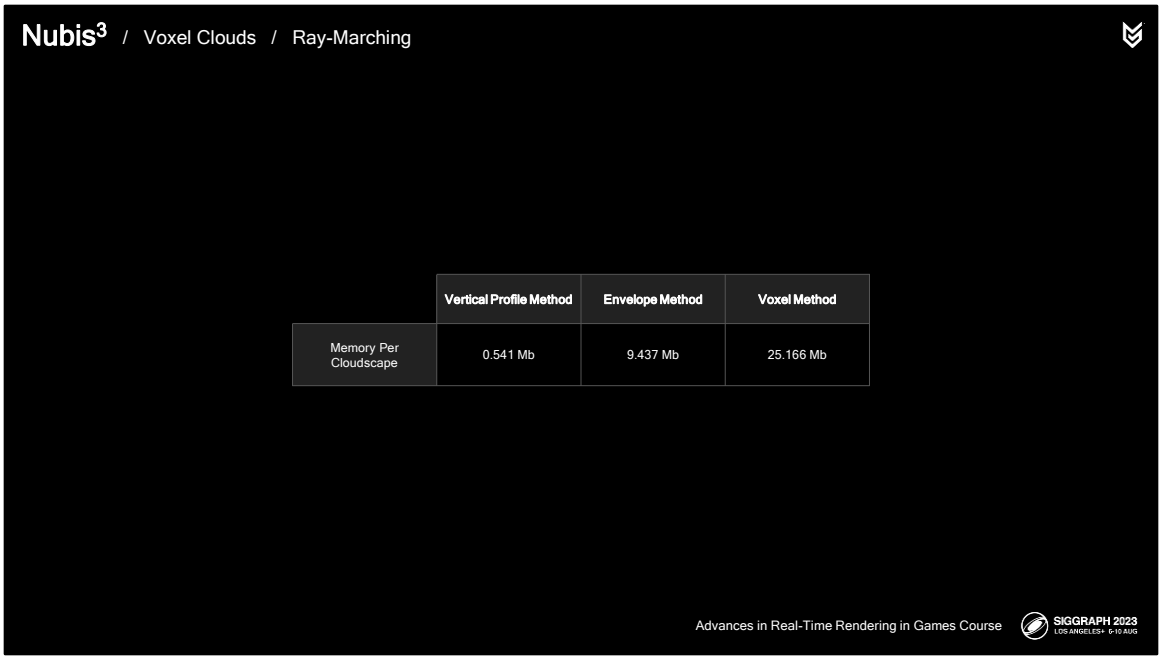
This means the dotproduct used for unpacking isn't so simple. The G value needs to be scaled up to cover the largest possible step in the R channel, and the B value needs similar treatment.

Once the dotproduct has been finalized, we can convert a scalar to an endpoint color with the pseudocode shown. First, the largest R value lower than the input value is found. There will be some error; the highest G value smaller than the difference is found. Then the B value gets the same treatment.

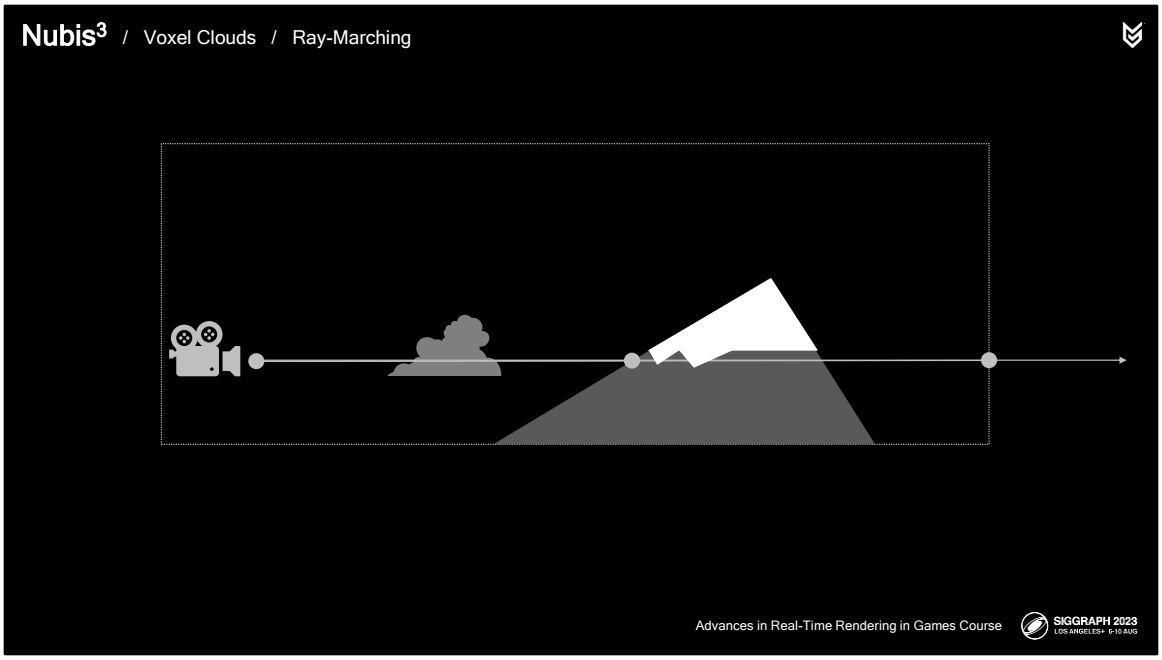
Once we have the two endpoint values, we can compute the two interpolated values

to generate the four possible values for each texel. Then for each of the 16 texels in the block, we find the option that's closest, while still below the full-precision value (to avoid the spheretracing errors described previously).

The obvious endpoint choice is to use the min and max values of the 4x4 block, but this means the encoded max color will only be used by one of the 4x4 texels. So we adjust the high endpoint value down to improve average error.



That’s the last of our NVDF’s for the Voxel Method, so let’s take a moment to compare the memory costs of each system.
VP = half a megabyte
EVM = 9 megabytes
VM = 25 megabytes, so you can see where all the extra detail is coming from.
Nothing comes for free but we are able to do so much more with voxel clouds than the other methods.

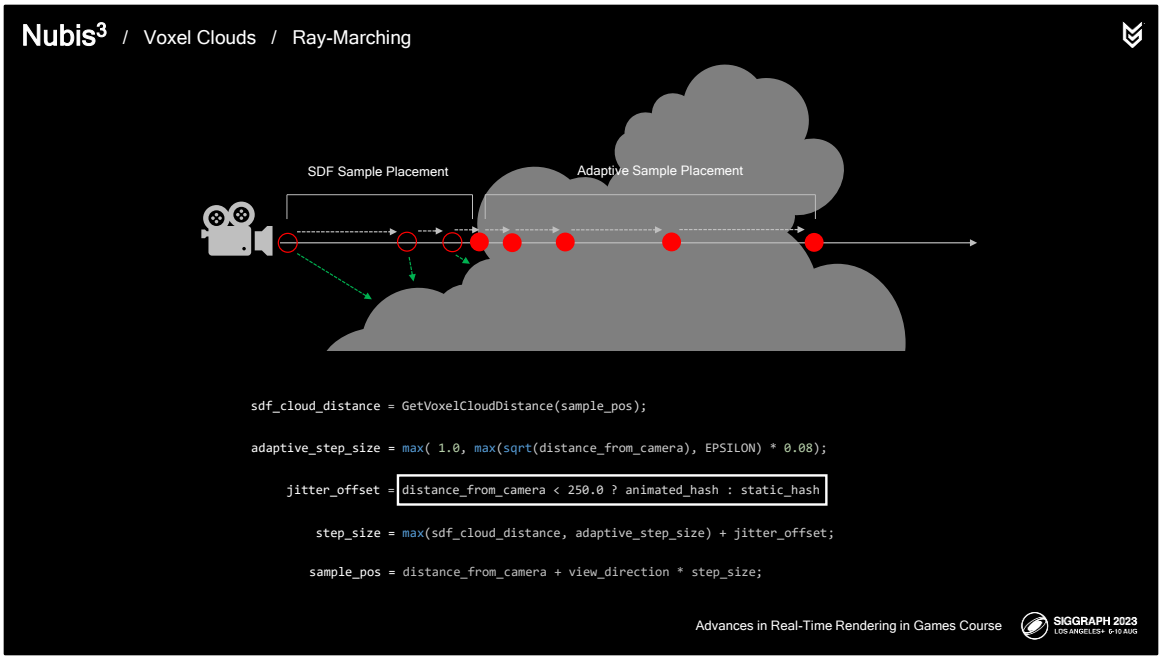


But this is not the only way that we accelerate a voxel ray-march. First, we define bounds for our ray to limit the amount of possible work.

- The ray starts at the camera and
- ends where it intersects the

voxel grid bounding box or

- where it intersects geometry, whichever is closer.



Next, we calculate the size of the step to the next sample. This distance is built from 3 sources.

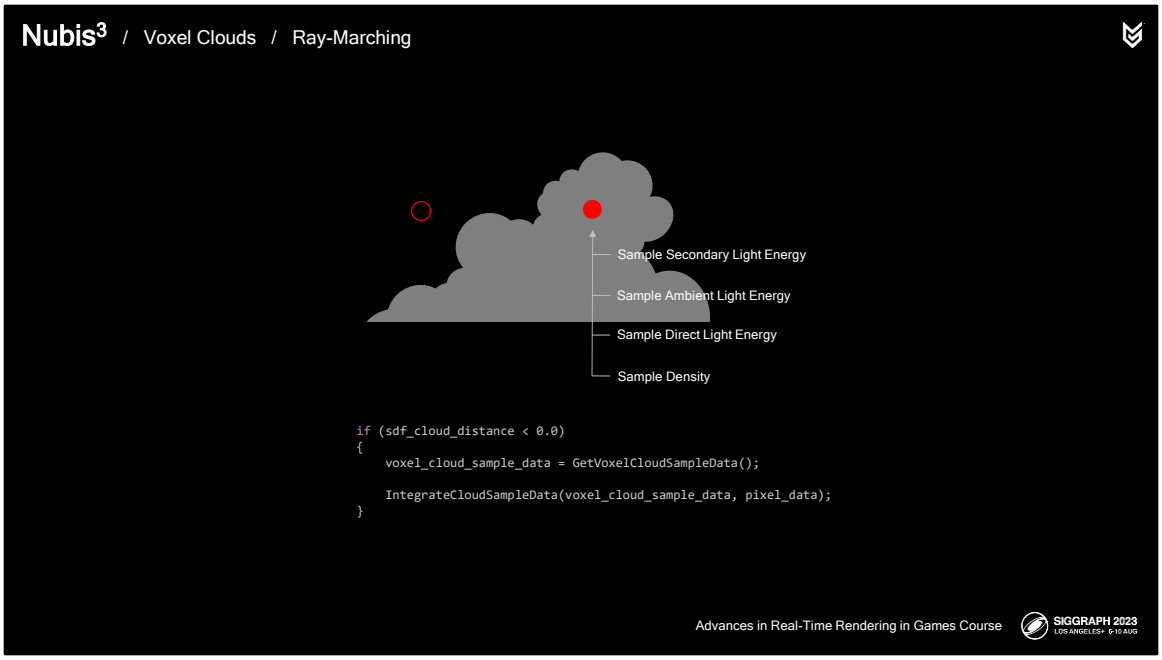
- The signed distance
- the adaptive step size which gradually increases over distance
- and a temporally jittered offset

designed to counter under-sampling noise.

- The SDF is used to place steps optimally outside of clouds.
- While The adaptive step size increases over distance from camera and is used to place samples inside of clouds. This way, clouds that are far from camera can benefit from larger step sizes While samples between clouds can be placed as optimally as possible.
- `The jitter switches from animated close by to static far away to reduce under

sampling artifacts without making distant clouds shimmer.

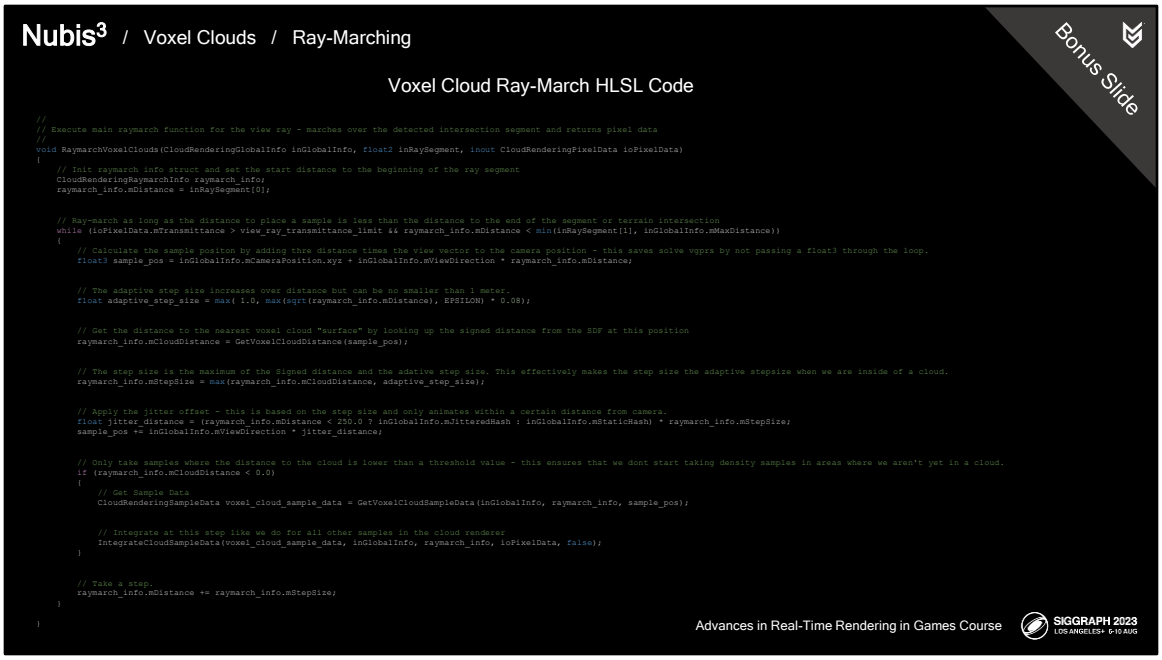
- The signed distance is maxed with the adaptive step size so that as we sample inside of the cloud and it becomes negative, we switch to using the adaptive step size.
- This step size is then added to the sample distance so that it can be used to build the sample position for the next sample.



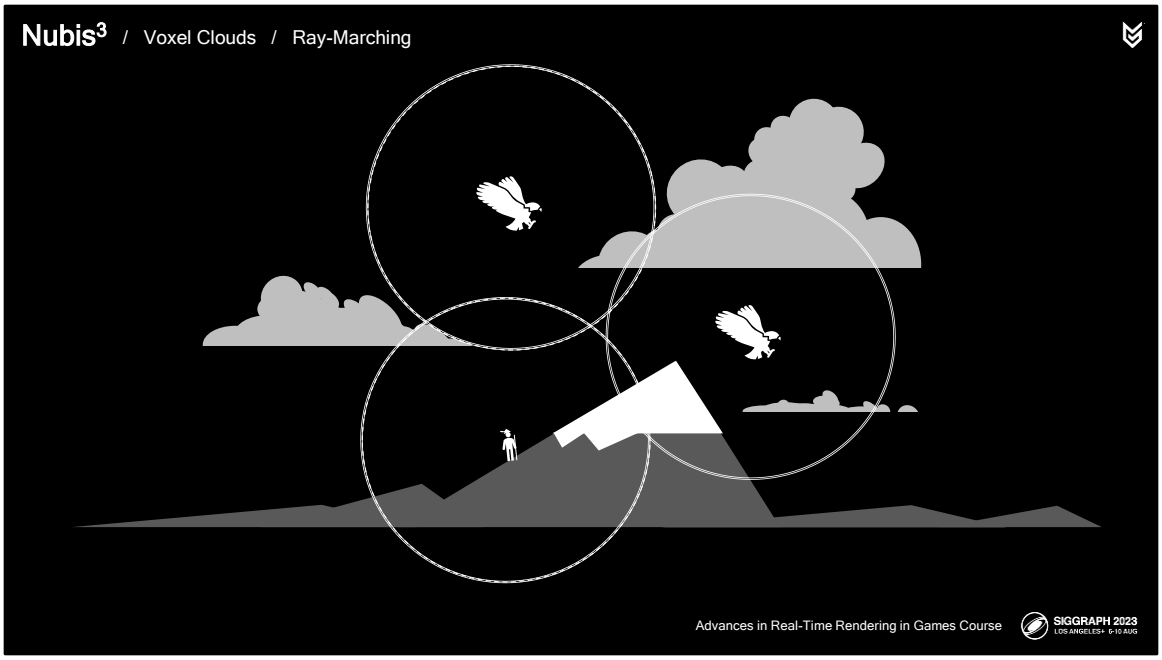
When it comes to taking a sample, we look to the SDF for an optimization.

- If the SDF is greater than zero we are outside the cloud and don’t need to collect any samples.
- and if it is less than zero, we

collect our density and lighting samples as described before and integrate them along the view ray.



Implementation of ray-march



For the past half hour I have talked a lot about performance gains in specific cases in terms of percentages.

It’s time to look at performance systemically.

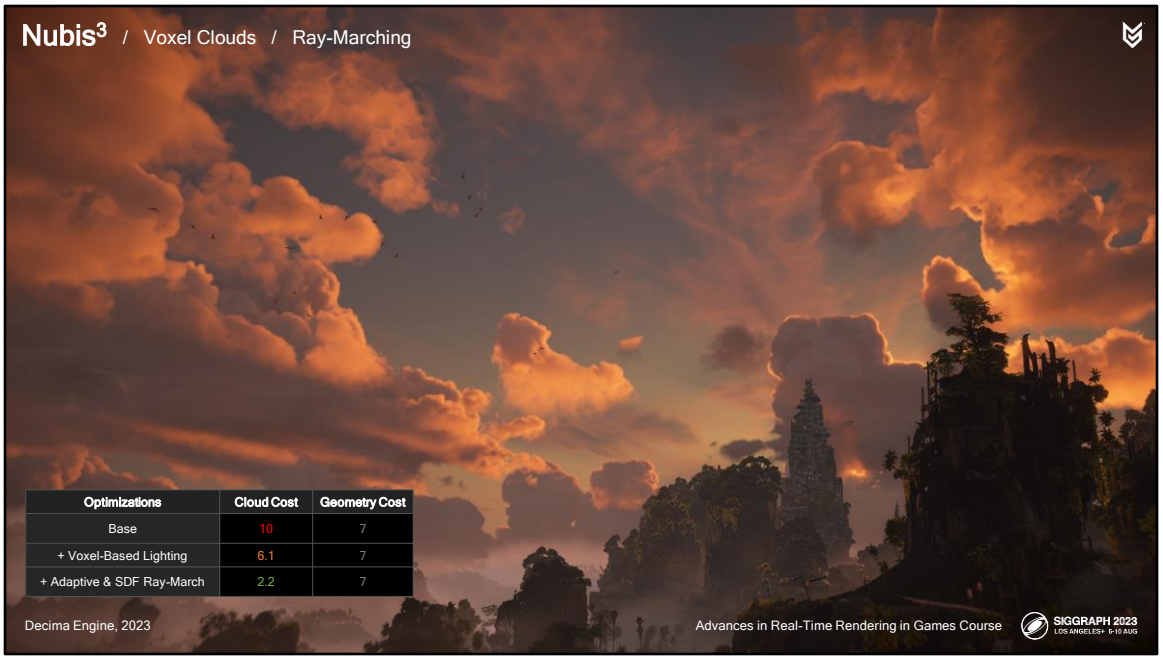
It is important to keep some things in mind with the numbers that you are about to see:

- When you are flying through clouds, then that is the majority of what you are rendering.
- When you view them from the ground at a distance, their cost should scale in proportion to the rest of the content. Making that rebalance work dynamically is just as key for clouds as it is for polygonal game content and this was our measure for success in this system for this game.

So, we are going to look at a

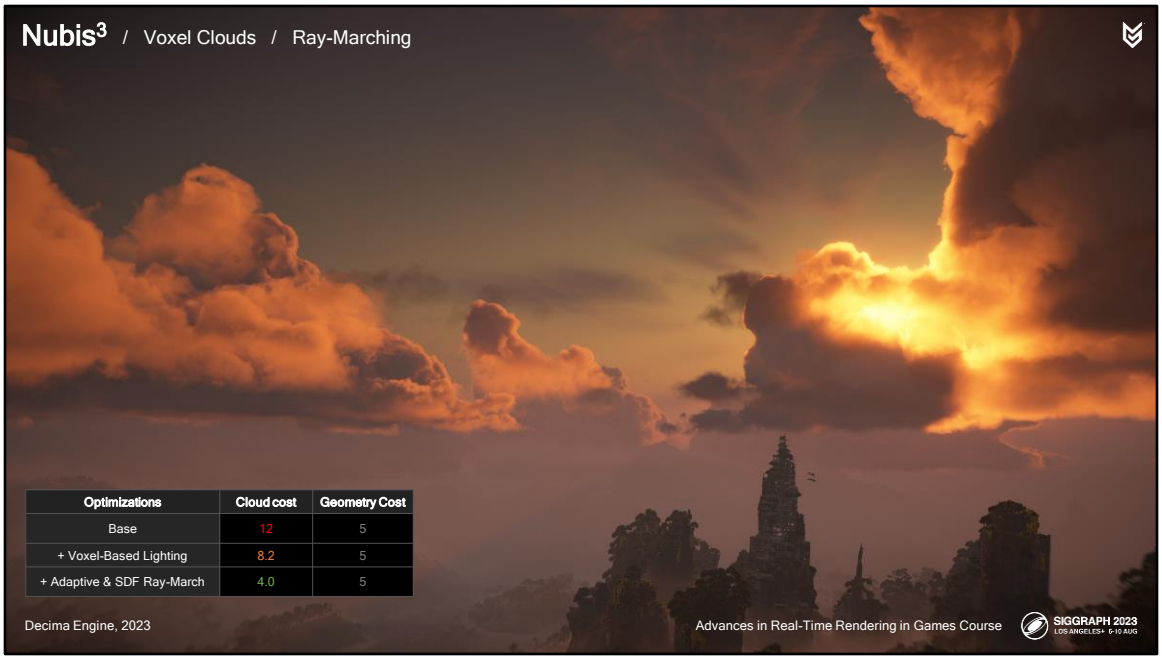
- view of clouds from the ground,
- One with mixed ground and cloud content
- and one that is fully clouds.

Each of these renders is done at 960x540 resolution



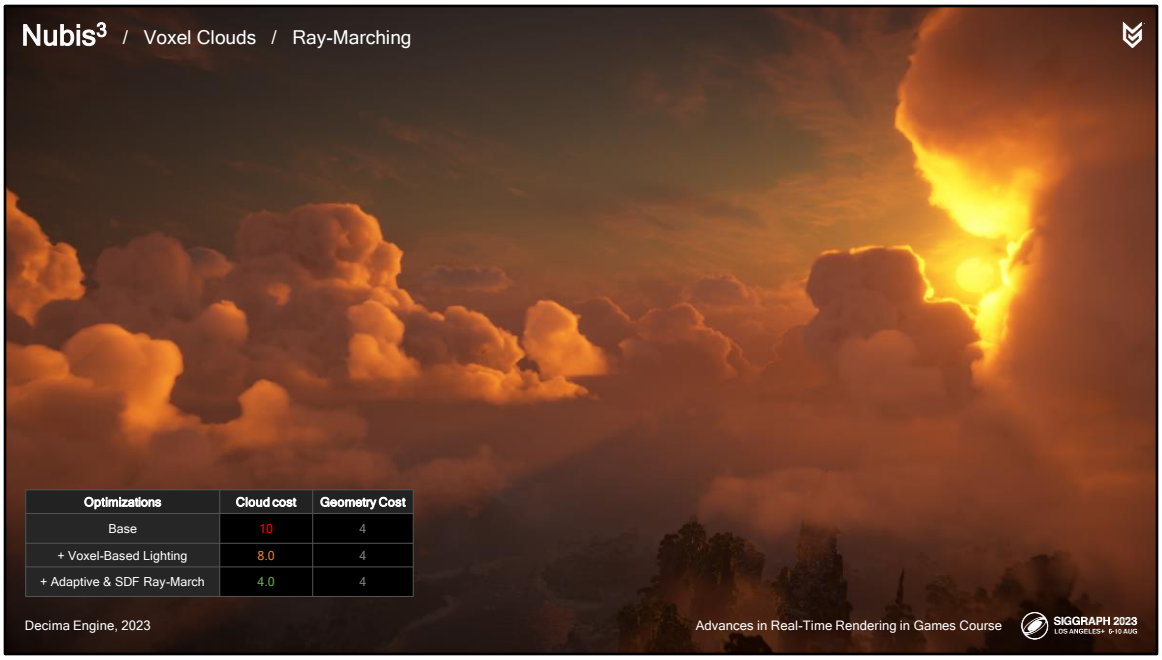
Starting with the ground, and with no lighting or ray-march optimizations, the cost is 10ms
With the Voxel-based lighting optimizations, it goes to 6.1ms
And with the ray-march optimizations it goes to 2.2ms

Take a look at the cost of the geometry pass. This and clouds combine to 9.2ms.



As we move to the air between clouds and visible landscape, and with no lighting or ray-march optimizations, the cost is 12ms – this increase is because there are a lot of rays that just glance along the undersides of these clouds. This can be mitigated by

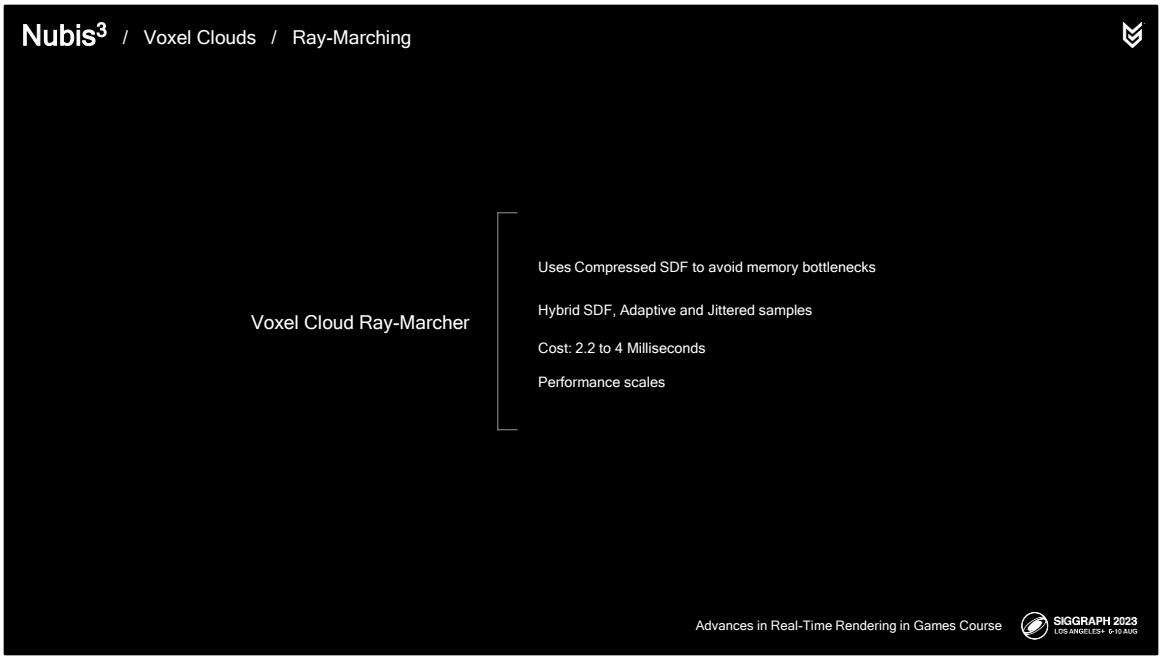
offsetting the heights of these clouds to shorten the rays.
With the Voxel-based lighting optimizations, it goes to 8.2ms
And with the ray-march optimizations it goes to 4ms
Take a look at how the cost of the geometry pass went down by 2ms while the clouds went up by 2. We are still in balance.



As we move to full immersion in clouds, and with no lighting or ray-march optimizations, the cost is 10ms
With the Voxel-based lighting optimizations, it goes to 8.0
And with the ray-march optimizations it goes to 4.0.

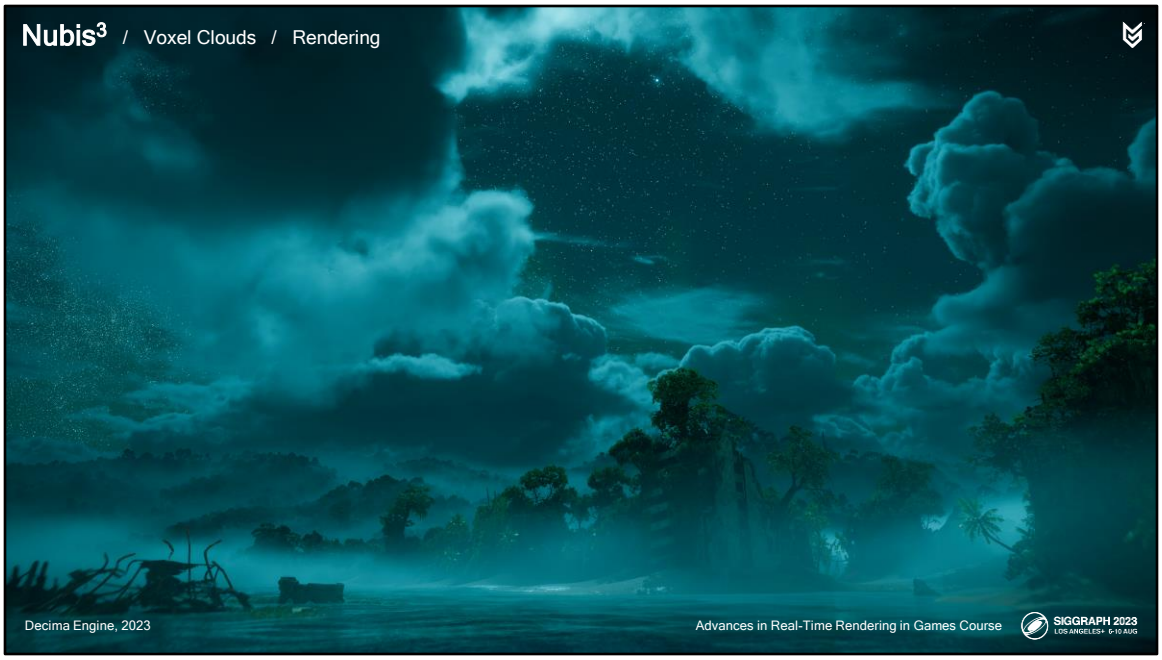
Now look at the cost of the Geometry pass, it went down by 1ms and the clouds stayed the same.

So there is a nice handoff of costs between geometry and clouds from one situation to the next.



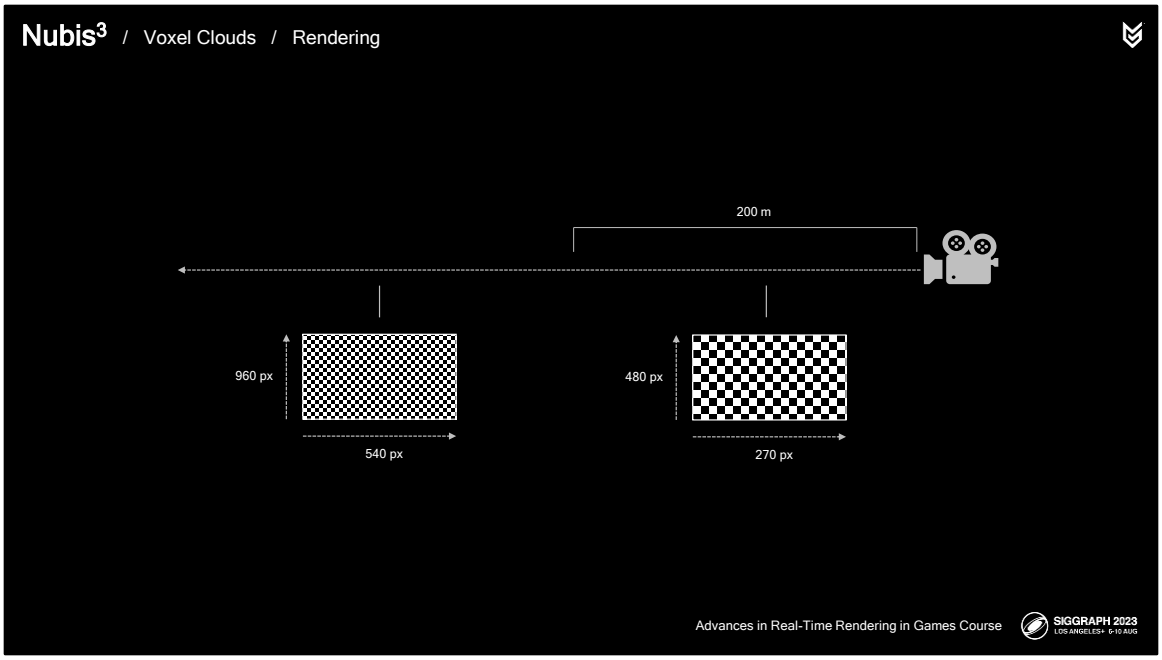
So In summary:
We use compressed SDFs to avoid memory bottlenecks for samples that get taken for every view-ray step.
We combine the SDF, Adaptive step size and Jittered sampling to get the best of both worlds while

mitigating under-sampling noise.
Costs range from 2.2ms to 4ms depending on if we are on the ground or in the air.
Performance scales properly from the ground into full immersion.



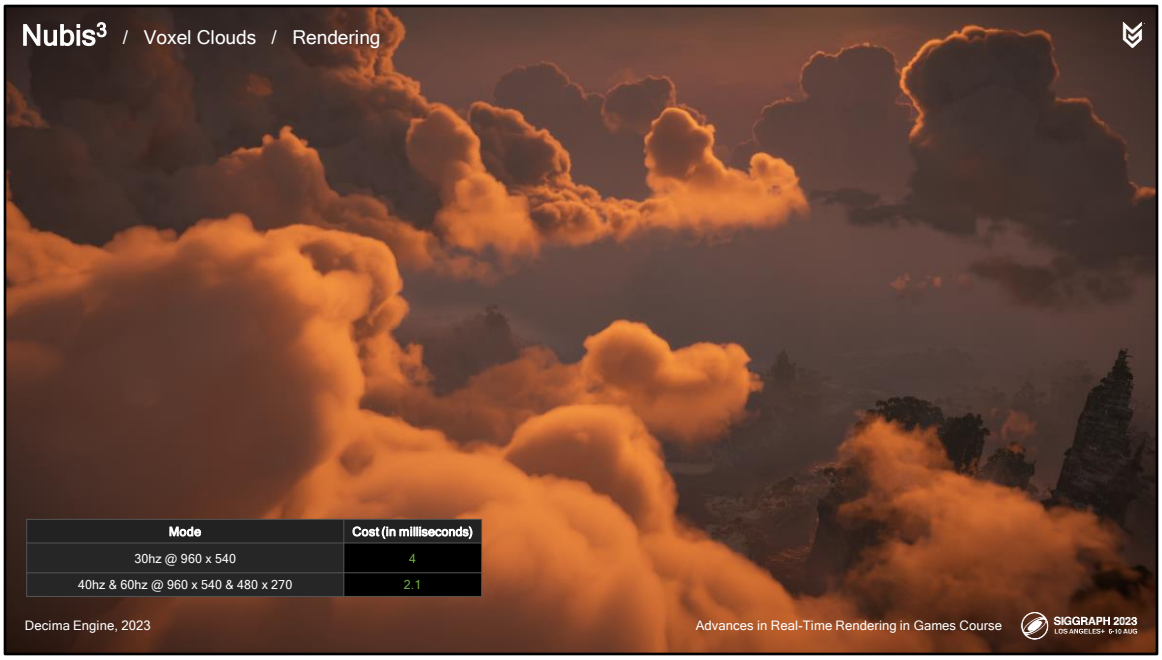
30hz, 60hz, 40hz. Potato, Patato….. Puh-tader? Everyone has their favorite way to play. Our job as graphics programmers and artists is to make sure that all modes are supported. And while the performance just squeaks by for 30hz mode, we

must still contend with high framerate modes, where half as much time is available to render a frame. .



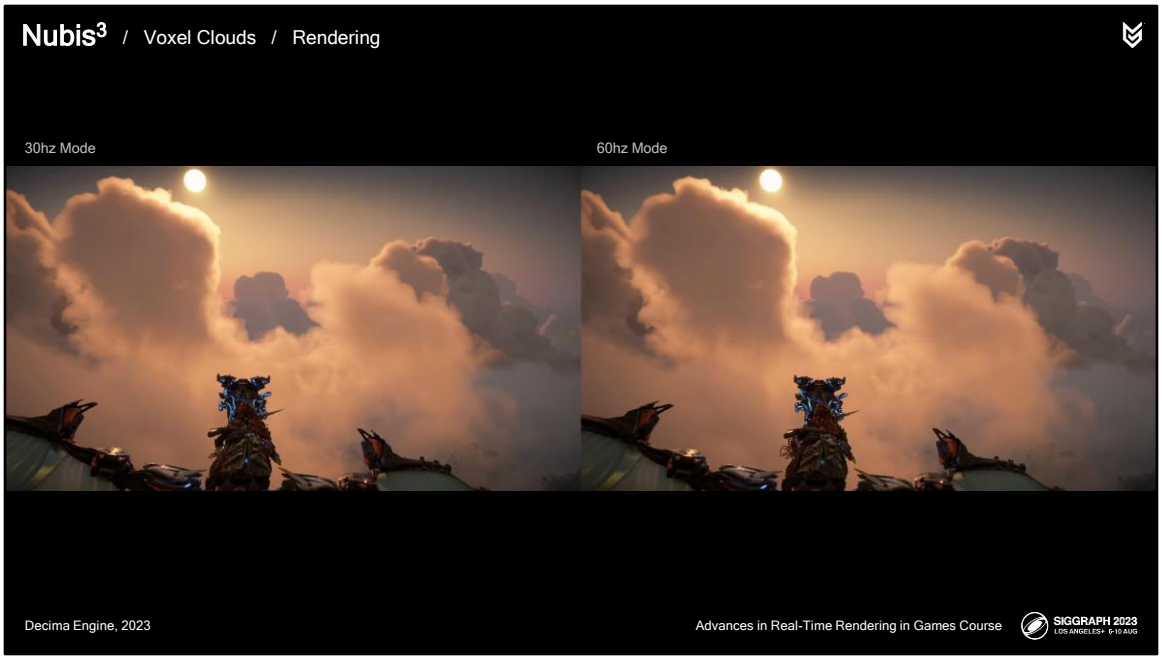
Recall that for the Envelope Method, we did away with temporal upscaling and split the render into two passes: High res in the distance to prevent aliasing and low res up close to improve performance for the most expensive parts of the ray-march.

Would a similar approach work for 40 and 60hz modes? The answer is yes. For High framerate modes, we do this exactly the same way.

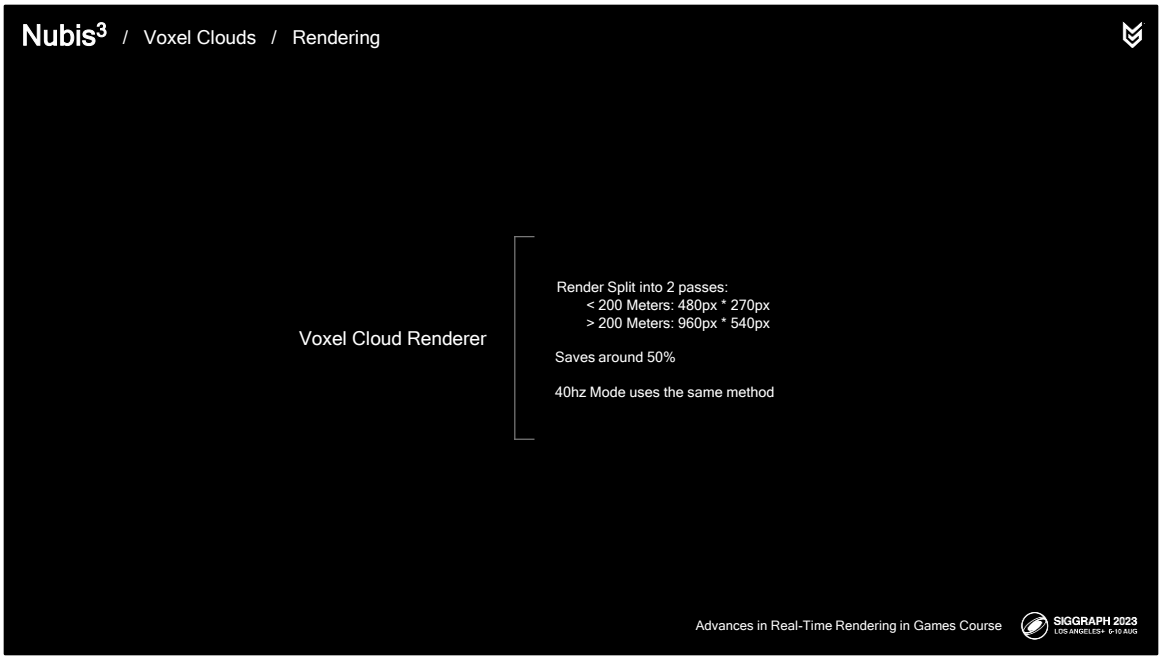


Here is an example:
This frame cost 4ms in 30hz mode. But when we split the render into two resolutions over depth the cost goes to 2.1ms. We used this solution for both 40 and 60hz modes.
Let’s take a look at how the two

approaches compare side by side while in flight.



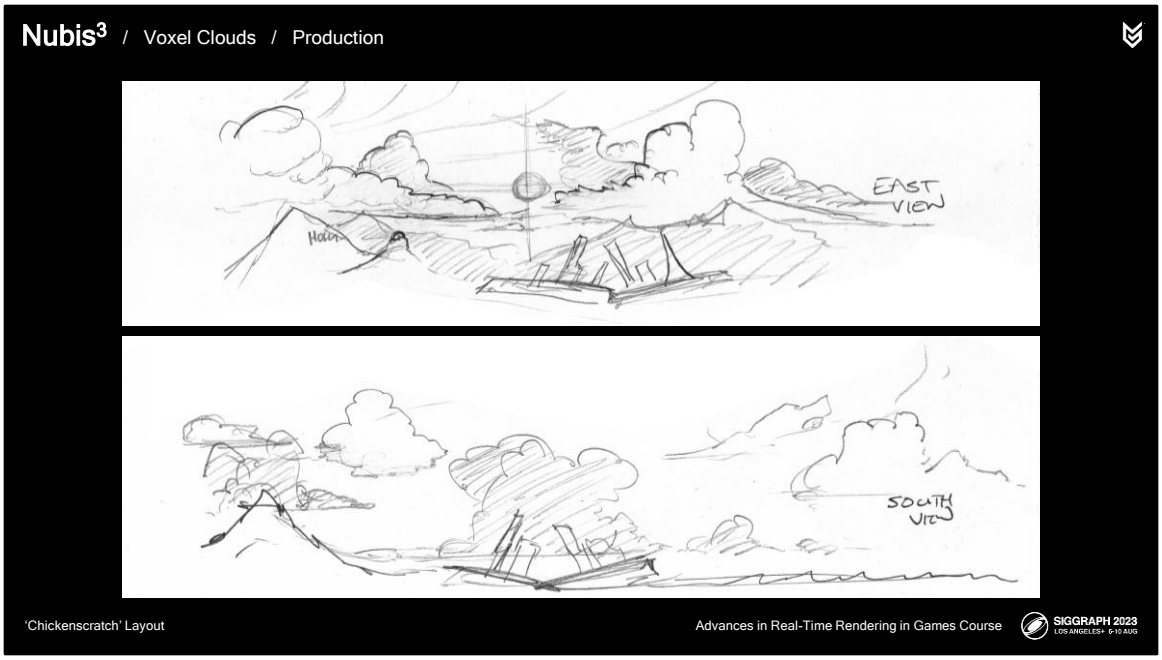
You can see that there is a bit more pixelation on sharp and dense features of the clouds on the right hand 60hz video, but lower density regions hide this better. Aside from this the result is pretty successful because the experience itself hasn't changed.



So, In summary:
We split the render into two passes; low res up close and higher res far away.
This nearly cuts render time in half, which as you know is good considering we have half as much time to draw a frame in

60hz mode.
40hz uses the same method.

Now let’s look at some practical applications for this system in our game, Starting with how we built a cloudscape that works as both a background element and an explorable environment.

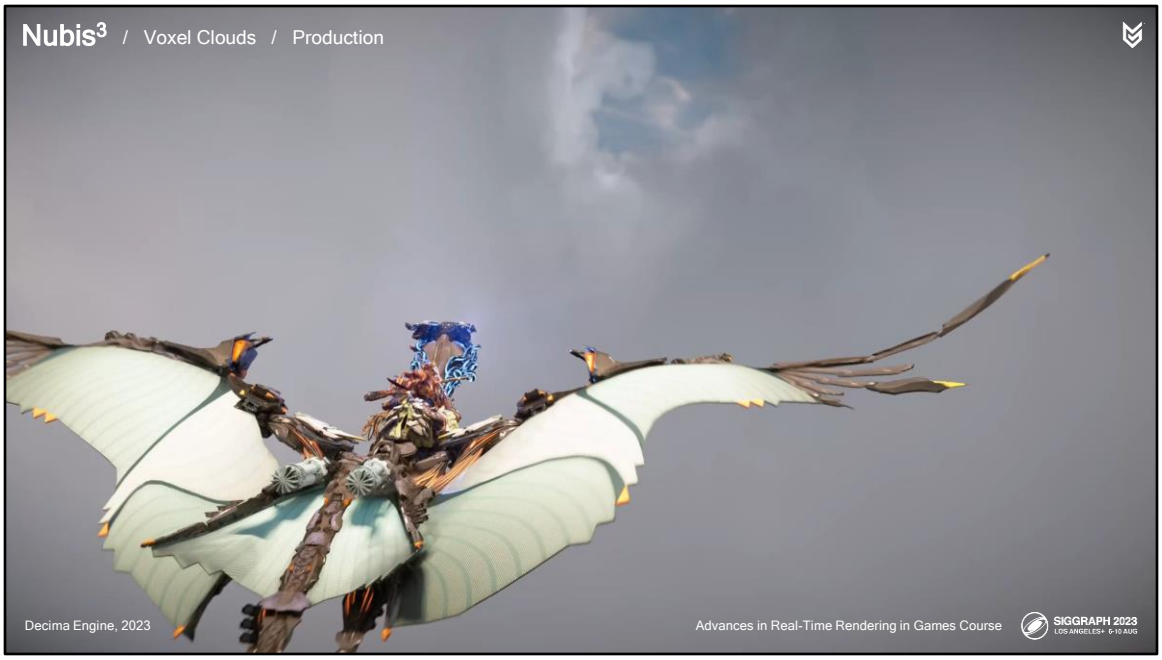


This is a “chickenscratch” doodle that I made of post apocalyptic Los Angeles. It helped to lay this out along the east-west and North-South views because these are the directions against which we need a good composition of silhouettes in

terms of the arc of the sun.

We start by placing a large cloud so that it looks good from the ground from multiple angles at all times of day.
Next, we add other features and connective tissue to integrate it into the rest of the cloudscape
For some clouds we added

tunnels and caves using the Atlas tools to enhance exploration.



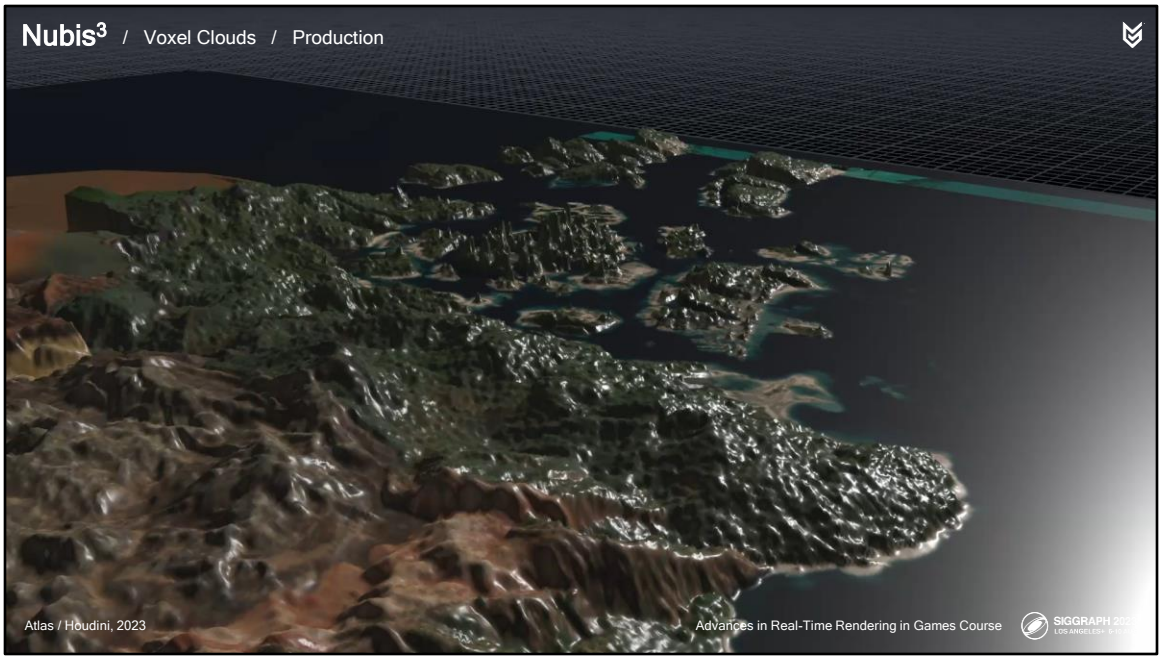
We would also fly around and keep an eye on what it looks like from the sky as well. - though not this fast.
And cave diving was actually a lot of fun, so it was hard to distinguish work from play at times.

Lets take a look at another Tunnel Diving video at normal speed.
Notice how the quality of light changes inside the cloud and toward the backside of the tunnel. The lighting just works.
Also see those little streamers on

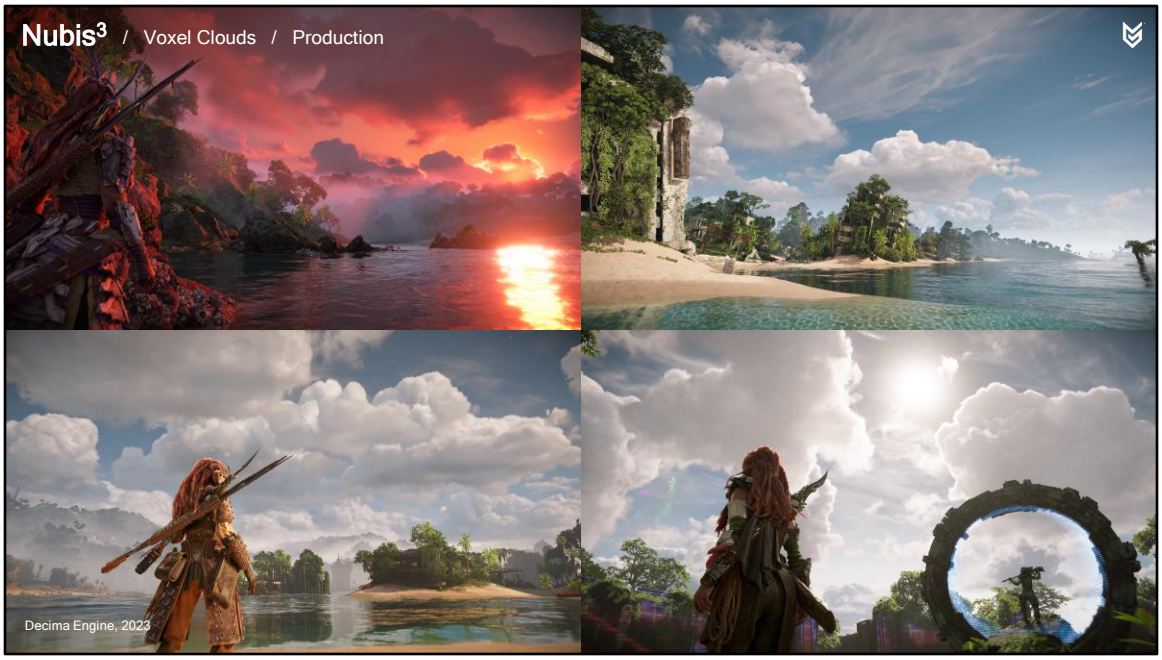
the wing tips? We emitted particles there whenever we detected intersection with the cloud.

And just in case you are thinking, “Clouds don’t have tunnels like this.”,

- Here is a photo I shot last Friday.
- And this is just a bird, not a mechanical dinosaur.

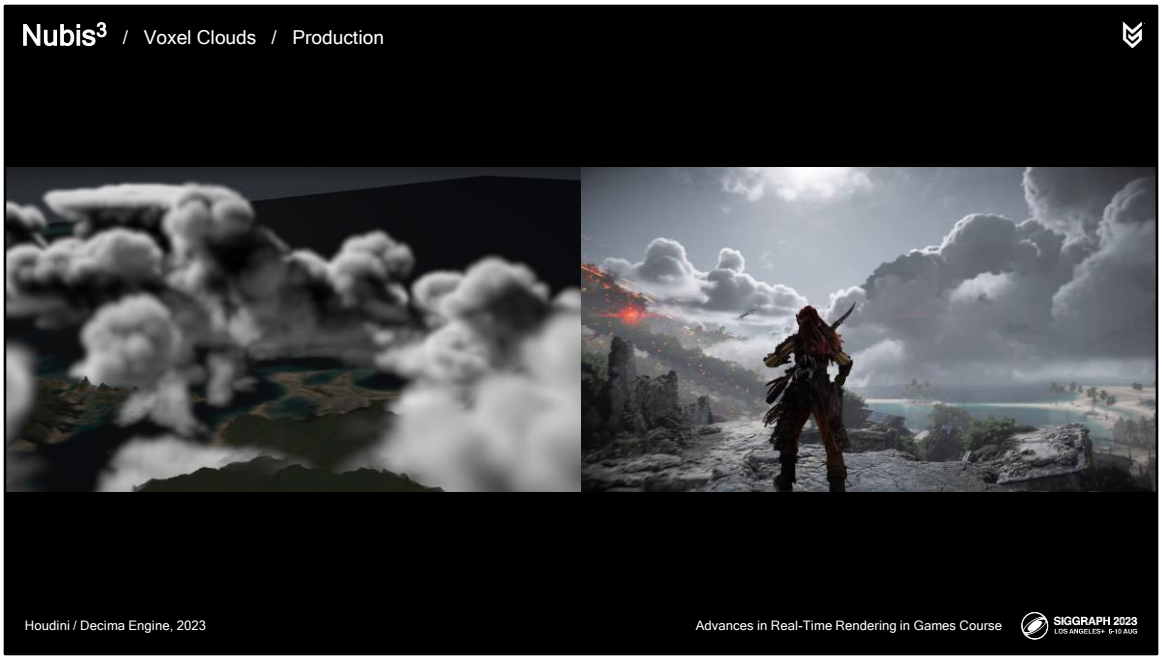


We did this multiple times for each cloud formation and combined them all into a cloudscape.

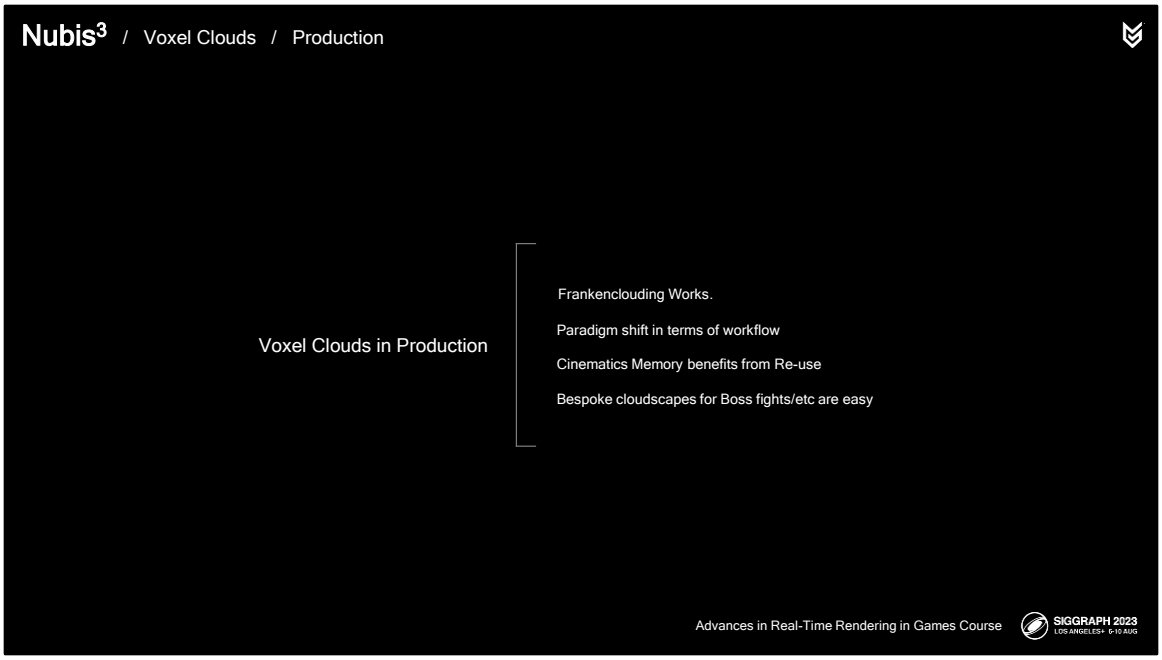


Here are few examples of what the gameplay cloudscape looked like from various locations.

For cinematics we could reposition the cloudscape per shot using a transform, allowing us to create variation without increasing memory.



For the boss fight at the end of the game, we built a cloudscape that wrapped around the playable area sort of like an arena.



So, In summary:

Modeling clouds with voxels is easier using frankenclouding

Cinematics can make use of existing cloudscapes using transforms to save memory
Bespoke environmental cloudscapes possible like for things like bossfights.

Well, we have almost reached the end of this story, so now it is time to start looking back at what we learned.

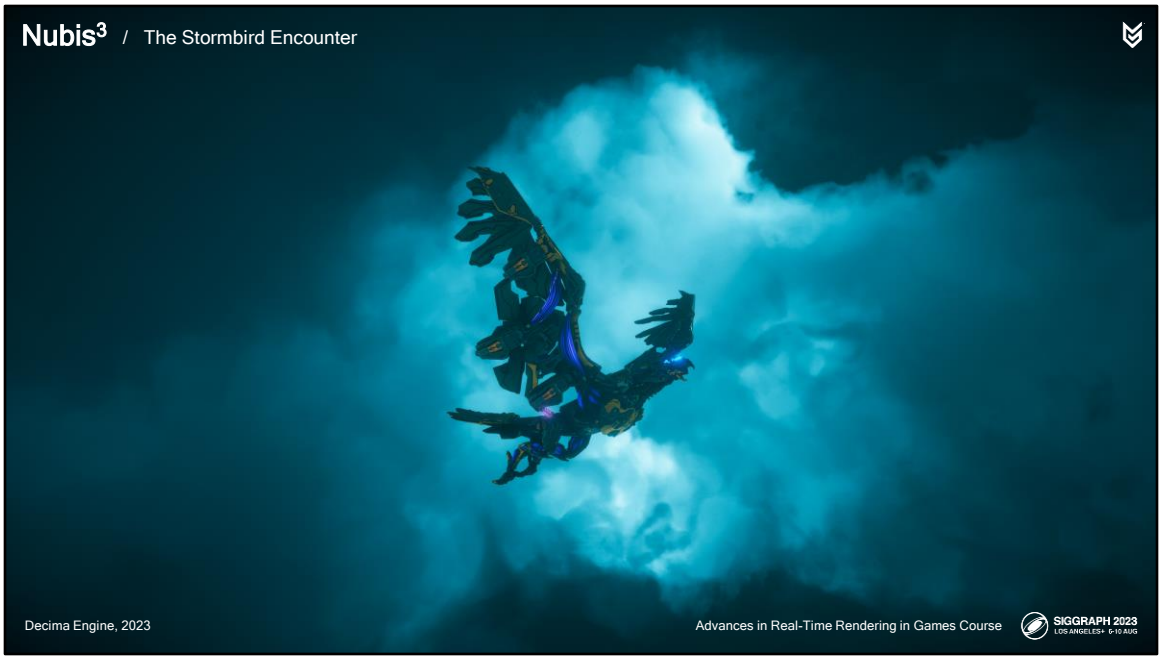
What we now call the Voxel Cloud Method turned out to work pretty well.
You can see that compared to before, Voxel clouds check almost all of the boxes.
Evolution is something that should be developed further in

the future, but for this you will not only need to solve real-time voxel-based cloud evolution itself, but also generate signed distance fields on the fly. Something for the future.
We call this a success. Rarely do we get to keep everything that we wanted while creating something more advanced.
One takeaway worth mentioning here is that it is not all about the voxel density, it is about the perceived visual complexity of the result. Working at higher resolutions is certainly one way to do this, but as we have shown, there is usually more than one

way to do a thing in computer graphics.

Let’s take a moment now and enjoy the experience of flying through clouds.


Nubis3 / Conclusions
Decima Engine, 2023
Advances in Real-Time Rendering in Games Course
SIGGRAPH 2023
LOS ANGELES• 6-10 AUG

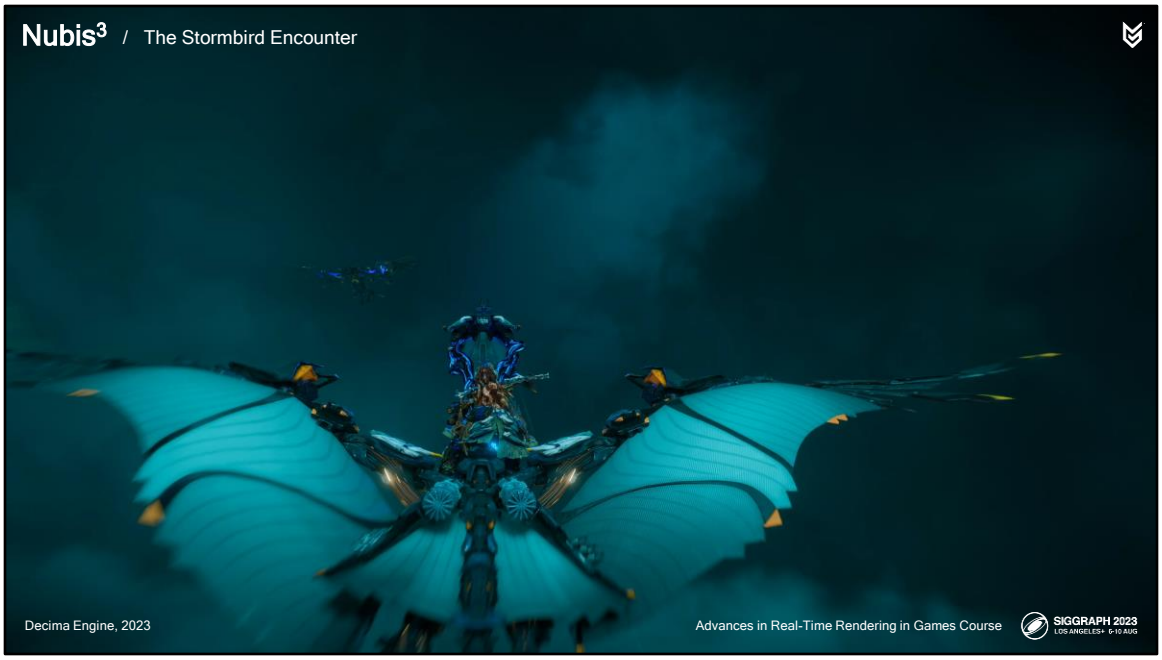


…But, before I go, there is just one more thing to share.
We wanted to do more than just offer flight through clouds.
We also wanted to create a unique gameplay experience that took place in a storm cloud.

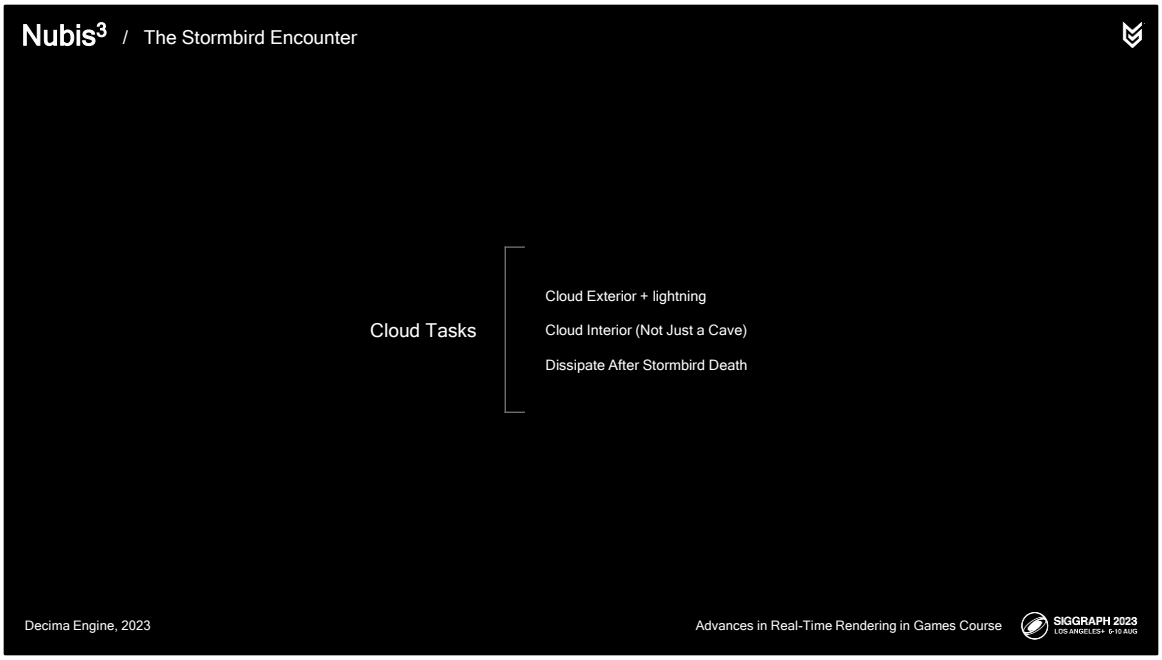
The Stormbird had a lot of lore behind it. This is an apex predator of the skies that can locally control weather to create small storm clouds with lightning. So why hadn’t we shown this in any of our games? Quite simply, there was no way to technically do it. Until now.

Principal World Designer, Elijah Houck had a sort of cat and mouse game in mind for an encounter. It’s a big apex predator and you are flying around on this little bird getting in its space. It might think you are cute and toy around with you a

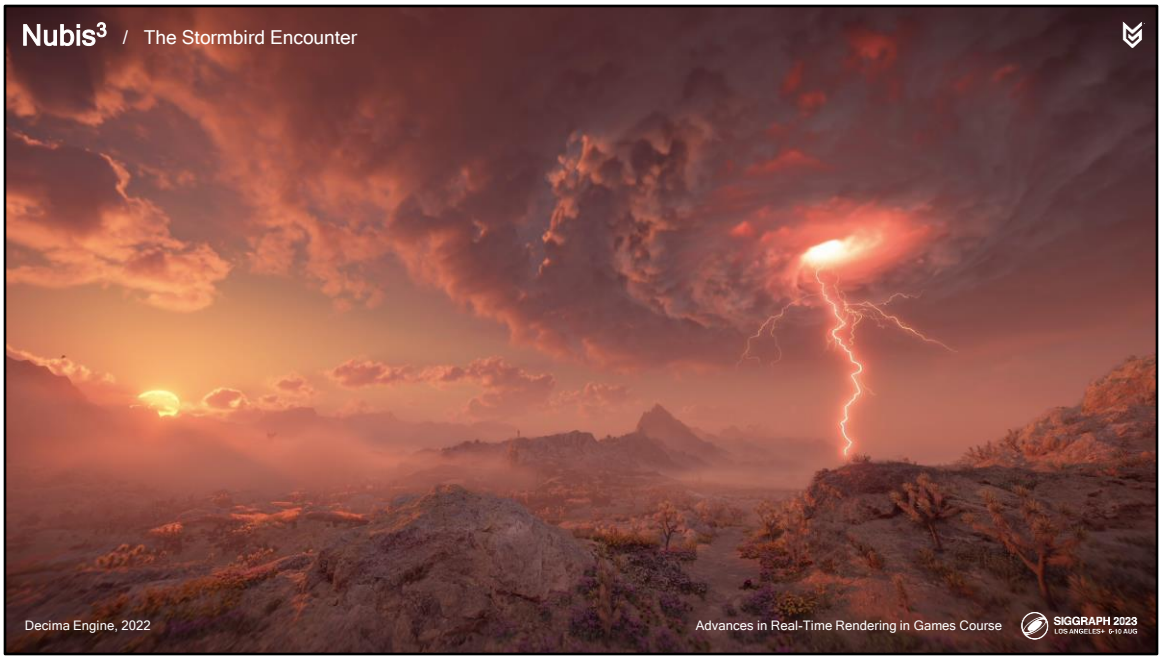
bit before killing you. Elijah envisioned it luring you into its lair – a storm cloud – and then launching several playful sneak attacks on you before finally killing your mount to see if you survived the fall. Thankfully, alloy has a glider. Let’s look at the encounter.



… as you can see the cloud dissipates after you kill the stormbird.



So, the task for me was to design the cloud exterior, complete with lightning, design the cloud interior - and most importantly make it something more than a just another cloud cave. And finally, make the cloud dissipate once the player kills the bird.



We wanted it to appear as a highly localized thunderstorm, not something massive like the Superstorms from Forbidden West.
What would something that small look like?

This was an early attempt to create the cloud from a fluid sim that resembled a standing plume with skirt of wispy material at the bottom.

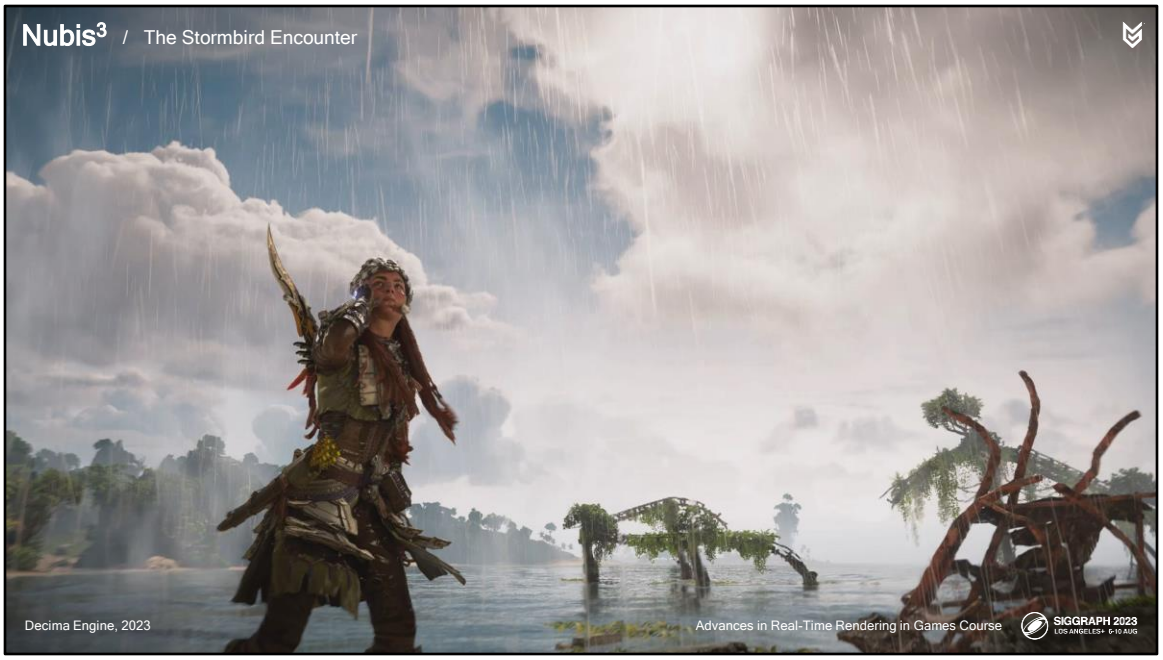
Here is what it looked like in game with some additional inner glow courtesy of the superstorm glow code from forbidden west. This result altered the course of the development of the storm cloud in two ways.
First, we realized that this was far

to small to host an internal structure large enough for the encounter that we wanted to create.

So, naturally, I scaled it up and frankenclouded the heck out of it. Here is the final model.

Second, notice how the glow suggests an opening on the underside. It got us thinking that we could use internal lighting to reveal the opening.

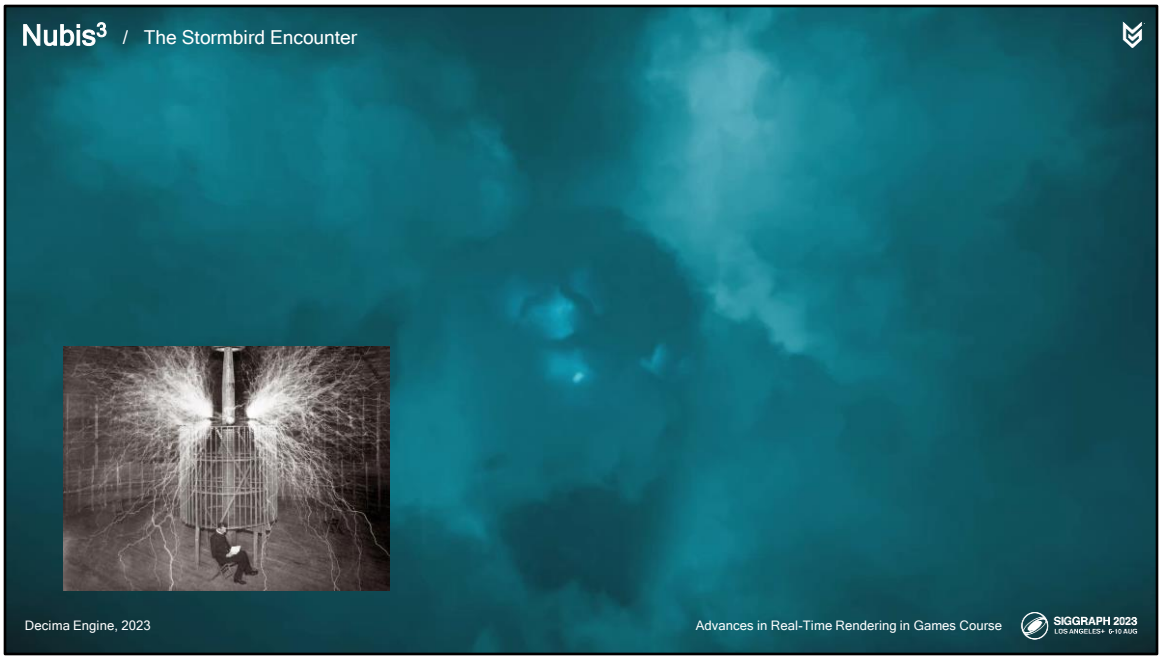
So I made sure that whenever there was a ground discharge of lighting, that the interior would light up and reveal some structure inside. I also added some other Internal flashes to suggest a buildup of some kind.



We used our local weather system from Horizon Forbidden West to create a localized rainstorm under the cloud.

Here’s the finished result of the exterior as seen from a little farther away.
Inside the cloud, I wanted the player to encounter something unexpected, not just a cave as we had modeled in other clouds, but something bizarre for a

human and yet cozy for a stormbird. The player should be able to look at what's going on in here and think, Oh, this is weird and terrifying but functional.



It made sense that there would be some focal point of energy that the stormbird used to keep the unnatural storm going.
And once you kill the stormbird, the focal point and the cloud would dissipate.

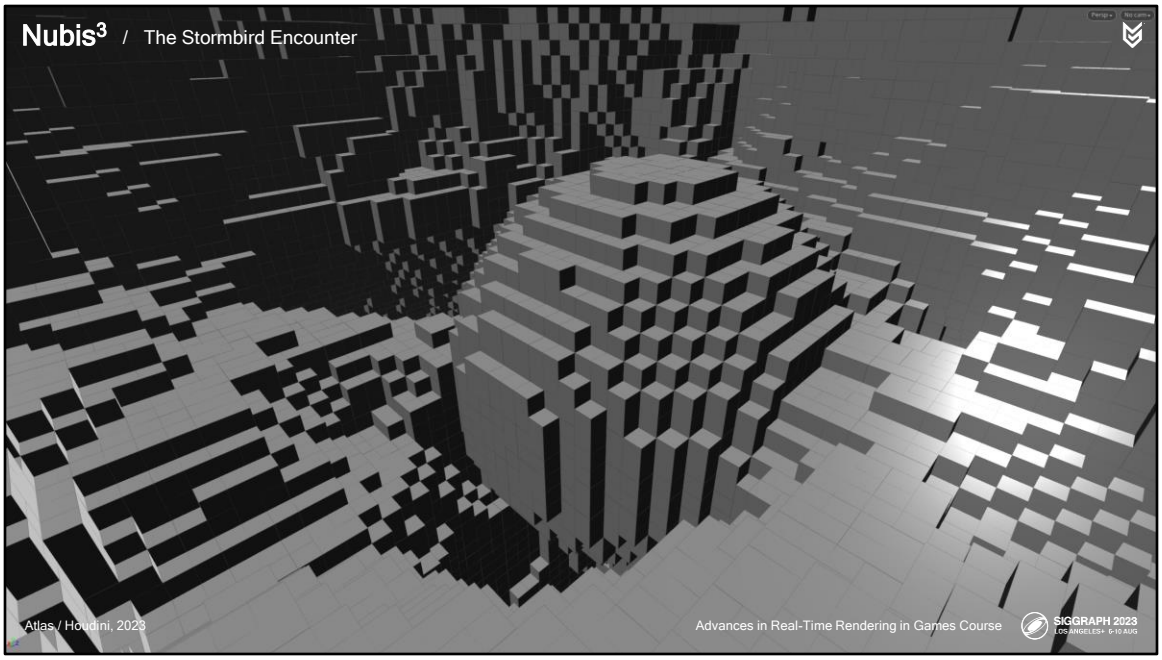
- Meet what I nicknamed the

Egg.

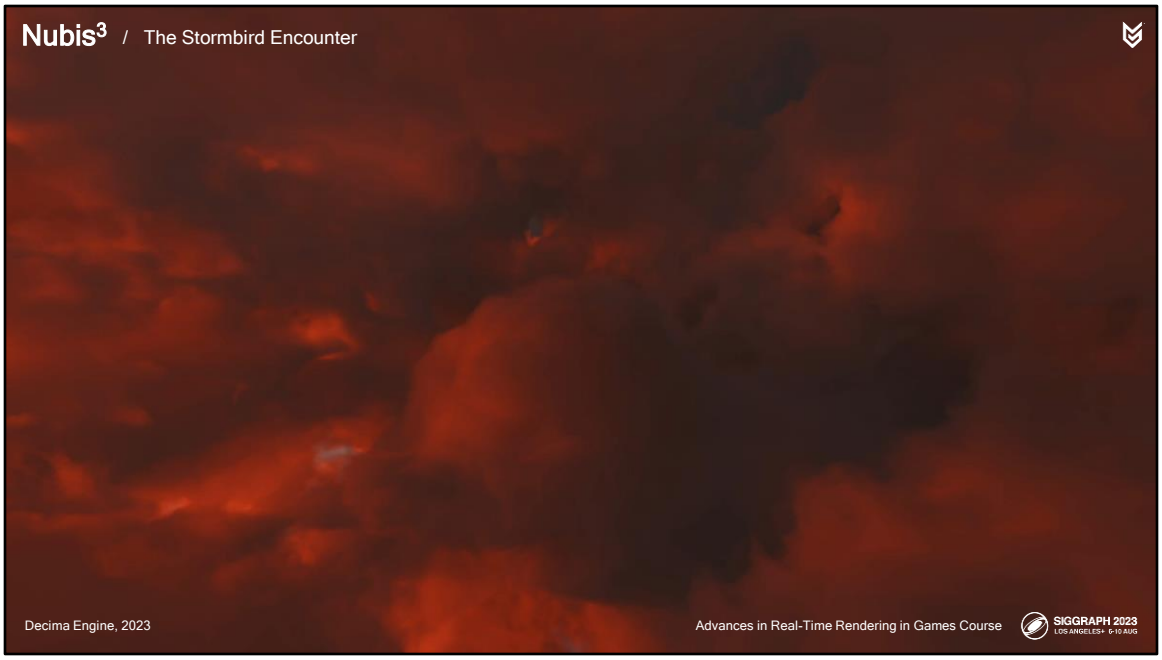
- The little Arcs of electricity were inspired by Nikola Tesla, of course.

Eventually enough charge builds up until we see a ground discharge through a hole under the “egg”, giving it some apparent function.

In terms of implementation, This is another place where we took the lighting tech from the Superstorm System and adapted it for voxel clouds.

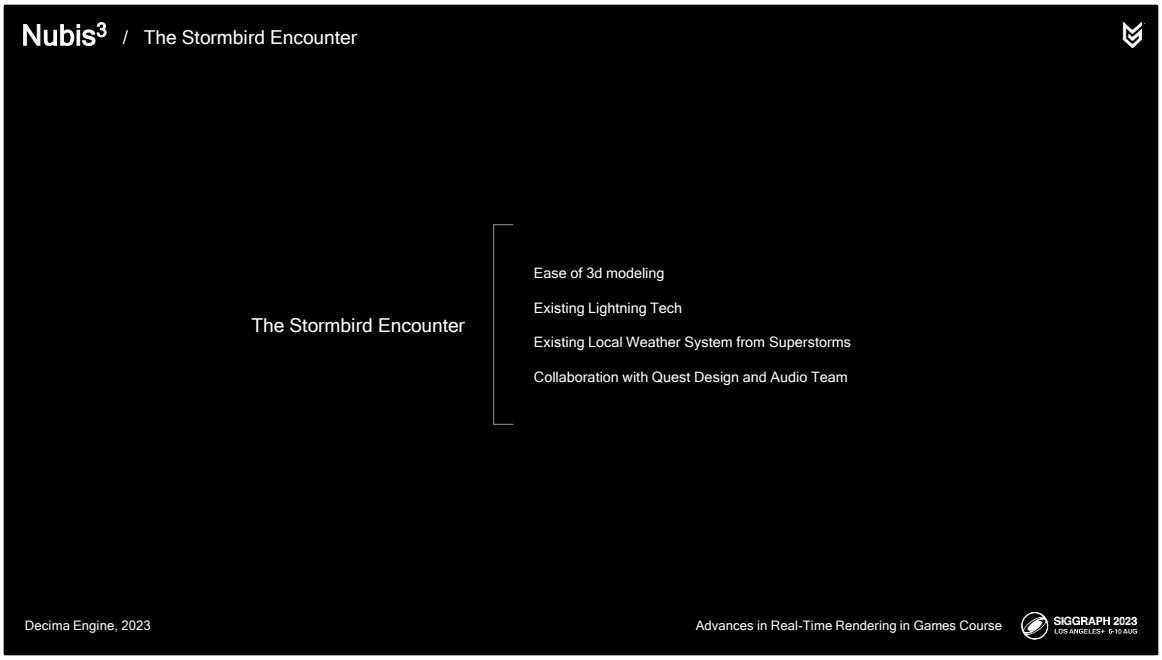


As for the internal space, I used our Atlas tools to cut out the cave and create the egg. I made sure to add plenty of pockets that the stormbird could hide in and launch sneak attacks from.



I also added enough openings so that light could enter and make the space appear different for every time of day – this is an open world game after all.
With all of the pieces in place, Elijah implemented the encounter and Audio provided some really

awesome sound effects.



So because we adopted this new approach to volumetric cloud rendering and because we started to look at clouds as physical things that you can touch and build environments with in the world, and because we could link all of what you saw

to game scripts, we were able to actually do more with other teams and offer a truly unique experience to players on top of just flying through clouds for fun. Its exciting because it feels like this is just the beginning for this kind of gameplay.

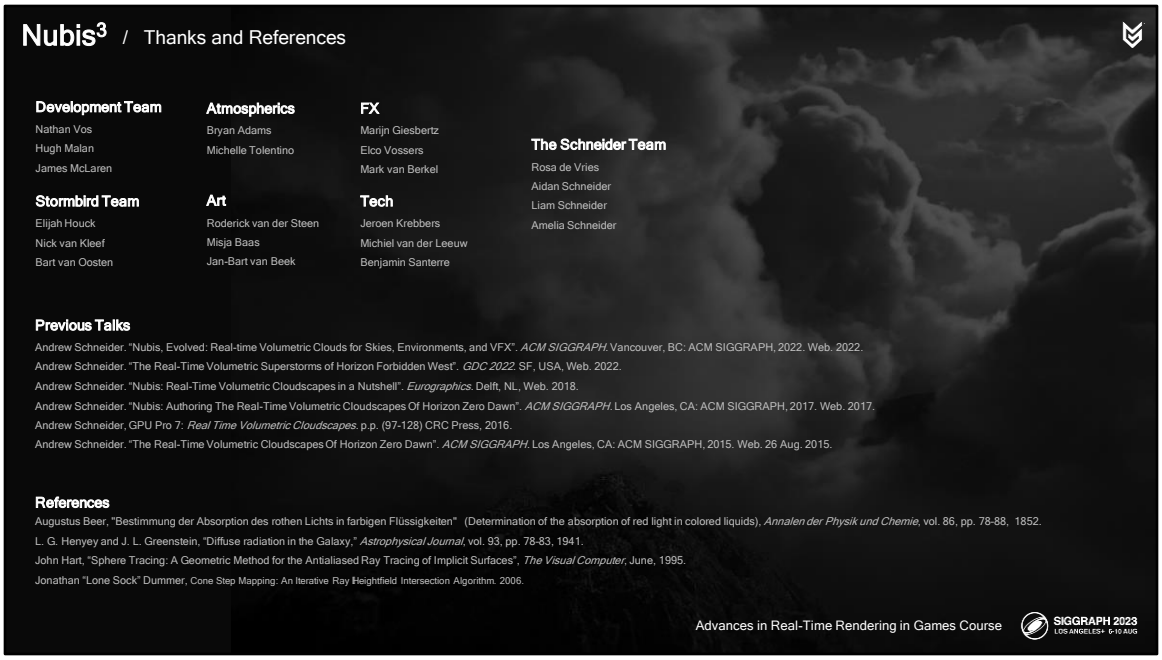
So lets add two rows to our chart. Not only can Voxel-based Clouds be used for quest design, but there is potential for growth here in so many new ways.

Since we are feeling celebratory, We have a DLC of our own design for you. We have packaged up two of our voxel cloud NVDF's in the form of TGA Slices and VDB files along with TGA's and a VDB of the 3d noise we used in the game to up-rez

our clouds. We also included the Generator for this noise in the form of a Houdini Digital Asset. Besides this you will find several bonus slides in the presentation when it goes online that include actual code used in production as well as further explanation of some subjects.
I will tweet or X or whatever it’s called today these links in a few minutes if you don’t catch them now.

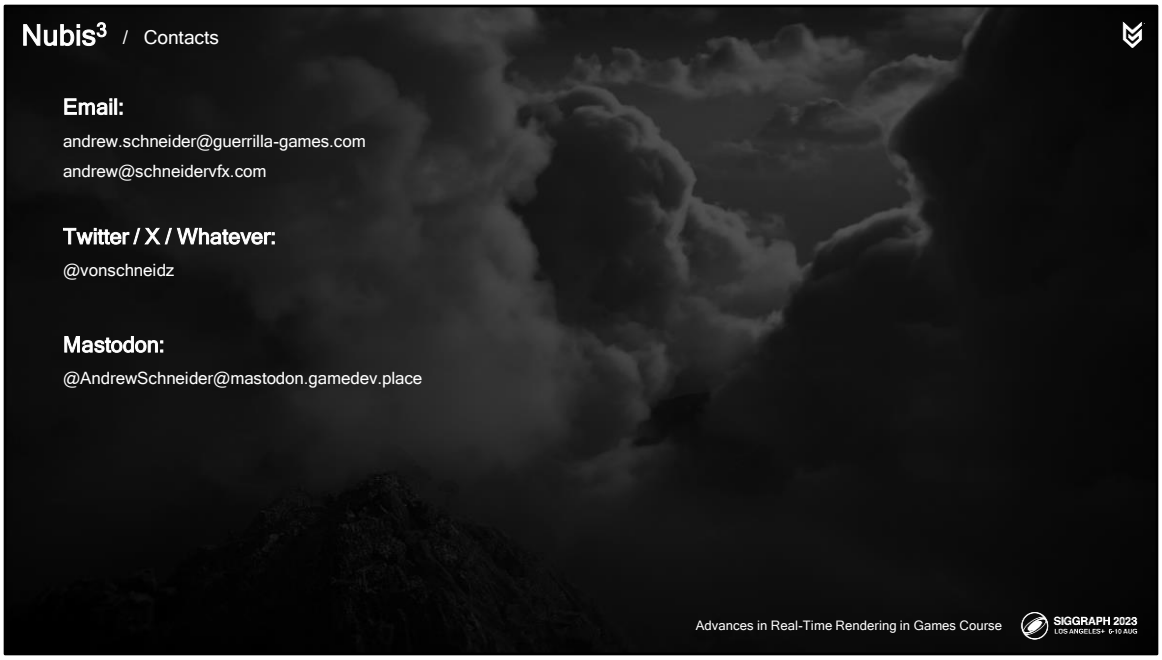
Immersive Voxel-based clouds can be done in real-time open world games. We shipped a game with this technology and while our methods and madness may not be what you had in mind originally, our approach allows us - and hopefully you - to start

working with performant voxel-based clouds now.



I want to thank the development team, who I mentioned during this talk. It takes a team to do something like this.
I also want to thank my team, Atmospherics, for putting up with me while I prepared this presentation. Especially Bryan Adams who captured a lot of the videos that you saw today.
Also, The FX department, The Stormbird team, Art direction and The tech team.
Most of all though, I want to thank my family, Rosa, Aidan, Liam and Amelia.

Thank you.


Nubis3 / Contacts
Email:
andrew.schneider@guerrilla-games.com
andrew@schneidervfx.com
Twitter / X / Whatever:
@vonschneidz
Mastodon:
@AndrewSchneider@mastodon.gamedev.place
Advances in Real-Time Rendering in Games Course
SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

Questions?