

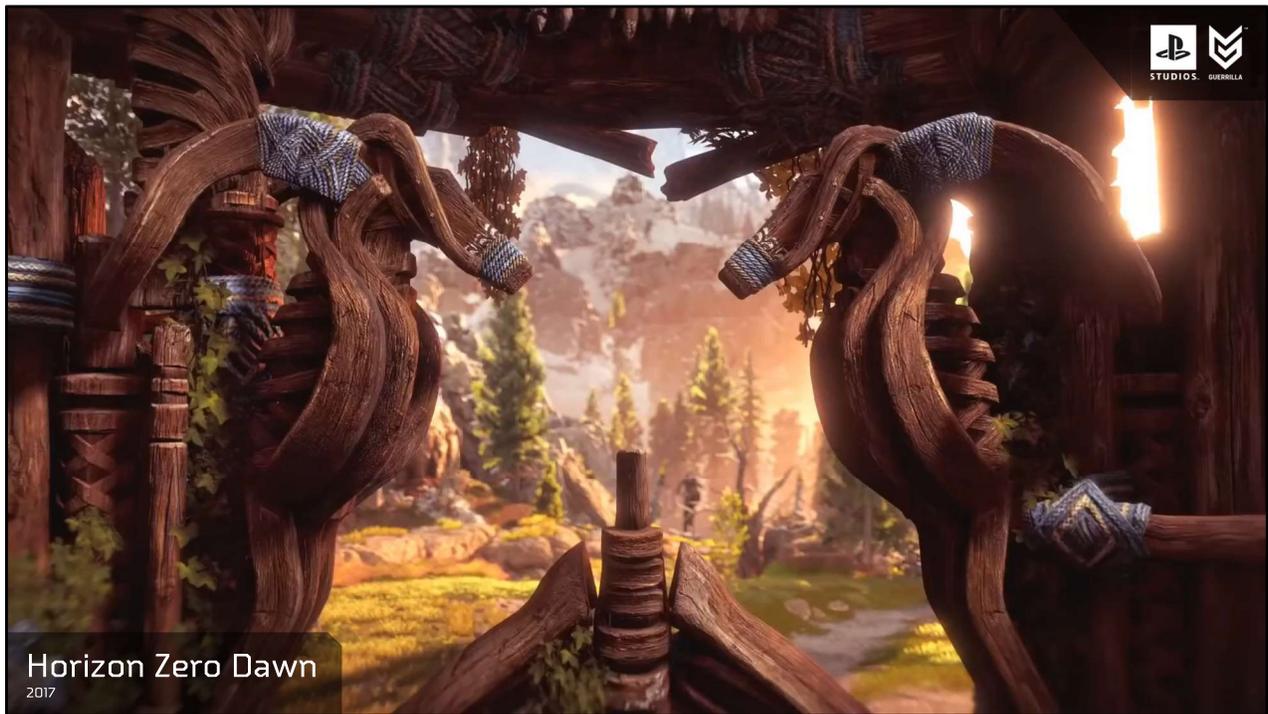


Welcome to the PDF Slides. It is recommended to view the Video of this presentation in the GDC Vault OR download the original Powerpoint presentation from the <http://www.guerrilla-games.com> website in order to get the full experience including videos and animations.

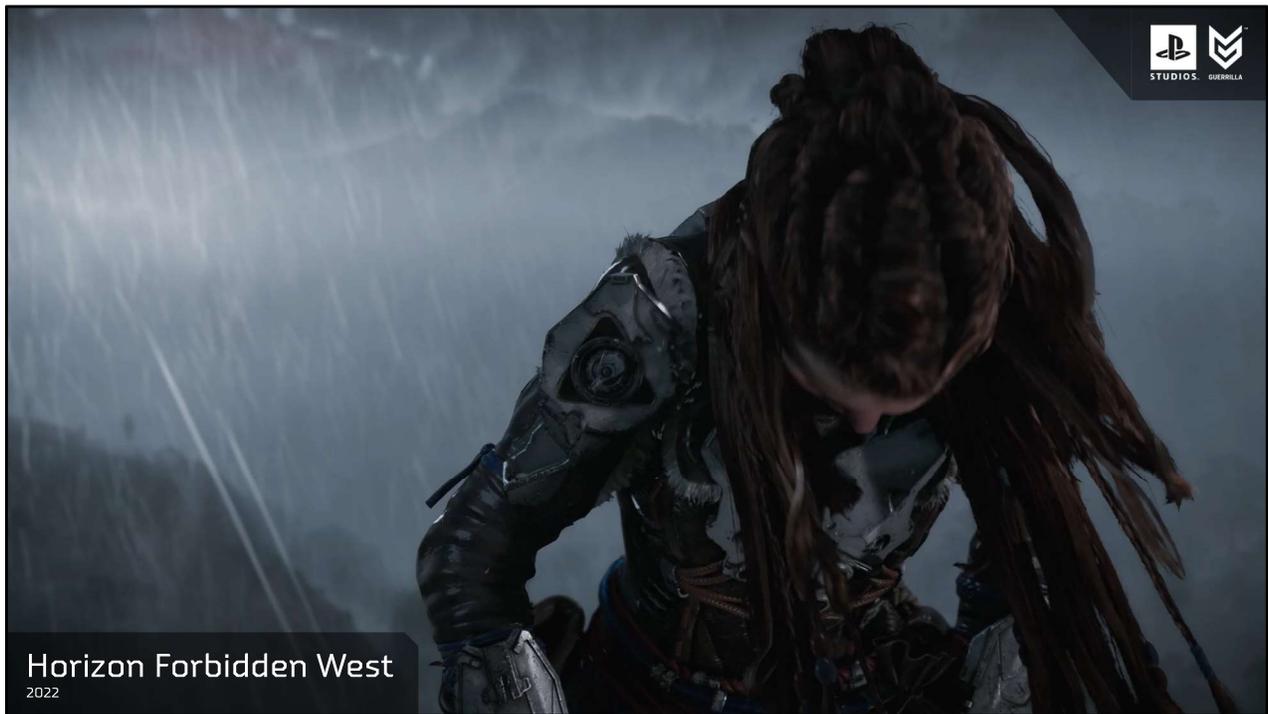
All of the spoken content is in the slide notes. Bullet points indicate a button press for animations.



Hello, I'm Andrew Schneider, a Principal VFX Artist at Guerrilla, a Playstation Studio. I work with the Decima Engine and develop the volumetric cloud systems used in the Horizon Franchise games. Before this I worked for Blue Sky Animation studios on movies like Ice Age and Rio developing the volumetrics pipeline from simulation to rendering for things like VFX and.... yes.. Clouds. I started at Guerrilla in 2013.



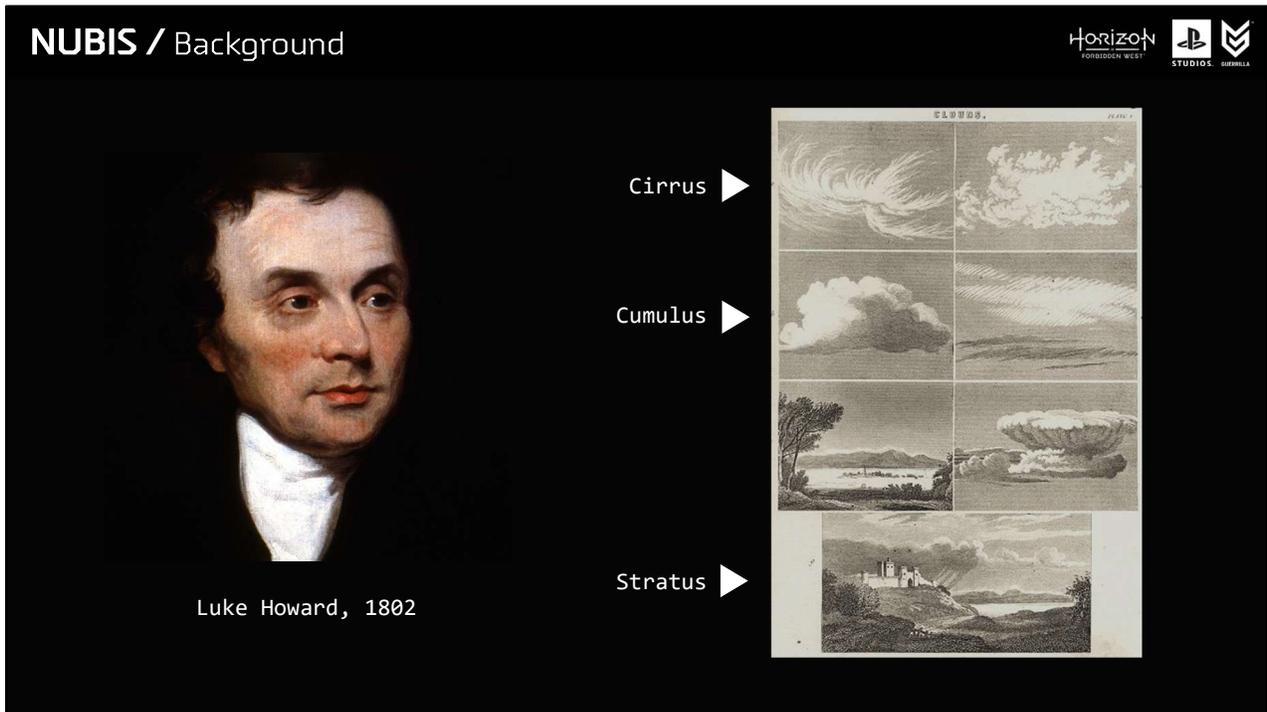
In 2017, we developed a cloud system for Horizon: Zero Dawn, an open world RPG featuring a red haired, bow wielding heroine named Aloy in a Future earth where dinosaur sized robots roam the landscape. The Nubis Cloud system featured real time volumetric clouds that evolved and supported a day-night cycle. We detailed this work twice at SIGGRAPH, Eurographics and in a chapter of GPU Pro 7. Since the 2015 SIGGRAPH course several Triple A studios and game development companies have developed similar systems for use in their games or products. For Zero Dawn, the Nubis system was designed to support a stable world.



For the sequel, Horizon Forbidden west, the world is anything but stable as it begins to disintegrate around Aloy. Ripples of instability generated from the malfunctioning climate control systems begin to overlap and converge into swirling vortices of destructive energy. This energy discharges in the form of red lightning.



For the next hour, I will describe how we engineered this effect by expanding the Nubis cloud system to create The Realtime Volumetric Superstorms of Horizon Forbidden West. By the end of this talk, you should have an example of how systems similar to Nubis could be expanded to support localized storms. In order to explain how we expanded our cloud system, we should first take a look at how the Nubis cloud system was originally designed for Horizon Zero Dawn to render skies full of clouds in real-time. Early on in development, I was inspired by the work of Luke Howard.



In an 1802 lecture called “On the modifications of clouds, he classified clouds by their altitude, basic physical characteristics and the conditions that gave rise to their development using latin nouns and adjectives.

- Here are some Renderings that Howard made to illustrate his points.

Howard dubbed the

- stacked round clouds Cumulus, which means a heap or pile in Latin.
- The Long flat clouds became Stratus, which means layer or sheet in latin.
- The wispy stretched clouds that exist high in the atmosphere became Cirrus, which means hair or fibre in latin.

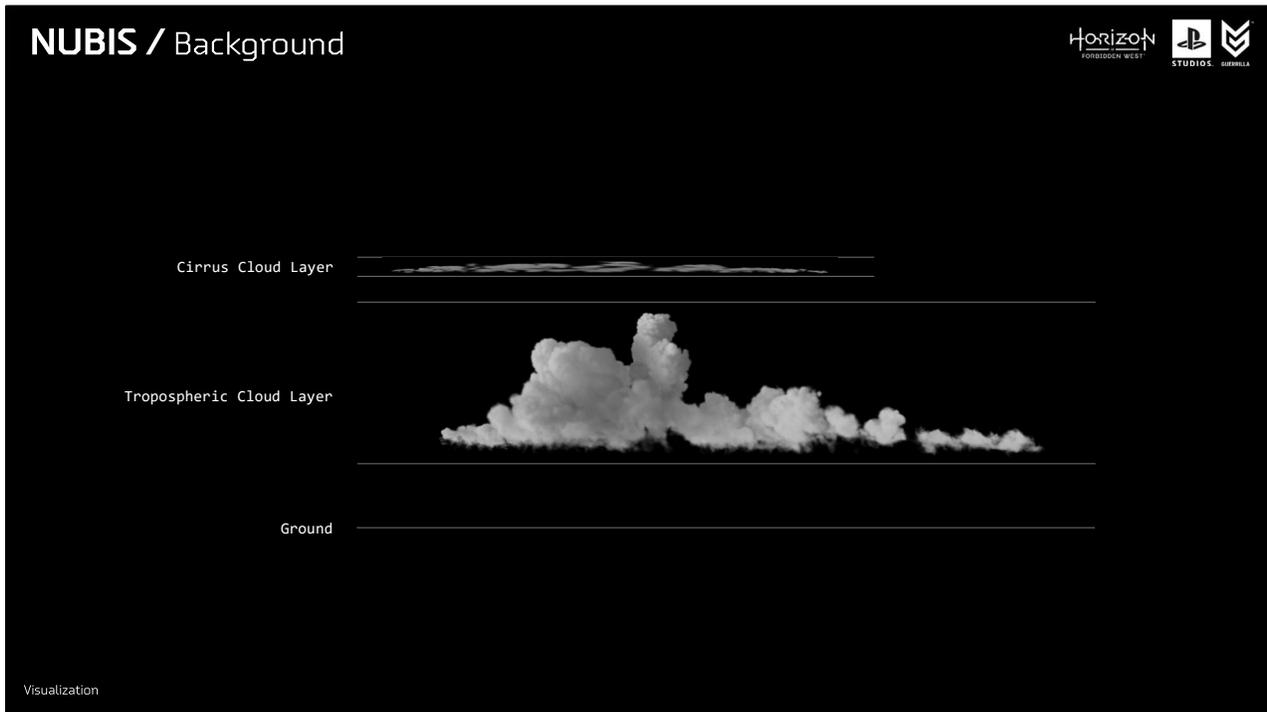
You might recognize some of these names and that’s because Howard was the person who came up with the first classification system that stuck. Howard understood that the amount of vapor, temperature changes over altitude, wind and turbulence all play a role in modeling clouds. Vapor rises and enters cold air, which causes it to condense on dust particles rendering it visible to us. He called this process Nubification, a term which never really caught on but we decided to honor by naming our cloud system after it.



Visualization

- Low emission of vapor into cool air produces the bandy stratus clouds that flow over the landscape and are generally more wispy.
- High emission corresponds to round billowing cumulus clouds that grow higher and higher until they flatten out on another layer of colder air.

We see this type of behavior play out in multiple layers of the atmosphere because there are multiple layers of air with differing characteristics that vapor can travel through.

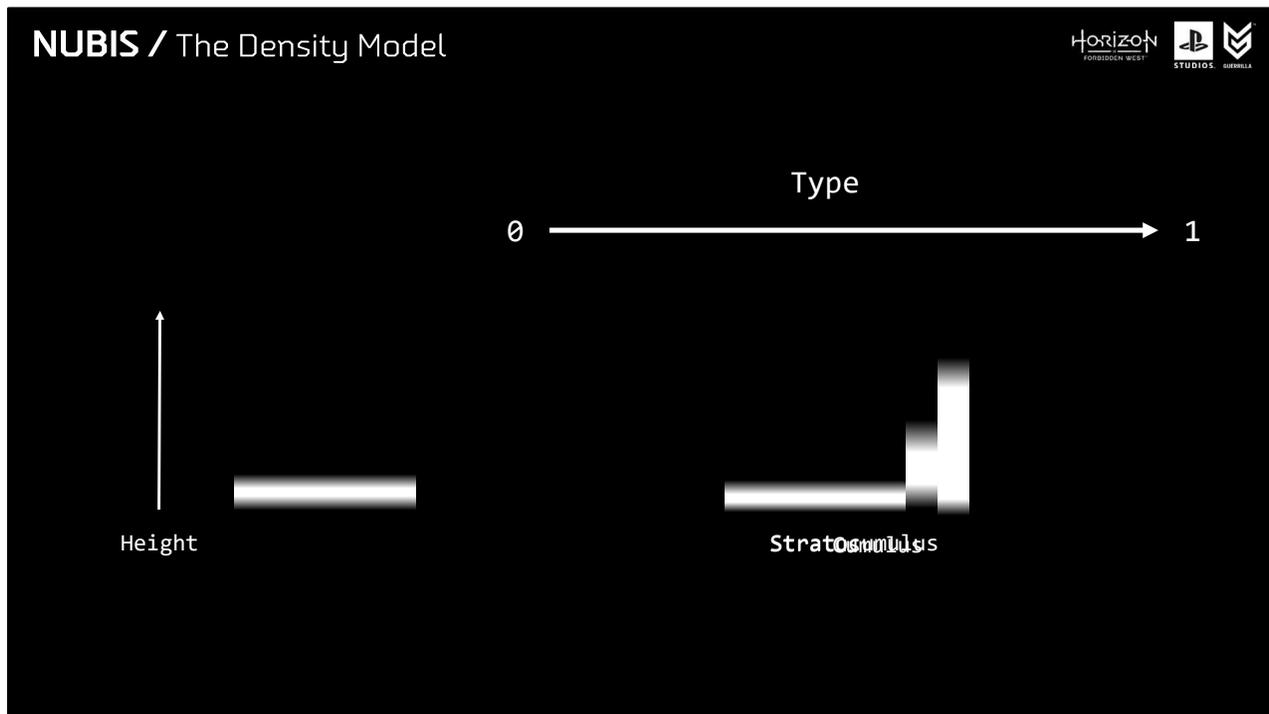


The original Nubis cloud system rendered clouds in two of these layers. The tropospheric low altitude clouds and the high altitude cirrus clouds. Let's take a look at how we modeled clouds in these layers. And let's do it in terms of finding the density of cloud material for a given point in the sky.

Cloud Coverage

Cloud Type

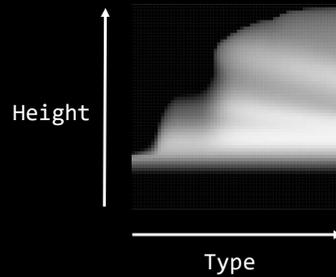
In the 2015 Advances in Real-time Rendering SIGGRAPH Course we defined two probabilities that tell us what type and how much of a cloud to draw at a given point. Cloud Coverage describes the presence of clouds. Cloud type describes the height of the cloud and whether it is billowy or wispy.



If we were to describe the probability of change in density over height

- for a cloud we might envision a gradient like this.
- Since we know that cloud type is determined by rate of emission and temperature, and that each cloud type has a different height range
- we can order a set of these gradients according to cloud type in order to represent the height probabilities for each type of cloud. Type, can then serve as the value which blends between these three height functions.

The Vertical Profile Gradient



When we combine these into one lookup, we accelerate and decelerate some of the transitions between types and add some areas with varied intensity to create a bit more variety.

We call this lookup the “Vertical profile Gradient”  
Lets look at this in practice.



Here are some clouds that use the cumulus vertical profile gradient.

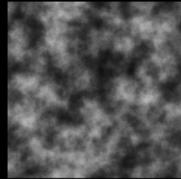
- And Stratocumulus
- And now stratus.

Now let's look at how we define the details that you see in this image.

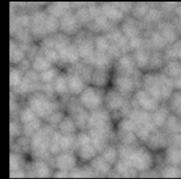


The detailed shapes on clouds can be classified into two categories: billows and wisps.

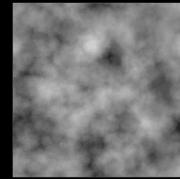
- Billows form when density increases in a given area and pushes water vapor outward and upward
- Wisps form when density decreases and vapor dissipates or curls around the wake created by turbulence forces.



Perlin



1-Worley



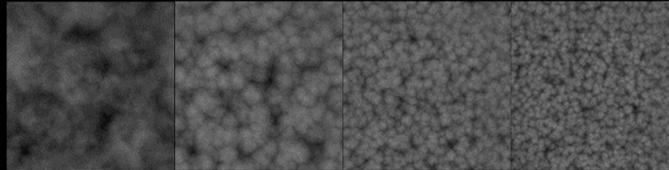
'Perlin-Worley'

If we were to describe the curling and roiling shapes over 3 dimensional space, we might use 3d textures like

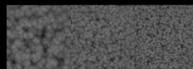
- Perlin noise,
- inverted Worley noise or, as we proposed in the 2015 SIGGRAPH Advances in Real-time Rendering Course,
- a combination of both depending on cloud type or height.

Our approach to creating perlin-worley noise was to subtract the web like shapes of Worley noise from the high density regions of perlin noise in order to introduce round shapes there. This way we get the best of both worlds while only having to take one sample of a 3d texture, something that is very expensive.

LF Noise [ 128<sup>3</sup> ]



HF Noise [ 32<sup>3</sup> ]



```
base_cloud_density = remap(lf_noise, 1.0 - hf_noise, 1.0, 0.0, 1.0)
```

As detailed in in the 2015 course, we define a base cloud density as

- low frequency Perlin Worley and Worley noise an then
- erode that shape using high frequency Worley noises to add detail. Using two noise samples allows us to only do the high frequency noise operations where needed, thus improving performance.
- The erosion is done using a remapping function. What we call a remapping function is just something that takes a number in a given range and repositions it into a new range. This is the same thing as Maya's SetRange and Houdini's FitRange. We decided to use remapping functions when constructing our base cloud shape because of a really useful behavior: as opposed to multiplying the noises together, Remapping prevents a loss of too much density at the core of the base cloud shape. I'll give an example...



Here's what our clouds look like WITHOUT using noise as a base probability for density.

- Our Low frequency noises really form the foundation of our clouds, as you can see.
- And the high frequency noises add detail without taking anything away at the center of the cloud.

Its kind of like carving out a block of clay, except that all of the details are already stored in the clay and you are just revealing them as you carve. But in that analogy what dictates how deep to carve?



```
cloud_density = remap(base_cloud_density * vert_profile_gradient, 1.0 - cloud_coverage, 1.0, 0.0, 1.0)
```

Large scale variations in emission of moisture are what create cloud formations.

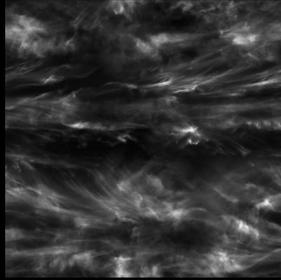
- We can express this by defining a cloud by its coverage of the sky at a given point.
- Also, because we are using our noise as a base probability for cloud density in a sample,
- we can apply the coverage value as an erosion to the product of our height gradient and noise. This allows the cloud appear to expand or contract over a gradient of coverage values in space.

Here is an example.

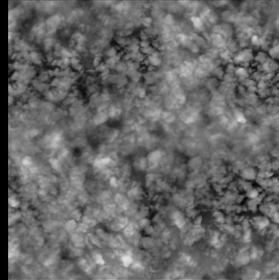


As the coverage value increases across the sky, the clouds in this image begin to inflate. This is useful when animating cloud coverage in transitions.

Cirrostratus



Cirrocumulus



The solution for cirrus clouds was much simpler.

- We compiled a texture containing 3 tiling cloudsapes That cover cloud types from the Whispy Cirrostratus to the Billowy Cirrocumulus
- And blended between each depending on the type of cloud being sampled. The inverse of coverage was used to erode the cirrus cloud sample density.



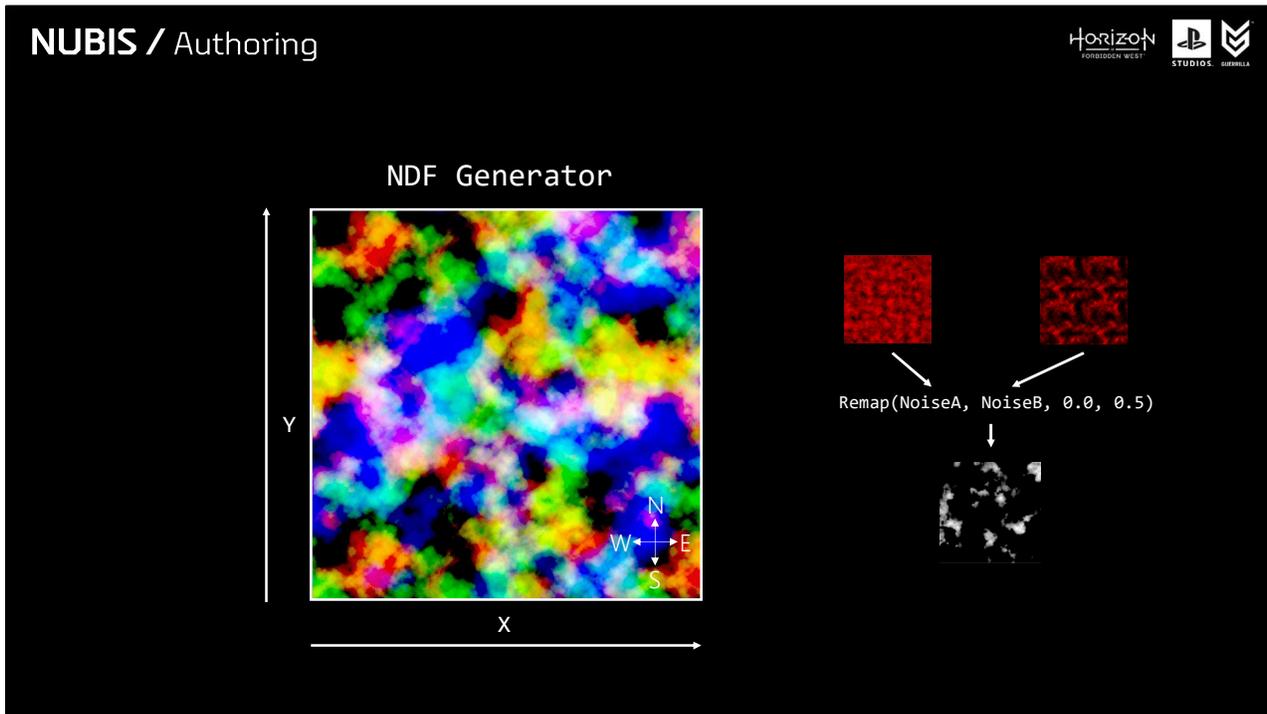
Heres an example of Cirrostratus

- Something in between
- Cirrocumulus



We can simulate the movement of clouds across their regions of type and coverage probability by

- adding an offset in the wind direction which is incremented in time and subtracted from the sample position of our 3d noises. Here is a sped up example. Notice that the general shapes do not change, but the details do. This is because while the noise in the density model is animated, the coverage and type probability fields are not. When we are constructing art-directed clouds as opposed to a simulated sky, we only want them to appear to be changing without altering the larger structure of the cloudscape.

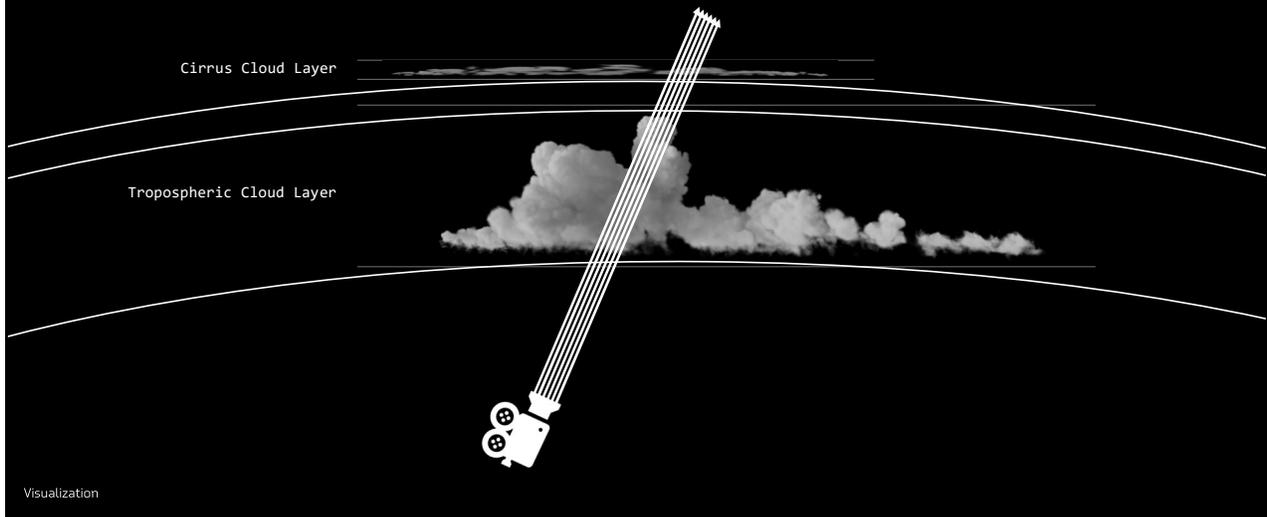


That's how we determine the density of cloud material for a given point in space. But in order to fill a sky with clouds we needed another system to define a field of cloud coverage and type data over the xy domain of the world so that it can be sampled in the Nubis renderer.

- That system is called the Nubis Data Field generator. Using a set of noises, remap functions and composites, we can produce a variety of different cloudsapes.



- Here are a few examples of what we were able
- to do for the first game, Horizon Zero Dawn.



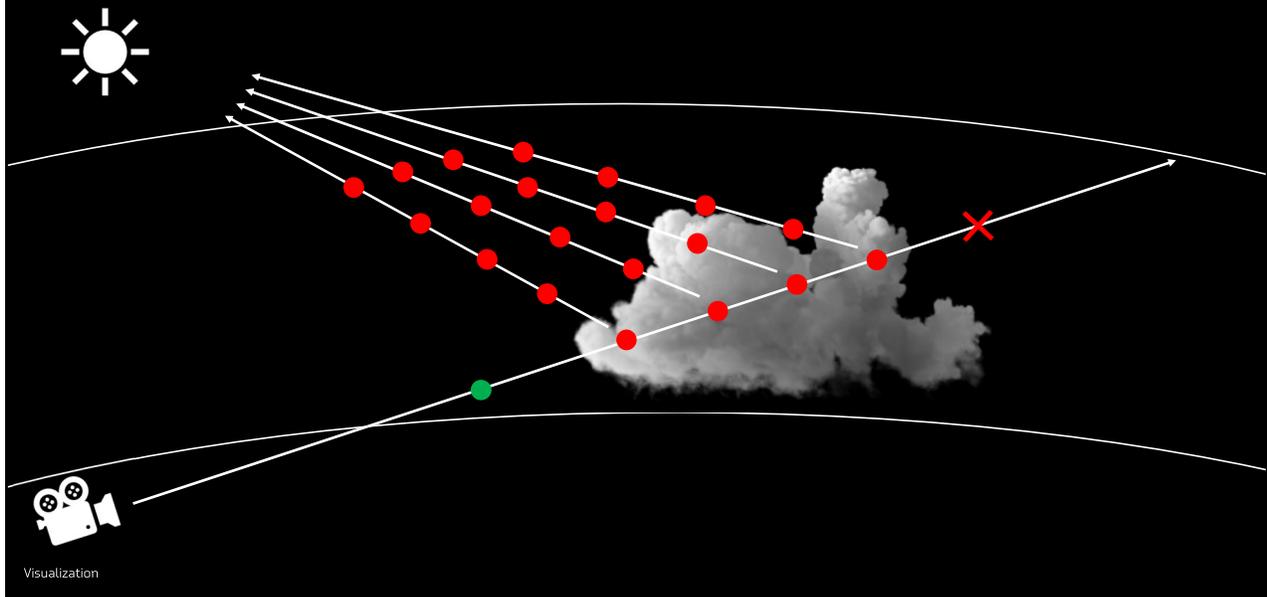
Now that we have covered modeling and authoring, we can take a look at how we actually start to construct renders using ray-marching.

- Ray-marching is one Rendering technique that can be used whenever you need to construct an image that contains volumetric objects. The idea is to construct density and lighting samples at several points along the view vector and combine them in a way that produces Color and Opacity data for each pixel.
- This is done for every pixel to produce a full render. To ray-march volumetrics like clouds efficiently you need a good idea of where to begin and end your march.
- Since the atmosphere of the earth is a spherical shell, we can define our ray-march ray segment using the intersections on the inside and outside of this shell. For the Cirrus cloud layer we only needed one intersection as it is a 2-dimensional layer.



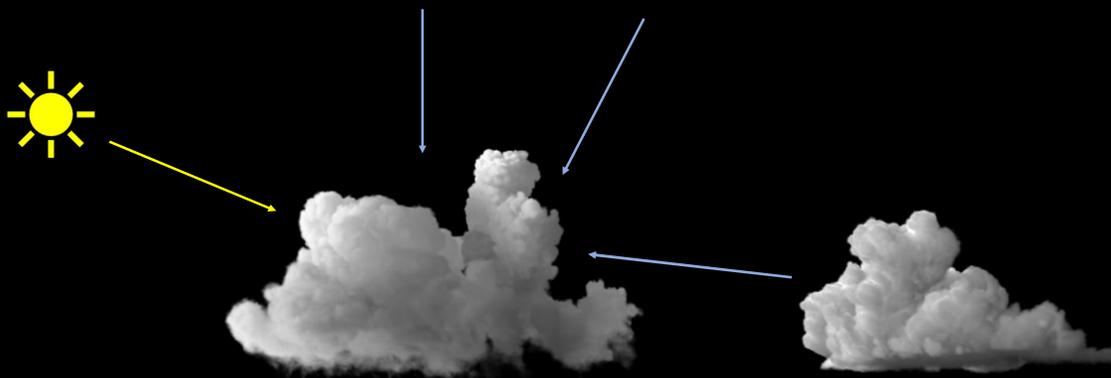
But before any of that, it's a good idea to know where you don't need to construct any samples at all so that you can skip ray-marching those pixels.

- if objects like trees and mountains render front of the clouds,
- we do not march from those pixels.
- This reduces the total
- amount of work that needs to be done.



For the pixels that do have visible sky in them,

- we march over the view ray to gather the density and lighting data needed to construct color and opacity for the pixel.
- We start with large steps and at each one we construct what we call a coarse sample. That is a sample built with only the cloud coverage and vertical profile gradient to keep the operation as cheap as possible. We construct and check these coarse samples until
- we one of them returns a result which is nonzero.
- Then we switch to taking smaller steps and constructing what we call Fine Samples which include the 3d noise texture samples needed to erode the base shape with cloud details.
- We do this until the total density sampled along the view ray sums to full opacity. Then we stop the march. To calculate cloud color, we need light intensity information
- For every fine sample, we also march toward the light along the light ray, taking fine samples in order to gather the amount of density between the sample and the light source for use in light intensity calculations.



$$\text{Light Energy} = \text{Direct Scattering} + \text{Ambient Scattering}$$

Visualization

Cloud lighting can be a complicated subject. There are a lot of things going on when a photon enters a cloud and it can all be very computationally expensive. So, we decided to focus on the characteristics of cloud lighting that were most important to us and model less expensive approximation methods for them that we combine to describe the light energy of a sample.

We define light energy as the sum of two components.

- Direct Scattering and Ambient Scattering.
- Direct scattering is all of the light energy associated with the strongest light source, the Sun.
- Ambient scattering is all of the light energy associated with contribution from other bright sources like the sky and neighboring clouds. Lets look at Direct scattering first.

`Direct Scattering` = Transmittance \* Scattering Phase \* In-Scatter Probability

We define Direct scattering at a given sample point in a cloud as a function of three probability functions:

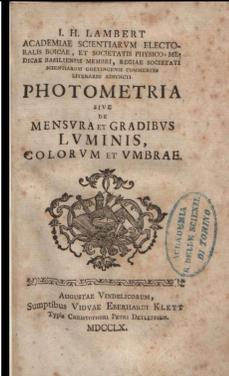
- Transmittance, phase, in-scattering probability. Let's break these down starting with transmittance

$$T = e^{-d}$$

78  
III. Bestimmung der Absorption des rothen Lichts in farbigen Flüssigkeiten; von Beer in Bonn.

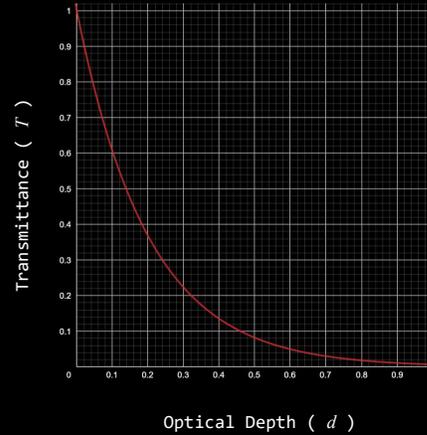
Ohnals schon ist die Absorption des Lichtes beim Durchstrahlen gefärbter Substanzen zum Gegenstande des Versuchs gemacht worden; man richtet hierbei jedoch immer nur die Augenmerk auf die relative Schwächung der verschiedenen Farben oder, bei kristallinisierten Körpern, auf die Beziehung zwischen der Absorption und der Polarisation-Richtung. Ueber die absolute Größe der Absorption, welche irgend ein bestimmter Lichtstrahl bei der Fortpflanzung in einem adäquaten Mittel erleidet, liegt meines Wissens Nichts vor. Nur auf Rückstände hierauf stütze ich in diesem Aufsätze eine Reihe von Maßbestimmungen der absorbirenden Kraft an. Anderserseits nämlich entspricht mir die Unvollständigkeit meiner Bestimmungen keineswegs. Sie beziehen sich nur auf rothes Licht, wie es von einem dunkelrothen Glase geliefert wird. Wünschenswerth aber wäre es, jedwede die Absorption von wenigstens allen Hauptfarben des Spectrums zu erhalten. Hieda kann jedoch nur mit viel complicirteren Einrichtungen erreicht werden, als mir zu Gebote stehen. Ein Gleiches ist zu bemerken in Betreff des Uebersandes, da ich nicht mit Sonnenlichte, sondern mit Lampenlichte operire. Ich sah mich deshalb genöthigt, meist nur geringe Dicken oder verdünnte Lösungen der farbenden Salze (denn auf solche habe ich mich hauptsächlich) dem Versuche zu unterwerfen; hiernach wird aber der Werth der numerischen Ergebnisse in sofern vermindert, als am Orte nicht mit Sicherheit aufwärts auf die Absorption in concentrirten Lösungen oder bei größeren Dicken, sondern nur abwärts gefolgert werden darf.

Die mittheilenden Messungen wurden mit Hilfe eines Photometers angestellt, in welchem gewissermaßen das Princip des Ritchie'schen Photometers mit der Arago'schen



Augustus Beer  
(1852)

J.H. Lambert  
(1760)

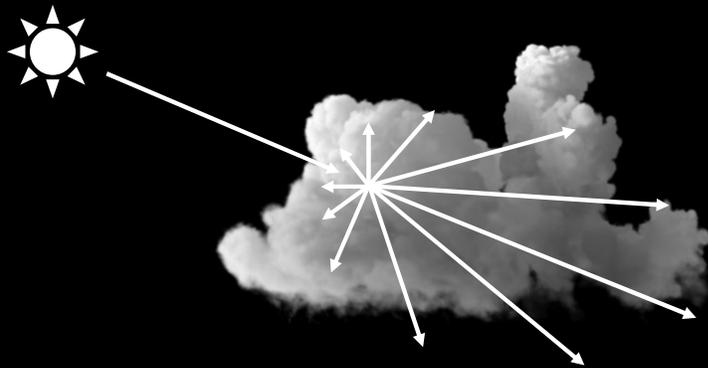


Back in 1852, a chemist and mathematician named Augustus Beer (great name) wanted to find a quick way to measure how much light was being absorbed by opaque molecules in a solution.

- In conjunction with work by Johann Heinrich Lambert from nearly a century earlier, He came up with a way to express the reduction in light transmittance
- as a function of optical depth in a given medium. In other words if you are a photon the deeper you are in a translucent object, the lower the probability that you will escape and reach someone's eyes in a behavior known as In-Scattering. Since clouds are a translucent medium with absorptive properties, beer's law is a reasonable approximation. You simply sum up density samples along the light ray and plug it into the equation as the optical depth.



Here are some examples. On the left are some clouds with the beer-lambert function being used as the attenuation component. The right shows the same cloudscape without the attenuation component.



```
float HenyeyGreenstein(float inCosAngle, float inG)
{
    float num = 1.0 - inG * inG;
    float denom = 1.0 + inG * inG - 2.0 * inG * inCosAngle;
    float rsqrt_denom = rsqrt(denom);
    return num * rsqrt_denom * rsqrt_denom * rsqrt_denom * (1.0 / (4.0 * M_PI));
}
```

Visualization

The second component deals with light scattering directionality.

- In clouds, light scatters in a forward direction, meaning that there is a higher probability that you will see scattered light when you look toward the sun, through a cloud. In 1941, Henyey and Greenstein developed a way to solve light directionality problems at galactic scales. Well, what works for astronomy also works for participating media at earth scales.
- Depending on the material transmitting light and the angle between the viewer and the sun, this function can approximate the intensity of scattered light that the viewer sees. Since clouds are slightly forward scattering we supply an eccentricity of .2



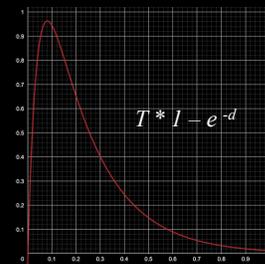
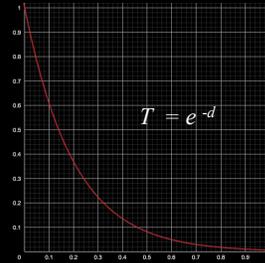
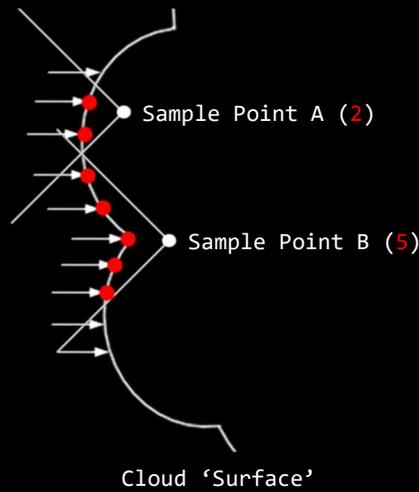
We actually combined several of these functions with varying

- eccentricities in order to get a more art directed silver lining effect that we could control at sunset.



Photograph

The third component is a function developed by us to model an oft-forgotten lighting effect in clouds. Those dark-edges as that you see when you look at clouds with the sun behind you. Well, recall that photons enter the cloud and then scatter in many directions when they hit clumps of water molecules and some of these photons escape to your eye in a behavior known as in-scattering. Instead of seeing a white cloud with dark edges, first, imagine it as a black cloud with no light scattering happening on the edges, but deeper in the cloud, in-scattering is occurring more frequently, which shines through like a flashlight in a cotton sheet. When you look at this effect geometrically as the result of in-scattering potential it is much easier to model.



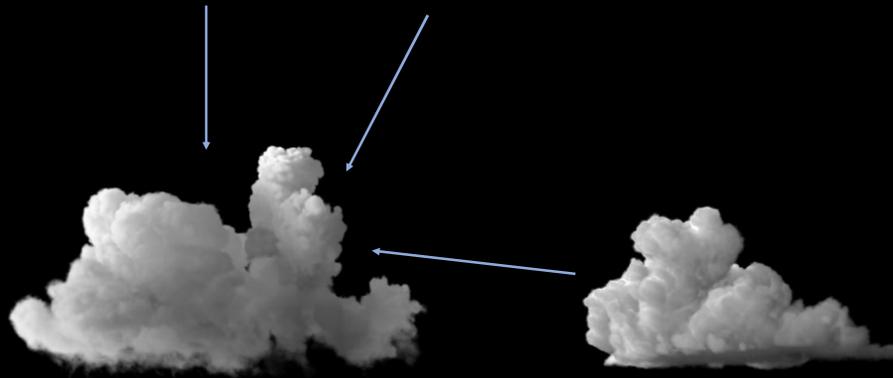
Take two points at the same depth in the cloud. One is behind a convex surface and the other is behind a concave surface. The one behind the concave surface will have more in-scattering contributors between it and the sun. How many more? Its better to ask what's the probability of in-scattering occurring in these points?

- It can be described as the inverse of the beer-lambert-law. We call this component the In-Scattering Probability Function.



This is what a cloudscape looks like with only the Absorption and Phase components.

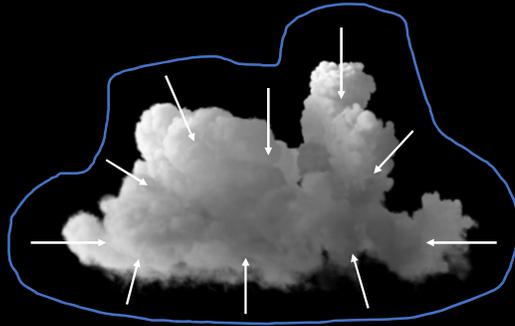
- And this is what it looks like when we apply our in-scattering probability function. Once you see this effect and understand it, it is quite hard to unsee it in nature.



Visualization

Those are light effects that deal with light from the sun.

- But the sky and indeed neighboring clouds contribute to cloud lighting. We call this Ambient Scattering.



`Ambient Scattering = pow(1.0 - coarse_density, 0.5)`

Visualization

To approximate the effect of light entering the cloud from all around without ray marching to many light sources, we model this geometrically as a probability field. The majority of light coming from above and around and penetrating just the surface of the cloud.

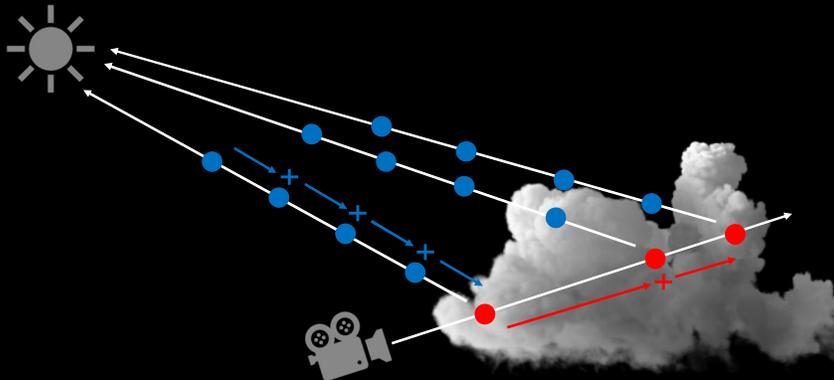
- The coarse density sample already provides a kind of gradient from the outside of the cloud to the inside.
- We use the inverse of the coarse density sample to create a gradient representing the probability that light will reach a given point inside the cloud.



Here is what the ambient component looks like on its own.

- Here is the direct scattering component.
- And here is the combination of the two. It adds quite a bit of dimensionality to the clouds.

## NUBIS / Ray-March Integration



```
// Accumulate light_absorption from sampled density
light_absorption += sampled_density * (1.0 - saturate(light_absorption));

// Accumulate energy and attenuate based on depth in the cloud along the view ray
light_intensity += (light_energy * saturate(sampled_density) * (1.0 - saturate(light_absorption)));

// Accumulate energy and attenuate based on depth in the cloud along the view ray
float3 color = float4(direct_intensity * sun_color + amb_intensity * amb_color);
float alpha = light_absorption;
```

Visualization

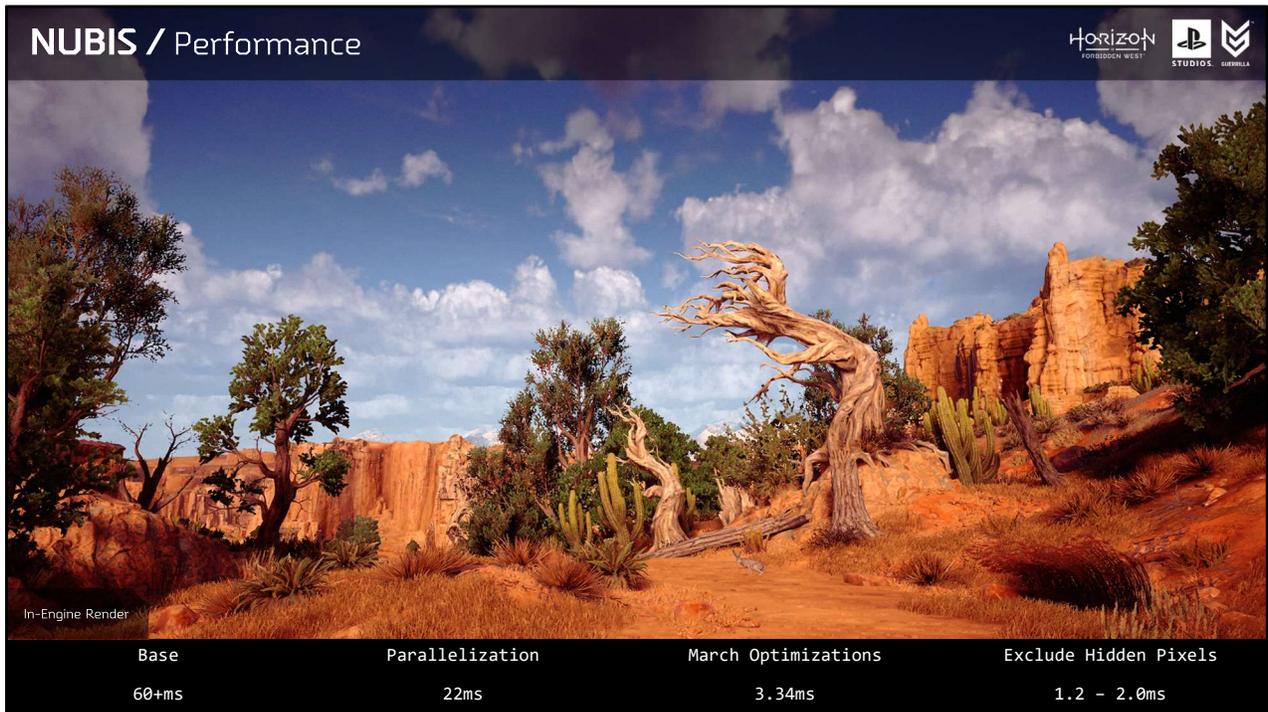
Now that we know how our ray-march works and how to determine the density and light energy at a sample, we can put it all together to describe how we determine color and opacity for a pixel. The samples deeper into the cloud have less influence on the final color and opacity because they are occluded by cloud material. Think back to the Beer-Lambert transmittance function. Once you know your depth, you can use it to attenuate light. However, in this case, we need to reduce the influence of samples along the view ray at each step.

- First we get the current amount of occlusion by density along the ray by calculating the product of the sampled density and the inverse of the current amount of occlusion by the preceding samples.
- Next, we get the current amount of visible light energy by calculating the product of three things: the current light energy, the density of the sample and the inverse of the current amount of occlusion by the preceding samples. This reduces the intensity of the current sample according to how much material is in front of it.
- Finally, light intensity is colored according to Ambient or Sun light sources and output to the color channel while the absorption becomes the alpha channel. This is done for every pixel in the image. In the cirrus layer, there is only one sample per pixel, since it is a 2-d layer.

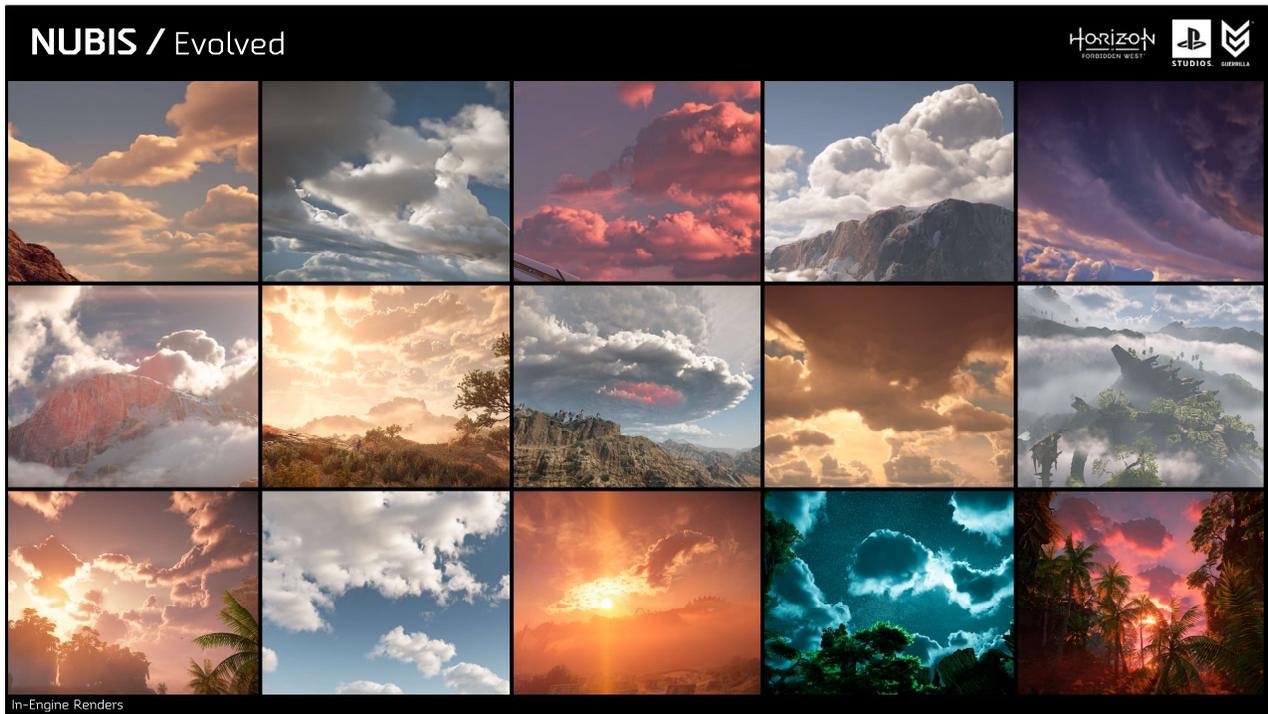


To accelerate our renders, We

- split our render up into blocks of 16 pixels that run in serial and build an image out of these pieces. The blocks themselves run in parallel. This technique is known as Temporal upscaling. The working buffer resolution is 480x270 and we upscale to 1920x1080. In the case of the original Nubis renderer, this had the effect of turning a 16ms render into a 2ms one.

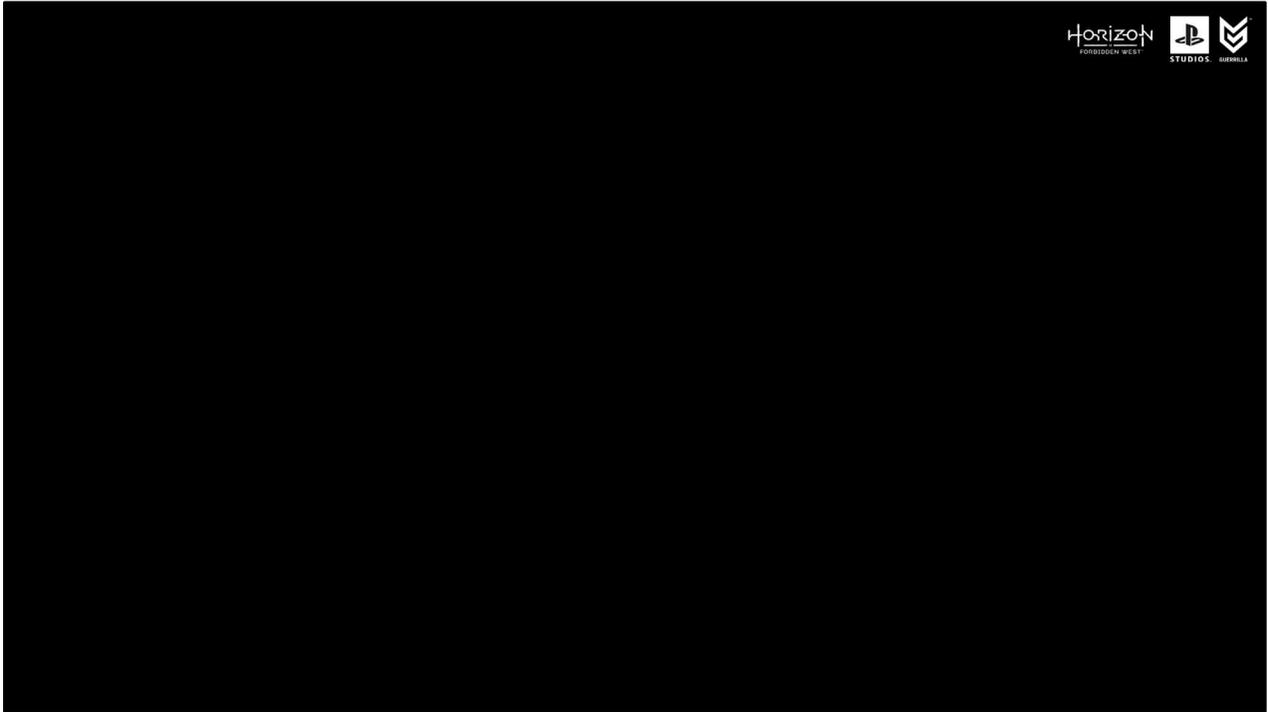


Here's the performance results of the original Nubis system. For this frame, the shader without parallelization or any optimizations costs more than 60 milliseconds. After we apply temporal upscaling it drops to 22ms. Still pretty far from ideal. After the ray march optimizations I just described involving coarse and fine samples, it drops to 3.34ms. And after we don't bother rendering below the horizon or behind large objects, the time drops to between 1.2 to 2ms depending on cloud type and coverage. To us, this is a reasonable cost to pay for something that takes up half of the frame. Anyway, that was the original Nubis system. It did its job well, but after 5 years we can finally share some of the advancements we have made for Horizon Forbidden West.



### The second iteration of Nubis

- includes many visual enhancements for authoring, modeling and lighting the existing 2 cloud layers. We also engineered a new rendering approach for a mountain-intersecting cloud layer that the player can actually fly through. In the near future we will share how we made these advances, but the reason we are here today is to talk about Superstorms. Here are a few of them as they appear in the Forbidden West.



(Play Video)

Realistic  
Powerful  
Ominous  
Dynamic  
Performant



We defined 5 goals for the superstorms.

- Realistic – Often In videogames we have to make sacrifices in realism just to get the big effect we are developing to run on the hardware. Not for this. We would need to emulate the most important details, characteristics and behaviors of large scale storms, otherwise it would serve as a negative distraction in gameplay.
- Powerful – The idea was that the malfunctioning weather system created buildups of destructive energy which manifested these storms and discharged in the form of lightning until the storm dissipated. So, we would need to create lightning and internal glow effects to make it look as though the storm was a powerful and destructive source of energy.
- Ominous – Not only would the storm itself need to look threatening, but the environment around it would need to be affected. Rain, Wind and Sound.
- Dynamic – The storms should form anywhere.
- Performant – it all needed to work on ps5 AND ps4 hardware. With the exception of performance, Realism intersected with all of the other goals.

# SUPERSTORMS



/Goals  
/Of Supercells...  
/Modeling & Animating  
/Modeling Light  
/Lightning Effects  
/Environmental Effects  
/Controlling Chaos  
/Scaling PS4 & PS5  
/Conclusion

In-Engine Render

Obviously, the best way to convince the human eye that it is looking at something real is to supply it with something based on reality. It made sense to model a lot of the superstorm on a natural phenomenon that most people playing our game will have seen before - at least on a tv screen.

- So, First I'm going to talk about Supercells.
- Then I will begin explaining our solution with the topic of modeling and animating density including our solution for fast movement with temporal rendering.
- Next I will talk about modeling light,
- and lightning effects.
- Then I will explain how The storms affected the environment around the player
- After that I will explain how we built a system to control this chaotic weather effect.
- Next I will explain how we made this effect scale between two generations of Playstation hardware.
- And Finally I will close with a summary of what we covered here today.

I contacted some Supercell experts to get permission to share some really awesome work that they did. First, lets look at a supercell in the wild.

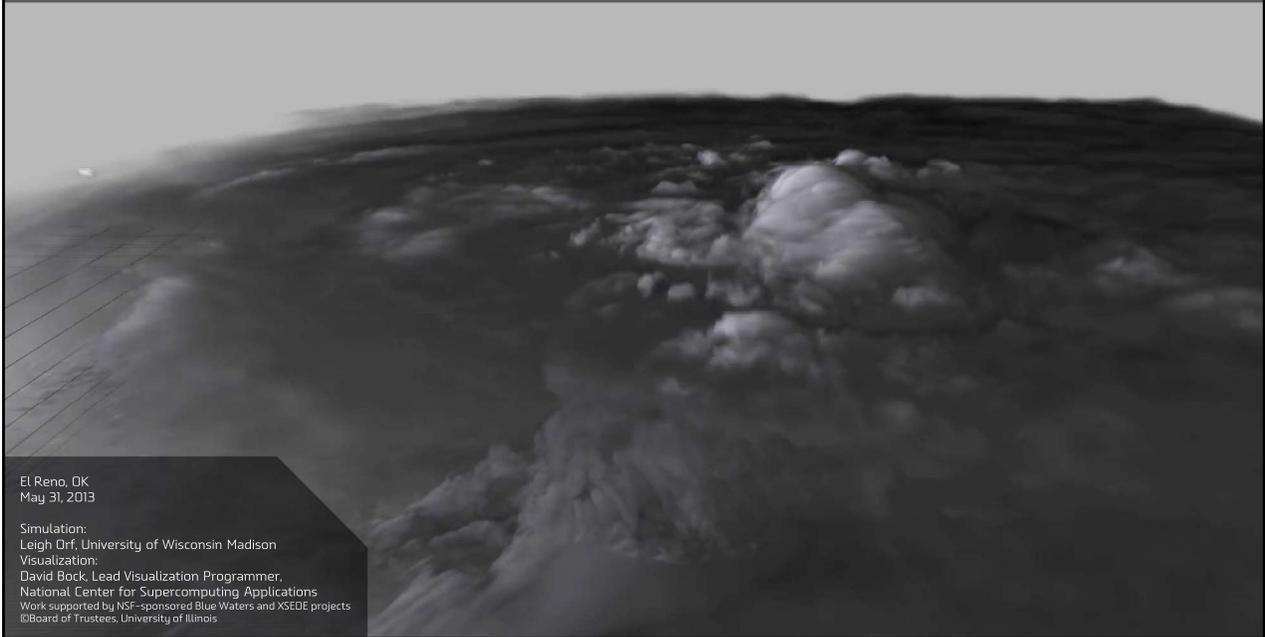
## SUPERSTORMS / Of Supercells...



Earth, TX - May 16, 2021  
8:40pm-8:50pm  
Photography: Isaac Schlueshe  
@stushywx

- This video was captured on Earth in a town *called....* Earth.... In my home state of Texas. We've got some of the best names for cities. The photographer, Isaac Schlueshe, attends classes at the University of Wisconsin Madison and said that to make this video he took 516 pictures over a 10 minute period. It's honestly one of the best supercell timelapses that I found in my research, better than what we were using as reference. Supercells are a unique type of storm that can produce tornadoes and occur primarily in the spring and summer months in the central United States. Supercells are a big source of destruction and also important to regulating the climate so University and Government researchers in the United States are actively contributing to our understanding.

## SUPERSTORMS / Of Supercells...

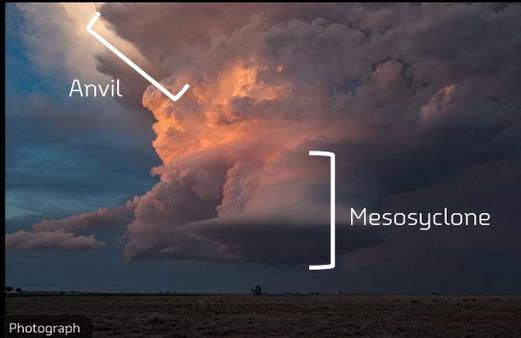


El Reno, OK  
May 31, 2013

Simulation:  
Leigh Orf, University of Wisconsin Madison  
Visualization:  
David Bock, Lead Visualization Programmer,  
National Center for Supercomputing Applications  
Work supported by NSF-sponsored Blue Waters and XSEDE projects  
©Board of Trustees, University of Illinois

- Raw atmospheric data gathered during the infamous supercell and tornado that struck El Reno Oklahoma in 2013 was used by Leigh Orf at the University of Wisconsin Madison to recreate the storm using a supercomputer. This ended up being extremely useful for modeling our superstorm because it lended us a view from multiple angles as well as a clear view of the underlying mechanics of a supercell. A few seconds ago I said that this particular storm was Infamous and that's because a tornado generated by it killed 8 people including the renowned storm chaser Tim Samaras. I highly recommend the Book, *The Man Who Caught The Storm*, by Brantley Hargrove if you want to learn more about the skill and dedication required to be able to study these storms up close as well as the history of storm chasing and the mechanics of supercells.

## SUPERSTORMS / Of Supercells...



There are four key features of supercells that we wanted to use to make our superstorms realistic.

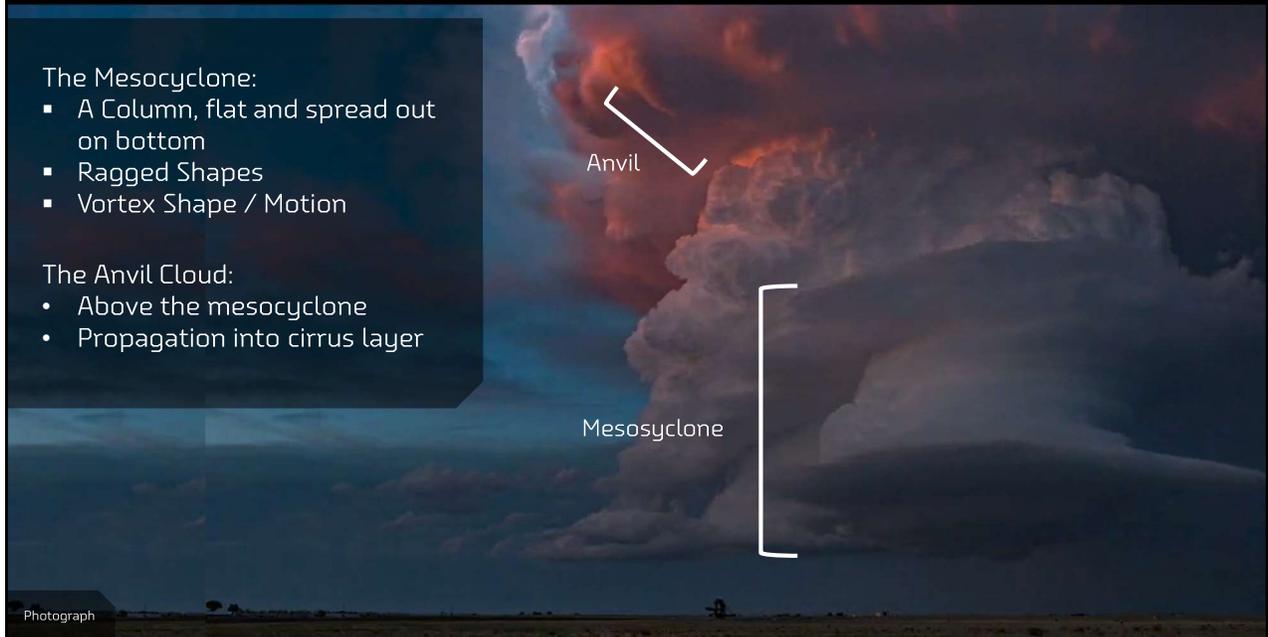
- The most important is the swirling monolithic and cylindrical “foot” of the cloud called a mesocyclone. Here, air moves in a vortex about a column of low pressure, slow on the outside of the mesocyclone and fast on the inside. Near the center is where tornadoes can form.
- Besides the mesocyclone there are the towering Anvil clouds that flatten at the top and spread into the upper cloud layers.
- There are also unique lighting effects, like internal lighting and ambient lighting sources. You just don’t see internal lighting in clouds like this unless it’s a massive cloud formation with abnormal shaping. This makes it a unique characteristic. Also, lightning and the lighting effects associated with it play a big part in making supercells unique.

### The Mesocyclone:

- A Column, flat and spread out on bottom
- Ragged Shapes
- Vortex Shape / Motion

### The Anvil Cloud:

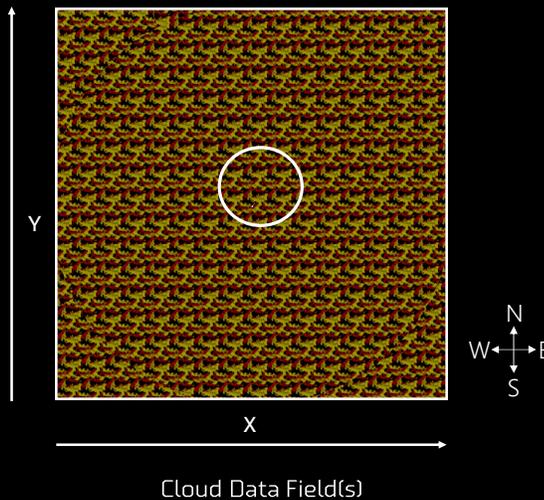
- Above the mesocyclone
- Propagation into cirrus layer



Lets look at how we modeled density to create the mesocyclone and anvil.

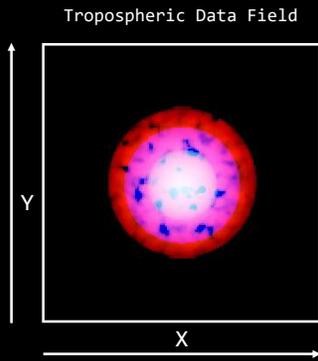
- The mesocyclone is a vertical cloud column that spreads out at the base. The underside is detailed with ragged wispy and billowy cloud shapes. There is also a distinctive vortex that manifests with slowly moving spiral tendrils near the outside and faster spinning, stretched out clouds near the center.
- The Anvil cloud is positioned above the mesocyclone. This is where rising cloud material meets another cloud layer and punches through, spreading into the cirrus cloud layers.

Rather than model these features separate from the rest of the clouds in the sky, we decided to integrate them as a function of the cloud system itself, which reduced the unique code and instructions required – something good for performance and sanity!



We also wanted to localize the superstorms. So, The Nubis Data Field generator was expanded to support Superstorm stencils so that we could avoid affecting the entire cloud map.

- Using a position, radius and coverage, stencils would only affect the region that they occupied.
- The modeling data would be overridden for the tropospheric and cirrus layers to create the mesocyclone and anvil clouds. First, lets look at how we modeled density with the stencil itself.



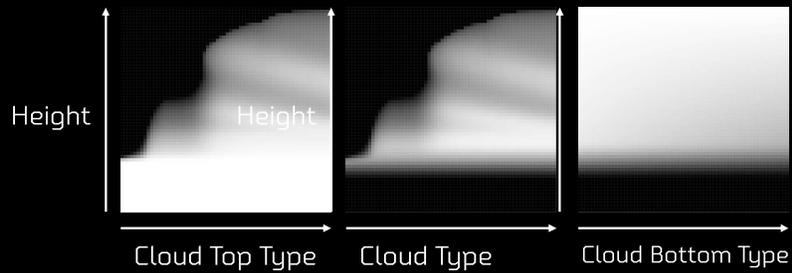
With the Nubis Coverage-Type modeling approach, its quite easy to make a cloud mountain. You just need to increase cloud coverage and type radially toward a point.

- So, Within the superstorm radius Several noises were composited and blended to their maximum values at the center to produce the characteristic cylindrical foot of the mesocyclone. Though admittedly, without the anvil it looks kind of like a sombrero. Next, We add some detail underneath.

	Cloud Top Type	
Cloud Coverage	Cloud Type	Superstorm Coverage
	Cloud Bottom Type	

In order to produce ragged details on the underside of the mesocyclone,

- the original Coverage/Type cloud model was expanded to support an additional bottom cloud type that could change the noise shapes on the bottoms of clouds to something more ragged.
- Also, in order to blend lighting and other modeling behaviors unique to the superstorms, we added superstorm coverage data to the model.



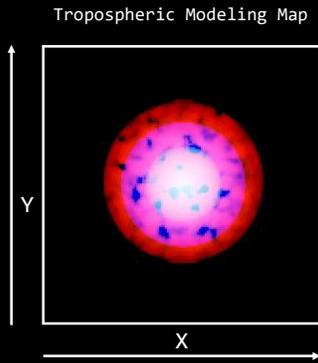
The vertical profile gradient was split into bottom and top gradients so

- that the bottom and top types could be applied independently. The results of the two lookups were multiplied to produce the vertical profile gradient for a given sample.



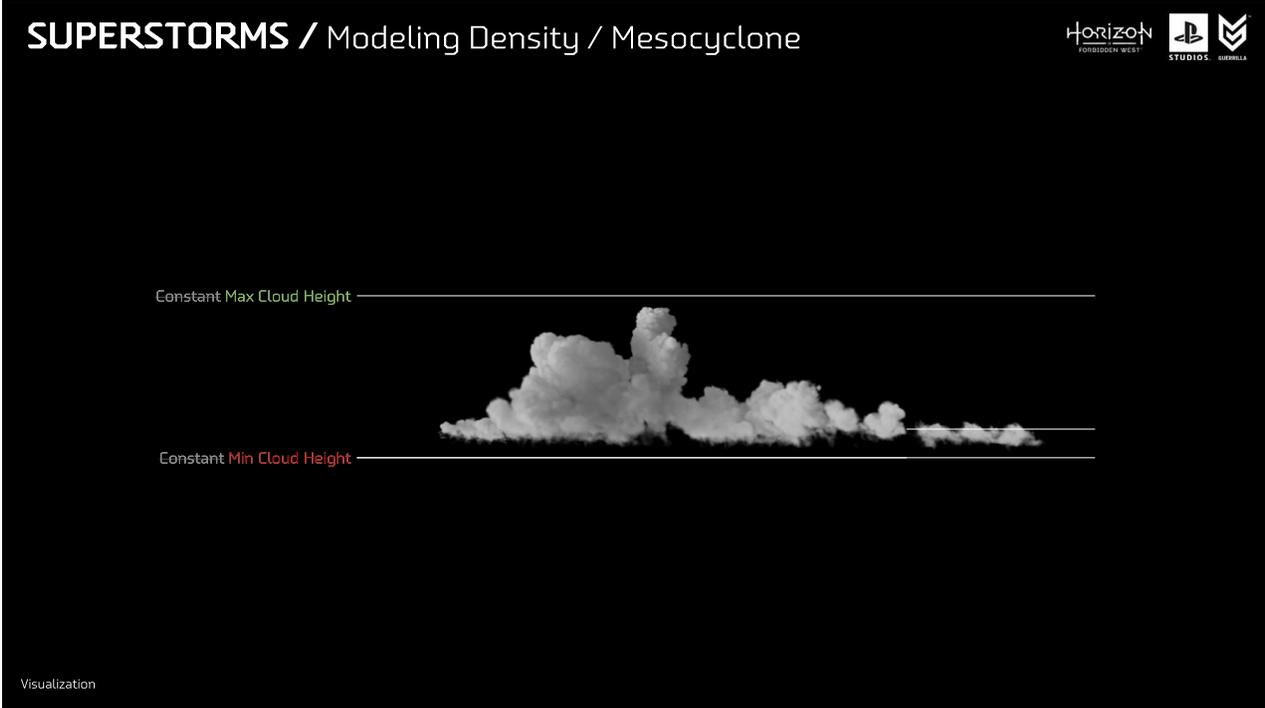
Here it is in practice.

- As bottom type increases, the bottom vertical profile curve relaxes and the noise types blend to wispy shapes. This gives us that torn ragged look on the underside of the mesocyclone. Now to model those spiral shapes.



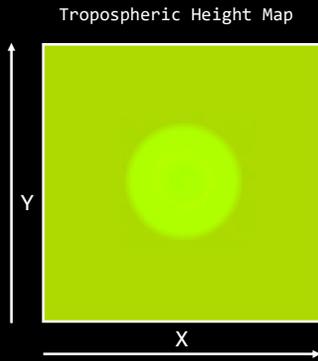
We

- twisted these noises around the center of the mesocyclone to produce the characteristic vortex tendrils at the base of the mesocyclone. Although the bottom of a mesocyclone flattens out, the underside is anything but flat.



The original nubis system was extended to allow for varying cloud heights.

- Min and Max height were encoded to a new data field so that we could alter the superstorm heights as needed.

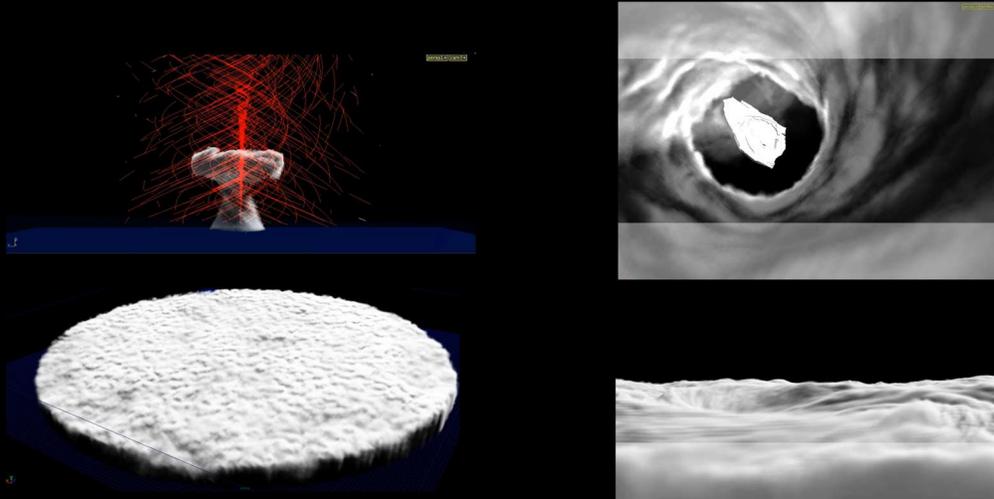


We

- used the Nubis Data Field Generator to create a set of distorted noises and radial gradients to use as signals for a height adjustment to give the underside more definition. Here is the result.



So, by adding two new modeling data, Bottom Type and Superstorm Coverage, and by adding variable height to the cloud layer, we were able to produce a promising approximation of a mesocyclone. But this is only half of work. We also need to make it move.



"MDR Vortex Field"  
Sine/Cosine rotation concept by Matthew D. Roach  
Fluid Simulations by Andrew Schneider (Houdini/Maya)

I worked in feature animation before this job and had the opportunity to make several volumetric vorticies using fluid simulation. It seems to be an effect that I can't escape 😊 Fluid simulations can produce realistic motion but they have a way of flowing out of control over time causing big changes in the overall shape if you aren't careful.

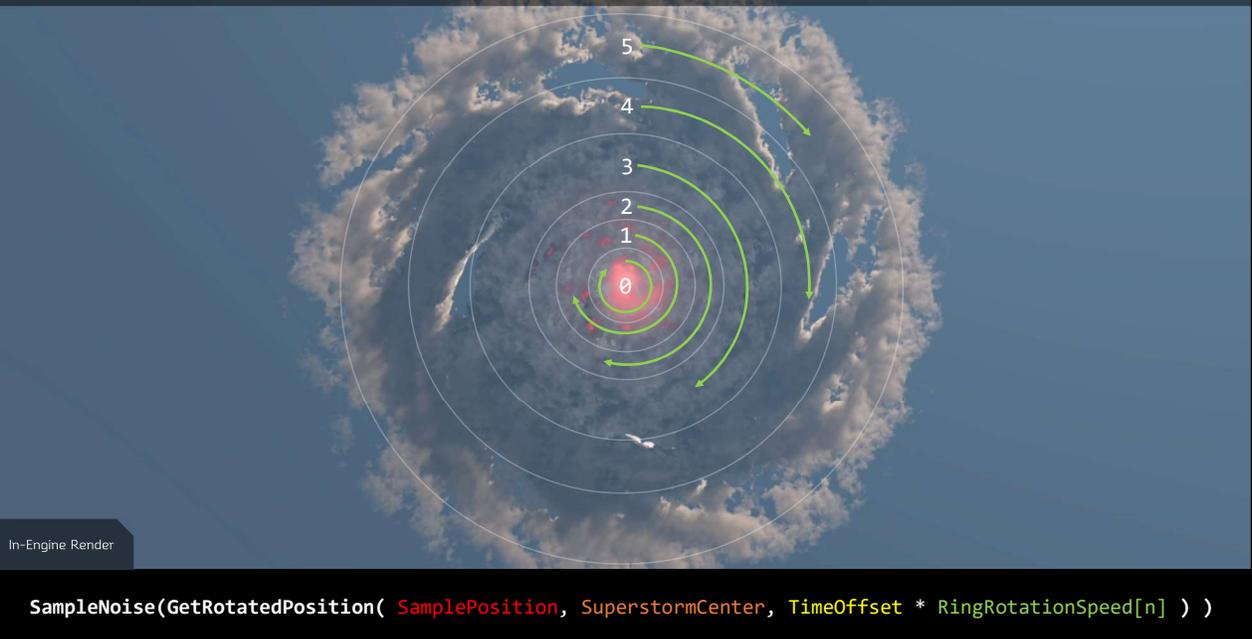
- For something like the underside of a mesocyclone the solution would be to advect an initial cloud pancake around in a vortex. Once you have a couple of seconds of convincing vortex motion, you just render it and use it in a shot. Unfortunately, given that this was a continuous effect that had to start and stop over several minutes and that we not only needed to do this on the PS5 but also on the PS4, Fluid simulation was out of the picture.
- The other option is to try to fake fluid vortex motion using rotations and some trickery as I did in some other vortex effects in the past at Blue Sky Studios.



In-Engine Render

```
SampleNoise( GetRotatedPosition( SamplePosition, SuperstormCenter, TimeOffset ) )
```

So First, we tried rotating the detail noise sample positions about the center of the mesocyclone over time. In this pseudocode example the third component to the rotation function is animation offset. This was promising but real vortices like those in a supercell mesocyclone spin faster at the center and slower at the edges.



At guerrilla we like to opt for solutions that give us the greatest artistic control. So rather than using a generalized technique like flow maps, we decided to explicitly

- determine rotation speeds in a set of nested rings.
- The mesocyclone detail noise rotation speeds would decrease incrementally over distance from the center. To prevent visible cuts where one ring ends and another begins, we overlap the rings, sample noise at both speeds, and then interpolate the results across the overlap. Note that this does mean that you sample noise twice in the overlap. I'll explain how we avoided this extra work on Playstation 4 a bit later.



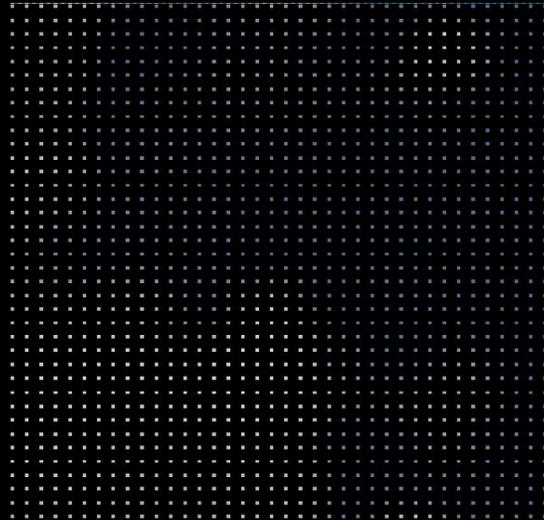
```
SampleNoise(GetRotatedPosition( SamplePosition, SuperstormCenter, TimeOffset * RingRotationSpeed[n] + RingSkew[n] ) )
```

To get a little bit of the Fluid simulation look back, we added some distortion to the noise.

- Noise sample positions for rings nested outside faster moving rings were twisted about the center of the mesocyclone in order to create the illusion of fluid boundary interactions.



So, all of this produces a convincing procedurally created and animated mesocyclone. However there is one caveat. What is depicted here is rendered without temporal upscaling. This is useful when showing how the vortex was created for a GDC talk, but not in gameplay where the costs for full resolution renders of clouds are prohibitive.

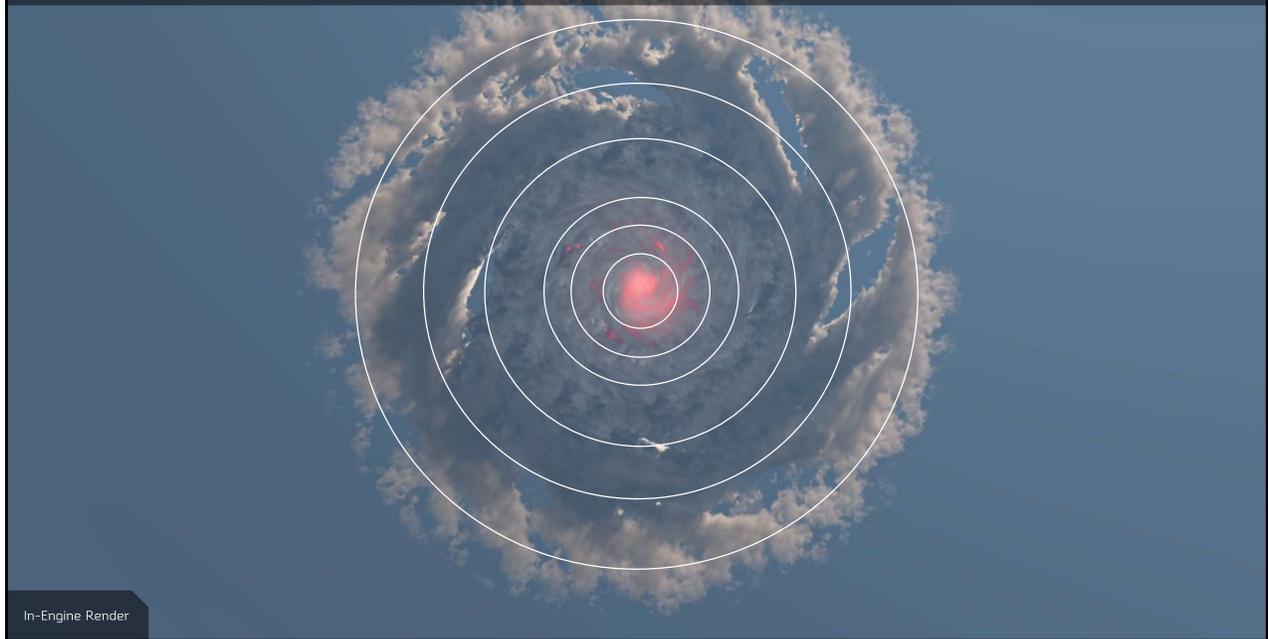


Recall that Nubis constructs a cloud render

- over several frames to reduce the total render time required. This is reliable for slow-moving volumetrics where you can take 16 frames to draw a full res image.



However, this can pose a problem when the cloud moves faster than the image can be constructed - as seen here. It produces a sort of pixel soup. This is because there is not enough time to resolve a shape before that shape has moved somewhere else in frame. The only way to mitigate this is to tell the upscale shader where the shape will be in the next frame and order the renders accordingly.



Because we made the decision to divide the underside of the mesocyclone into

- 6 rings with different rotation speeds, we could calculate these translation values, known across the industry as motion vectors, for each ring and use this to solve the serious temporal artifacts that would occur in the fast-moving vortex.

## SUPERSTORMS / Animating Density / Mesocyclone



```
// Get world space cloud position
float3 view_space_vec = CreateEyeRay(inViewportUV, inFovScale);
float cloud_distance = inCloudAttrWorkingBuffer.SampleLOD(inSampler, inUV, 0).r;
float3 cloud_world_space = mul(inInvViewMatrix, float4(view_space_vec * cloud_distance, 1.0)).xyz;

// Rotate around superstorm center
float rotation_speed = GetSuperstormRotationSpeed(cloud_world_space.xy, superstorm_center, superstorm_radius, superstorm_blend_factor);
float2 rotating_motion_offset = GetRotatedPosition(cloud_world_space.xy, superstorm_center, rotation_speed * inDeltaTime);
float3 superstorm_rotated_world_space_position = float3(0.0, 0.0, cloud_world_space.z);
superstorm_rotated_world_space_position.xy = rotating_motion_offset;

// Get superstorm mask for blending - powered linear distance from radius to center of superstorm
float superstorm_mask = pow(saturate(1.0 - length(cloud_world_space.xy - superstorm_center) / superstorm_radius), 0.1);

// Blend vectors from normal to superstorm over superstorm mask.
cloud_world_space = lerp(cloud_world_space, superstorm_rotated_world_space_position, superstorm_mask);
view_space_vec = mul(inViewMatrix, float4(cloud_world_space, 1.0)).xyz;

// Construct previous sample position from new view space vector
float4 prev_sample_pos = mul(inReprojectionMatrix, float4(view_space_vec, 1.0));
prev_sample_pos /= prev_sample_pos.w;
prev_sample_pos.xy *= float2(0.5, -0.5);
prev_sample_pos.xy += float2(0.5, 0.5);
```

Here is the actual code that we used to make these calculations. It's a lot of text that you can review later when the slides are published but for now I will summarize the steps involved.

- First, we generate a world space position for the point where the view ray from the given pixel intersects the cloud.
- Next, we rotate the position around the center of the superstorm using the same rotation transformations that would be used for the noise deformations at that position in space.
- Then we blend between the normal sample position and the superstorm translated position using the linear distance from the radius to the center of the superstorm as the blend factor.
- Finally we construct a 2D UV offset for the previous position from the resulting 3d vector.

This vector can then be used in the upscale shader to order the pixel updates..



So that our pixel soup becomes... A more temporally stable vortex. We were able to increase the highest rotation speed to something realistic for the scale of the storm while maintaining temporal stability. As it turns out, this motion vector solution was also applicable to a much broader use case...

## SUPERSTORMS / Animating Density / Mesocyclone



```
// Get world space cloud position
float3 view_space_vec = CreateEyeRay(inViewportUV, inFovScale);
float cloud_distance = inCloudAttrWorkingBuffer.SampleLOD(inSampler, inUV, 0).r;
float3 cloud_world_space = mul(inInvViewMatrix, float4(view_space_vec * cloud_distance, 1.0)).xyz;

// Rotate around superstorm center
float rotation_speed = GetSuperstormRotationSpeed(cloud_world_space.xy, superstorm_center, superstorm_radius, superstorm_blend_factor);
float2 rotating_motion_offset = GetRotatedPosition(cloud_world_space.xy, superstorm_center, rotation_speed * inDeltaTime);
float3 superstorm_rotated_world_space_position = float3(0.0, 0.0, cloud_world_space.z);
superstorm_rotated_world_space_position.xy = rotating_motion_offset;

// Get superstorm mask for blending - powered linear distance from radius to center of superstorm
float superstorm_mask = pow(saturate(1.0 - length(cloud_world_space.xy - superstorm_center) / superstorm_radius), 0.1);

// Blend vectors from normal to superstorm over superstorm mask.
cloud_world_space = lerp(cloud_world_space, superstorm_rotated_world_space_position, superstorm_mask);
cloud_world_space = lerp(cloud_world_space + scroll_direction_2D * inDeltaTime, superstorm_rotated_world_space_position, superstorm_mask);
view_space_vec = mul(inViewMatrix, float4(cloud_world_space, 1.0)).xyz;

// Construct previous sample position from new view space vector
float4 prev_sample_pos = mul(inReprojectionMatrix, float4(view_space_vec, 1.0));
prev_sample_pos /= prev_sample_pos.w;
prev_sample_pos.xy *= float2(0.5, -0.5);
prev_sample_pos.xy += float2(0.5, 0.5);
```

Normal Clouds that scroll in one direction. Generating motion vectors for scrolling clouds is as simple as multiplying the scroll direction by a timestep. Instead of blending between static cloud motion vectors and superstorm motion vectors at this point, we just need to blend between scrolling vectors and superstorm vectors.



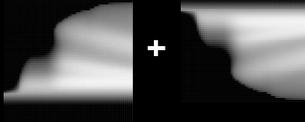
Here is the result. On the left we have scrolling clouds with no motion vectors for upscaling and on the right we use motion vectors. There is an upward limit to scroll speed of course, but this is far faster movement than was possible before in our system.



New let's look at how we modeled the anvil cloud.

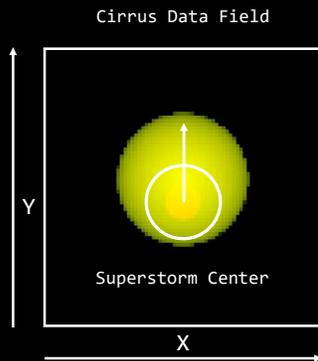
Recall that the characteristic anvil forms as the billowing top of the mesocyclone rises into the upper cloud layers and flattens out and spreads in the wind direction. We modeled this in the tropospheric and cirrus cloud layers.

Vertical Profile Gradient



In the tropospheric layer, for the areas where superstorm coverage was nonzero,

- we flipped the Vertical profile gradient and combined that lookup result with the normal vertical profile gradient in order too produce the horizontal spreading at the top of the cloud layer. We also rotated and skewed the noises in this area to produce a more wind-swept look.



For the cirrus layer, we made some changes to the the Nubis Data Field generator

- We used the superstorm position, radius and wind vector
- to create a cirrus cloud formation above the superstorm mesocyclone and offset in the wind direction.



So now, instead of a sombrero cloud, we have something that feels ominous and connected to the higher altitude cloud layers and that could be seen from great distances around the game world to remind the player that the weather system was still malfunctioning.



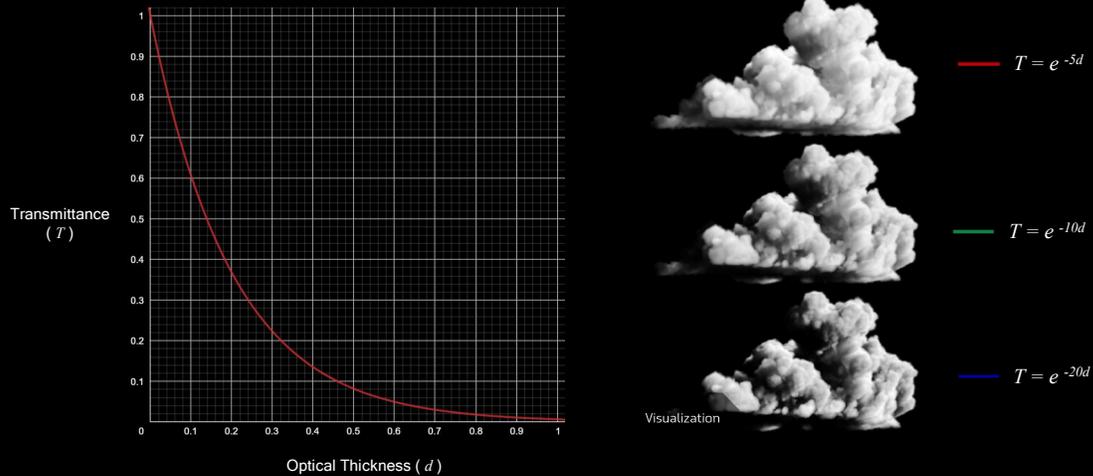
Since the superstorm used the same modeling data as the rest of the cloud system,

- this transition from cloud to superstorm was seamless.

Now that we have covered the modeling of density as well as the animation of the density of the superstorm, we can take a look at how we modeled light differently for the superstorm.

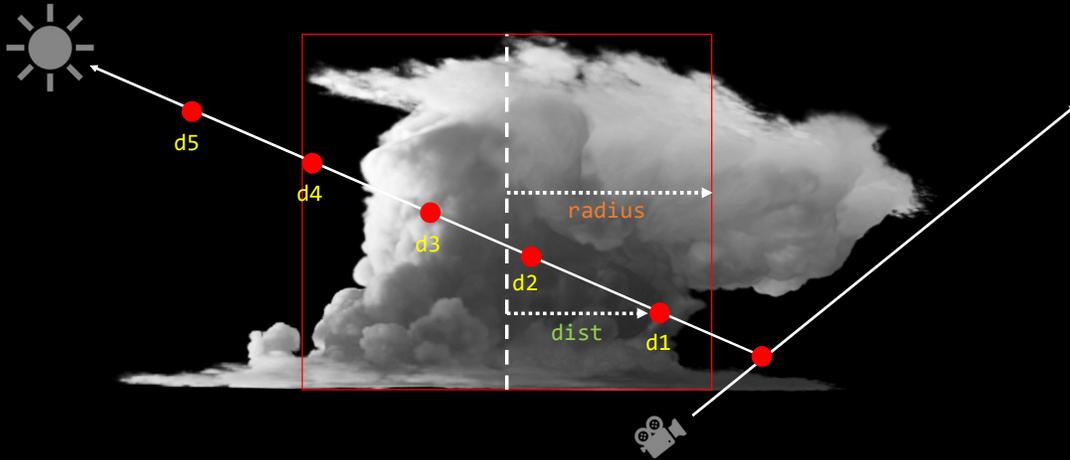


The real-world phenomenon of supercells present several unique lighting features that we wanted to simulate. They are so dense that they can appear quite dark. The clouds are so low to the ground that they can pick up a lot of ambient light that bounces from the ground. In addition to the natural effects we would also need to model the sinister internal red glow. First let's look at how we modeled direct scattering from the sun for the superstorm.



Recall that we use the Beer-Lambert equation to approximate light attenuation in the direct scattering component of our lighting model.

- By artificially increasing the optical thickness, we can make clouds appear darker and thicker. However, we only want to apply this effect for the superstorm. So, we need to model a probability field to dictate where this can happen.



```
density_scale = pow(min(1.0 - (dist / radius), 0.0) + 1.0, 0.5);  
summed_density += d(n) * density_scale;  
transmittance = exp( -1.0 * (summed_density));
```

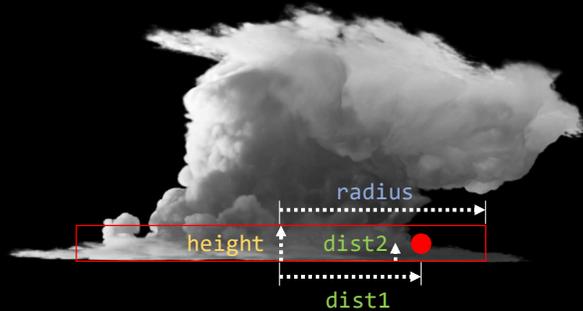
Visualization

Here is an example of a cloud with a light ray march that samples density along the light ray.

- Let's focus on one of these samples.
- we describe the probability of an increase in the density sample scale by its proximity to the superstorm center.
- If our light sample falls within a given radius from the center of the superstorm, we can increase the density of the sample in the density sampler function itself.
- We repeat this for each sample and sum them along the ray.
- Then we use this altered density sum in the attenuation function.



Here is the result. You can see that as we increase the artificial attenuation scale, the cloud becomes darker and more menacing. It's a simple and old trick, but it works when applied locally in this way.



```
amb_settings_blend = min(1.0 - (dist1 / radius), 0.0) * (dist2 / height);  
ambient_scattering_settings = lerp(cloud_amb_settings, superstorm_amb_settings, amb_settings_blend);
```

Visualization

Next, let's look at ambient light effects. We wanted to be able to independently control the amount of ambient light contribution on the underside of the superstorm mesocyclone.

- The solution was simple. We used a cylindrical probability field like the one used for attenuation but with a decrease in probability over height to blend between ambient scattering settings for the clouds and the superstorm .

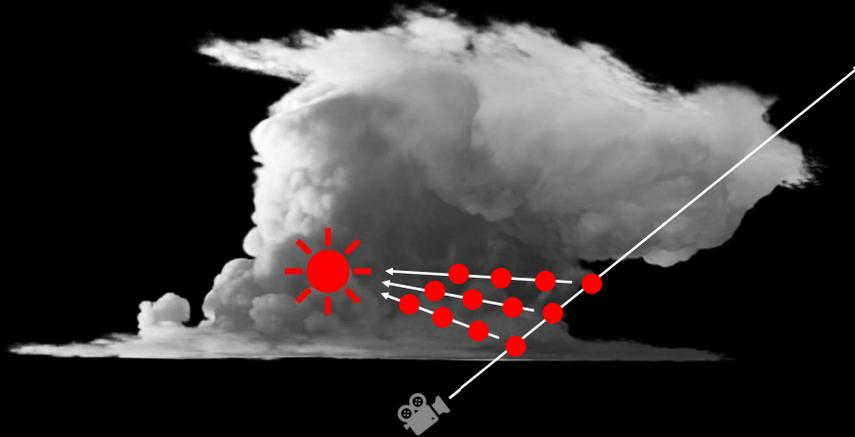


And here is the result in-engine. You can see that the interior gets an ambient boost while the exterior and by extension the rest of the clouds do not.



In-Engine Render

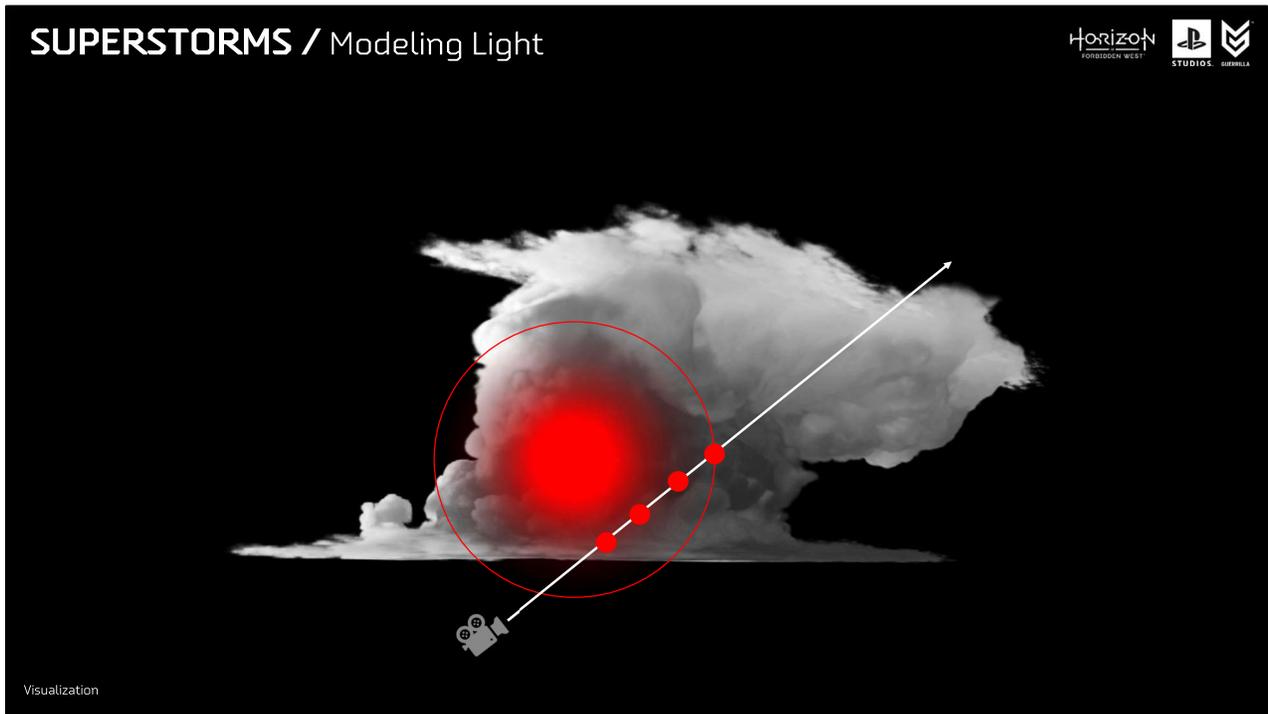
To create a constant sinister internal red glow to communicate that this is a destructive energy effect, we created Internal light source that avoids a second light-ray march...



Visualization

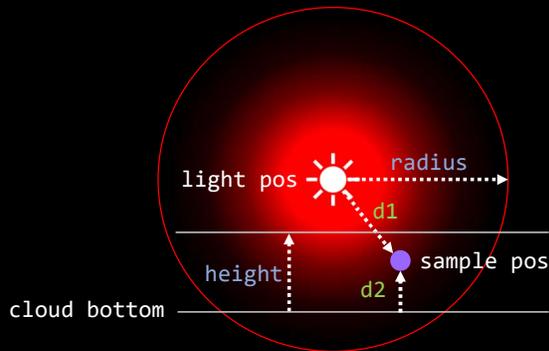
Normally, for any light source that is positioned inside a volumetric effect,

- you need to do an expensive second light ray march for each sample taken. For those keeping track that would mean 3 Ray-marches. For a large scale volumetric effect like clouds which can cover most of the screen, this is not an ideal approach even on the newer Playstation Hardware. Especially for games like Horizon that also have to render a lot of other content on the ground.



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.



```

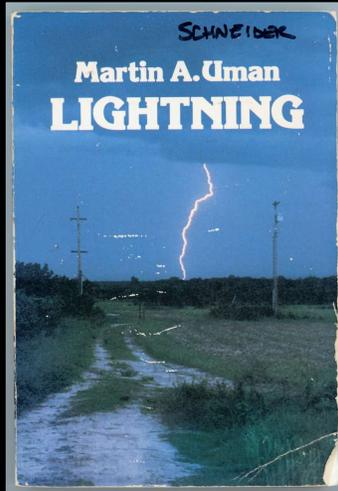
potential_energy = pow( 1.0 - (d1 / radius), 12.0);
height_gradient = (d2 / height);
pseudo_attenuation = (1.0 - saturate(fine_density * 5.0));
glow_energy = potential_energy * height_gradient * pseudo_attenuation;
light_energy = direct_scattering + ambient_scattering + glow_energy;
    
```

- First we define a spherical volume that represents the area where light from the internal glow could possibly be detected and we
- define our potential energy as a function of a powered linear distance from the sample to the center of the light.
- We defined a height gradient representing the light attenuation along the ray from the sample to the light position. But density is not homogeneous in the space between the sample and the light source. To fake the effect of heterogeneous density along the imaginary ray
- we define another component that represents the probability of light absorption due to local density variation using the fine density sample data.
- To approximate the light energy present at the sample position, we multiply these three components together.
- Then this light intensity value was added to the result of the light energy function.

## SUPERSTORMS / Lightning Effects

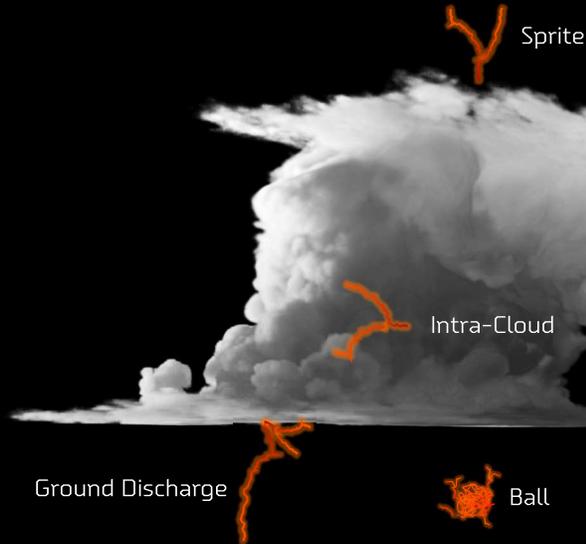


The superstorm was supposed to be a source of destructive energy that discharged over time through red lighting. If you are going to take care to attempt a reality-based approximation of lightning for film or games then there is one book you need to know about. The title is easy to remember.



“Lightning” by Martin A. Uman. Mr. Uman has been researching and publishing about lightning for decades. This book is from 1969. Sorry about my name being written on the cover, this is from an earlier time where people read physical books and you had to label what was yours in the office. What we took away from this book was information about lightning classifications, behaviors and timing.

## SUPERSTORMS / Lightning Effects



Visualization

There were two forms of lighting that were relevant for the superstorm effect:

- intra-cloud lighting – bolts that happen inside of the cloud – all you see are internal flashes.
- And Ground discharge lighting – the bolts that go from clouds to ground
- There are other forms of lighting like sprites which form on top of clouds and ball lighting. These are so rare and strange that they deserved a mention here in my opinion. But Intra-Cloud and Ground discharges were the most applicable for our purposes.

## SUPERSTORMS / Lightning Effects



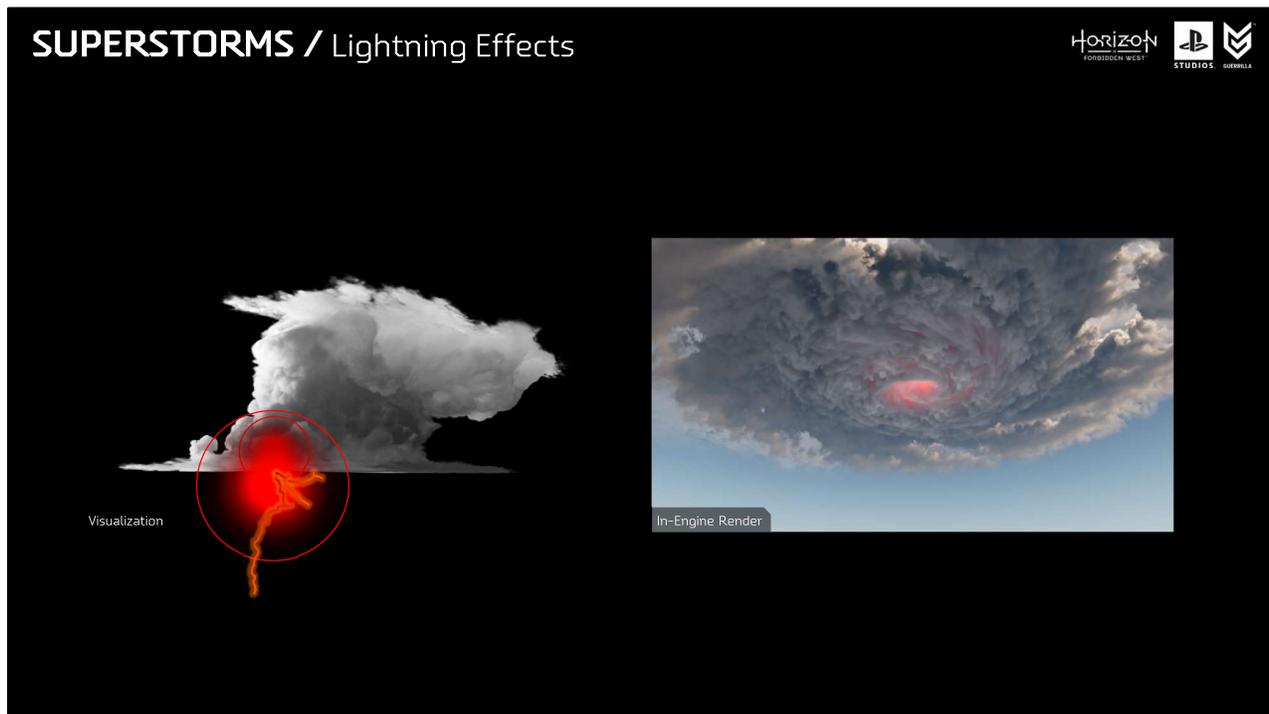
Visualization



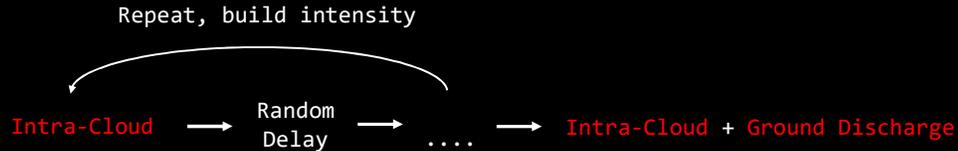
In-Engine Render

For the intra-cloud lighting effects, we used the same approach as the internal red glow, we just fed it a variable position for each flash and animated the intensity over the lifetime of the flash in patterns that matched those found in Martin Uman's research material.

## SUPERSTORMS / Lightning Effects



- The ground discharge effects were the combination of a particle bolt
- and internal flash
- and external flash. The external flash was a modification of the internal flash that added highlights to cloud features on the underside of the mesocyclone.



We used Decimas Effect Graph tools to control lighting effects and pass the required data to The Nubis Renderer, particle effects and sound effects.

- First, the system would execute several intra-cloud effects while increasing the intensity of the flashes each time.
- Then it would fire off 1 to 2 Ground discharge effects. There was one problem though.

## SUPERSTORMS / Lightning Effects

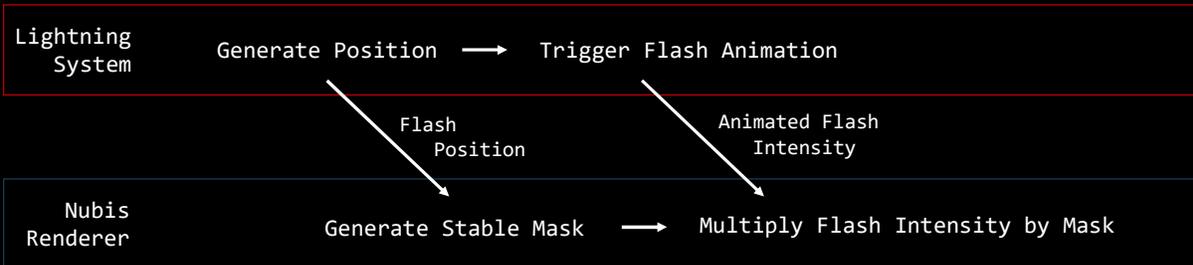


No Solution

Our Solution

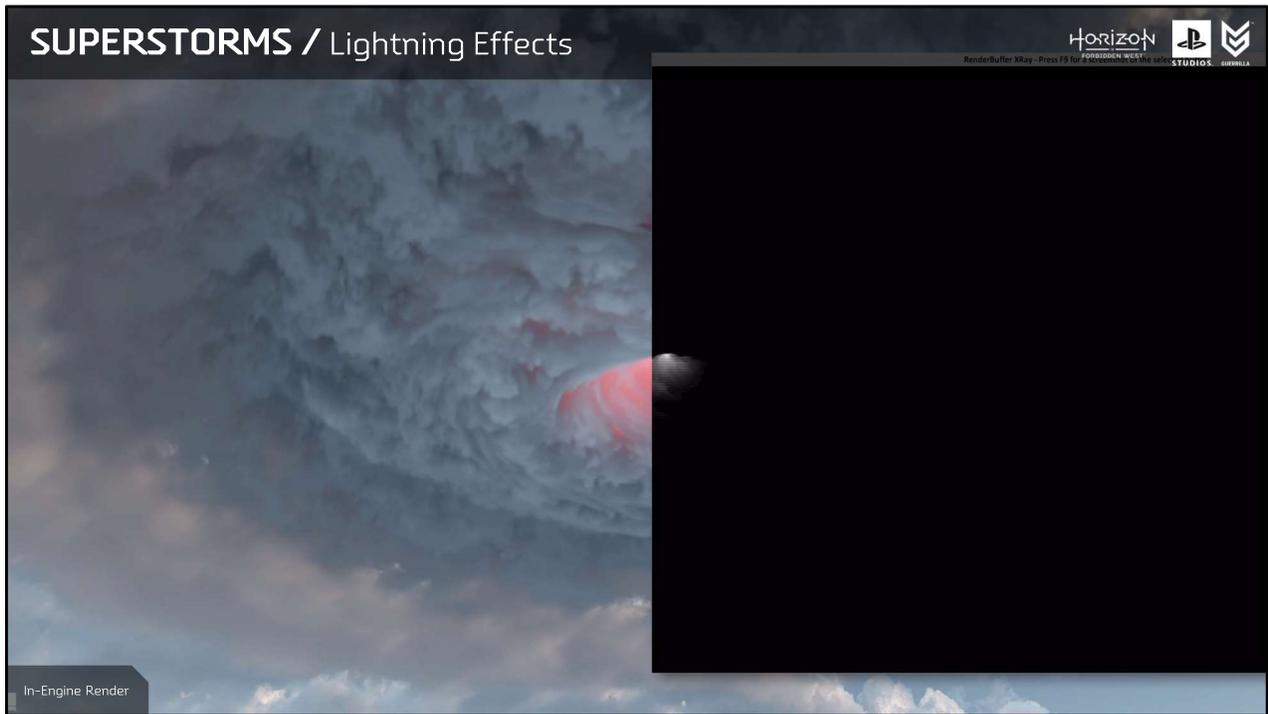


Recall once again that we build cloud renders over 16 frames. The lighting flashes can sometimes occur over 4-8 frames, which means that the results would appear less intense and degraded by temporal artifacts.



To solve this issue, Nathan, my co-partner in cloud related crimes against the GPU, suggested that we Generate a mask using position and radius data before each flash So this could be used to modulate the internal flash intensity over the duration of the animation.

- After the lightning system generates a flash position,
- this is passed to the Nubis Renderer to generate the mask.
- Then when the lightning system triggers the flash animation,
- the animated intensity is passed to the
- renderer where it is multiplied by the mask.



Here is a view of the mask that gets generated for every flash. You can see that it builds to a full image in time for the flash and then dissolves before the next flash.

## SUPERSTORMS / Lightning Effects



In-Engine Render

And here is the final result.

## SUPERSTORMS / Environmental Effects



I have talked a lot about the superstorm itself at this point. But in order for it to be a powerful, ominous and dynamic presence in the game, it should dramatically change the environment around it, exposing the player to a darkened world with high howling winds and dust, snow or rain.



In-Engine Render

The superstorm quite literally casts a large shadow on the world which helps to integrate it into the landscape and make the environment generally more spooky underneath.



In-Engine Render

For Desert environments, The superstorms would produce dust storms or high wind, so the ambience effects were adjusted to produce low visibility and warm colors. Dust particle effects as well as heavy haze.



In-Engine Render

For Snowy environments, Heavy haze and snow particle effects to go with windspeed increases.

**SUPERSTORMS** / Environmental Effects / Lighting & Ambience



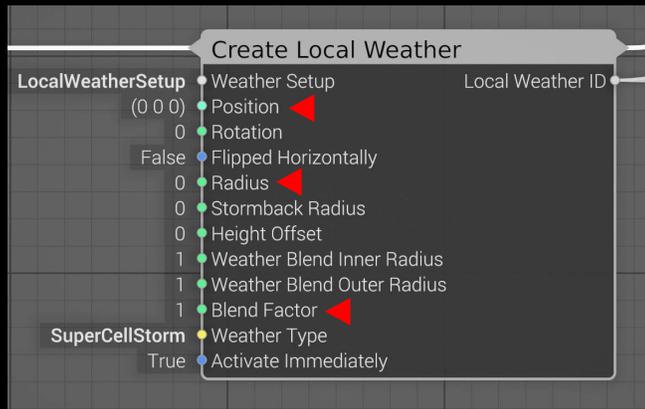
In-Engine Render

For Green environments with moderate temperatures, rain and heavy wind effects along with heavy haze.



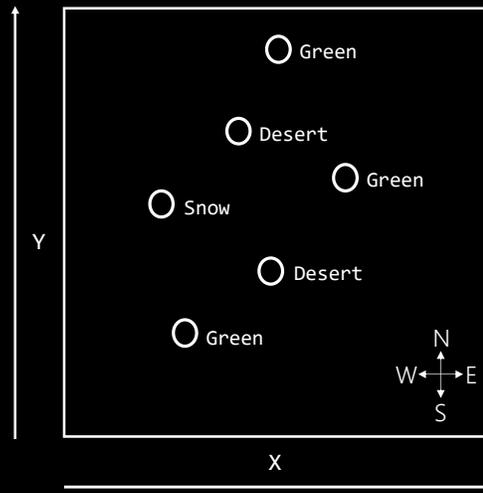
Ok, at last we come to the most ironic part of this talk. Recall that the out-of-control weather system was causing these storms.

This is where I explain that in order for us to direct where and when this would happen, we developed a weather control system to simulate an out of control weather control system. Earlier I mentioned that we modified the Nubis data field generator to include superstorm stencils that could replace clouds with a superstorm in any given location on the map. I also just talked about how we altered the weather differently according to where the supercell formed. Well, we also needed to make these things happen randomly in different parts of the world throughout gameplay and smoothly transition from no superstorm to superstorm and back again.



- Like the lightning system, we relied on Decima's Effect Graph workflow to control when and where superstorms would occur as well as transitions between states.
- We defined a local weather event by Position, Radius and blend factor, which were animated over time to produce a gradual transition. These variables were what informed the Nubis Data Field Generator and Nubis Renderer about how to model and light the superstorms and it informed the lightning system about where to place the lightning effects.

Horizon Forbidden West World Map



There are

- 5 different locations in the world where the superstorm can randomly form and override the local climate. Each has their own settings so that the appropriate weather and ambience effects occur there.

## SUPERSTORMS / Controlling Chaos



Every few minutes the system picks a location, like vegas here, and begins to load a superstorm there over a period of 2 minutes.

## SUPERSTORMS / Controlling Chaos



In-Engine Render

The Storm persists for a few minutes, discharging its energy. And then...

## SUPERSTORMS / Controlling Chaos



...it dissipates over 2 minutes and the cycle repeats itself. That is until the player completes the quest that repairs the weather control system.



By now I'm sure you are asking How much did all of this cost?!? This is a question that we asked ourselves multiple times of day for the past few years. We were faced with a unique challenge for HFW. We wanted to make a game that could push even the newest ps5 hardware to the limits, while still providing an experience on PS4 that surpassed the original quality of Horizon Zero Dawn. Before I get into the differences between the superstorms on PS5 and PS4, lets look at how we scaled Nubis in general between both platforms.

## SUPERSTORMS / Scaling PS4 and PS5



	PlayStation 4	PlayStation 5
Max Resolution	960 x 540	1920 x 1080
Light Ray Samples	6	10
View Ray Samples	60 - 90	96 - 180
Blur Scale (Pixels)	2x	1x
Noise Texture MIP Level	1	0

On Playstation 5 the maximum resolution was Full HD, while on Playstation 4 it was quarter res 960 x 540

We took 10 light ray samples on Playstation 5 and 6 on Playstation 4

View ray samples were also reduced for Playstation 4, but we mitigated artifacts with a stronger blur filter on that platform.

Additionally, the 3d Noise textures were samples at a higher MIP level for PS4 to reduce access times.

This allowed us to meet our 2ms budget on both platforms.



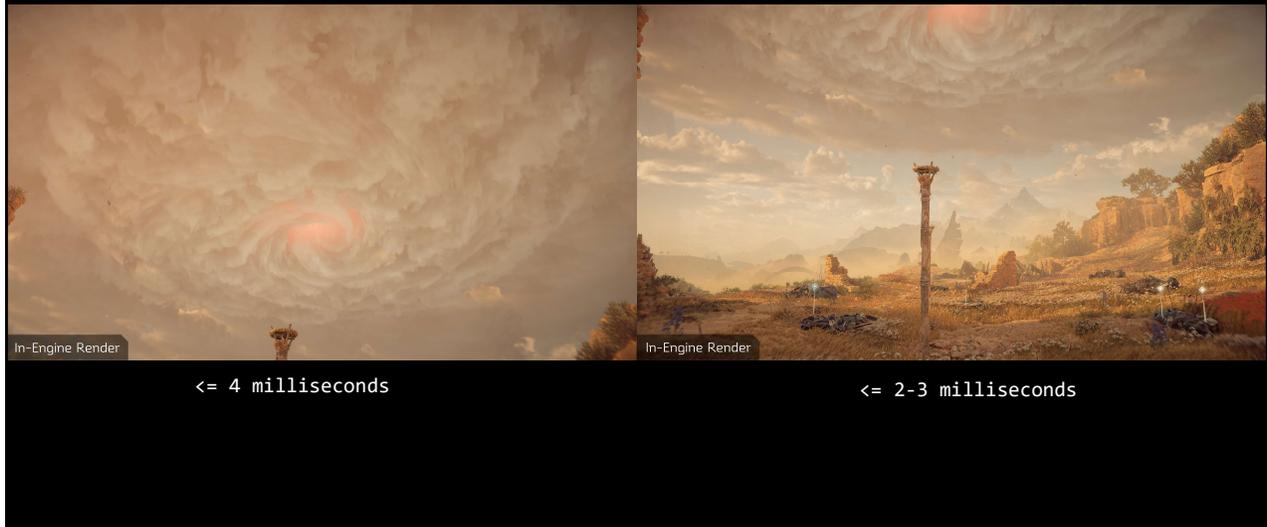
In the case of the superstorm, we made some adjustments as well. The blended double samples in the transition zones between rotating rings of Noise were replaced with a jittered boundary to avoid sampling noise twice in these areas. Here is the PS5 Version.

- And here is the PS4 version With all of the standard PS4 differences and the jittered transition between rings. (Toggle back and forth)



The top of the superstorm anvil was also simplified to a non-distorted billowing cloud shape. Here is the PS5 version.

- And the Ps4 Version. (Toggle back and forth)



We decided on two primary test cases for performance.

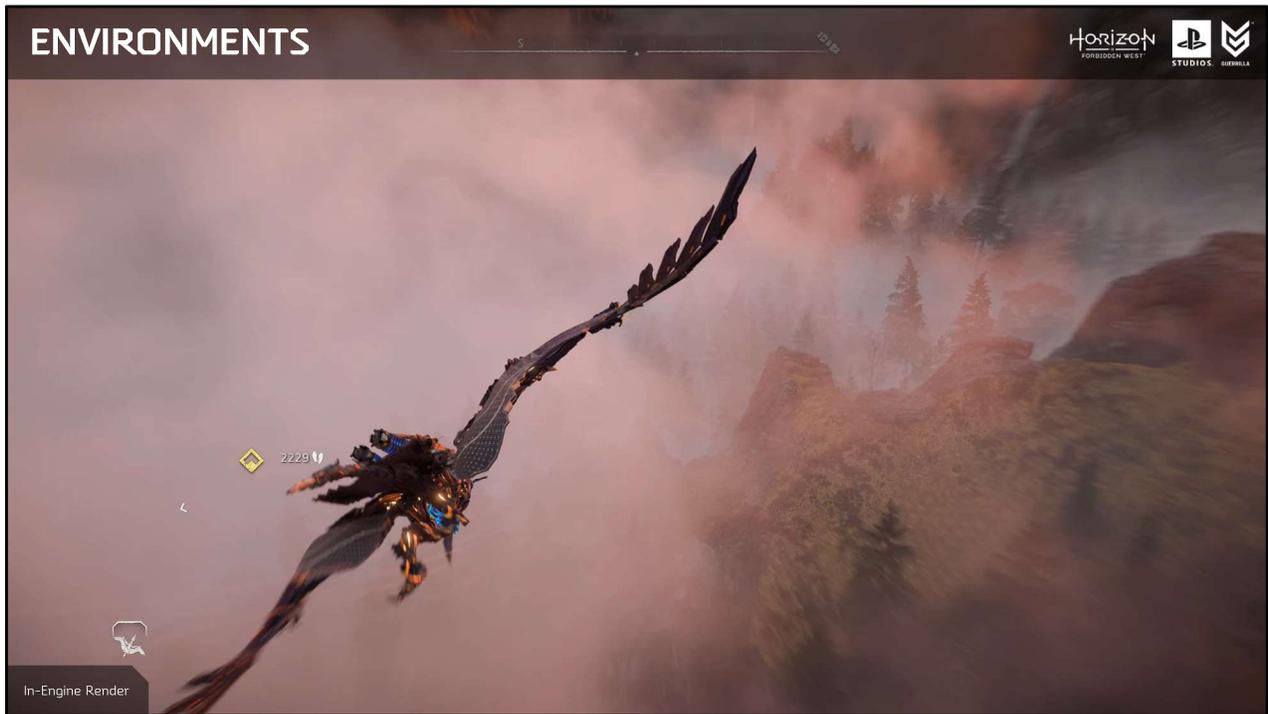
- Full frame coverage, such as when you look up at the superstorm from underneath,
- and half frame coverage such as when the superstorm only covered half of the frame. Because this was an effect, and an important one, we allowed our budget to grow beyond 2ms in these conditions.
- Max Cost Full Frame = 4ms – keep in mind that when you are looking up at the superstorm, that's the vast majority of what's being rendered.
- Max Cost Half Frame = 2-3ms

So it wasn't the cheapest thing to do on PS4, but because of where it sits into the camera window, it managed to fit and it saved us from a serious downgrade for PS4, something that is not fair to the players.



In-Engine Render

The Real-time Volumetric Superstorms of Horizon Forbidden West served to dramatically reinforce the idea that the world was falling apart around the player. I mentioned several goals about realism, dynamism and impact earlier, but there was one that I left out. We wanted to do this to prove that volumetric clouds could do more and should do more than just make pretty skies in games – that they could be effects themselves. For Horizon Forbidden West we also defined another new role for clouds in games.



As Environments. We will get back to the community soon about how we redesigned Nubis as a whole and extended it in another direction to allow the player to fly through clouds.

## /Guerrilla

Nathan Vos

Elco Vossers  
Bart van Oosten

Anton Woldhek  
Nick van Kleef

Jan-Bart van Beek  
Misja Baas  
Marijn Giesberts  
Roderick van der Steen  
Jeroen Krebbers  
Niklas Modrow  
Alexis Russel

Bryan Adams  
Ben McCaw  
James McLaren  
Mark van Berkel  
Angie Smets  
Michiel van der Leew

## /External

Chris Zimmerman

Matthew Roach  
Kirk Garfield

Isaac Schlueshe  
Leigh Orf  
David Bock

## /Other

Donald Sajda   
Jerry Ford   
David Kaul   
Joe Pasquale   
Malcom Kesson   
Clarke Stallworth 

Tim McLaughlin

Rosa, Aidan, Liam  
& Amelia 

## /References

Richard Hamblyn, *The Invention Of Clouds*. New York: Picador Reprints, 2011.

Brantley Hargrove, *The Man Who Caught The Storm*. New York: Simon & Schuster, 2018.

Martin Uman, *Lightning*. New York: Dover Publications, 1969.

Augustus Beer, "Bestimmung der Absorption des rothen Lichts in farbigen Flüssigkeiten" (Determination of the absorption of red light in colored liquids), *Annalen der Physik und Chemie*, vol. 86, pp. 78-88, 1852.

L. G. Henyey and J. L. Greenstein, "Diffuse radiation in the Galaxy," *Astrophysical Journal*, vol. 93, pp. 78-83, 1941.

Andrew Schneider. "Nubis: Real-Time Volumetric Cloudscapes in a Nutshell." *Eurographics*. Delft, NL, Web. 2018.

Andrew Schneider. "Nubis: Authoring The Real-Time Volumetric Cloudscapes Of Horizon Zero Dawn". *ACM SIGGRAPH*. Los Angeles, CA: ACM SIGGRAPH, 2017. Web. 2017.

Andrew Schneider, *GPU Pro 7: Real Time Volumetric Cloudscapes*. p.p. (97-128) CRC Press, 2016.

Andrew Schneider. "The Real-Time Volumetric Cloudscapes Of Horizon: Zero Dawn". *ACM SIGGRAPH*. Los Angeles, CA: ACM SIGGRAPH, 2015. Web. 26 Aug. 2015.

I want to thank my colleagues at Guerrilla who helped make this work possible. Specifically Nathan Vos and Jan-Bart van Beek. Also a few other people including my wife and three kids who were with me the whole way.



Questions



54 minutes total



REAL-TIME  
**VOLUMETRIC CLOUDS**  
FOR SKIES, VFX AND ENVIRONMENTS