For slides with proper formatting and video/audio use the PPTX version.

The following was presented at SIGGRAPH 2015 as part of the Advances in Real-time rendering Course. http://advances.realtimerendering.com

Authors: Andrew Schneider – Principal FX Artist, Nathan Vos – Principal Tech Programmer

THE REAL-TIME VOLUMETRIC CLOUDSCAPES OF HORIZON: ZERO DAWN

ANDREW SCHNEIDER | PRINCIPAL FX ARTIST

ADVANCES IN REAL-TIME RENDERING 2015

Thank you for coming.

Over the next half hour I am going to be breaking down and explaining the cloud system for Horizon Zero Dawn.

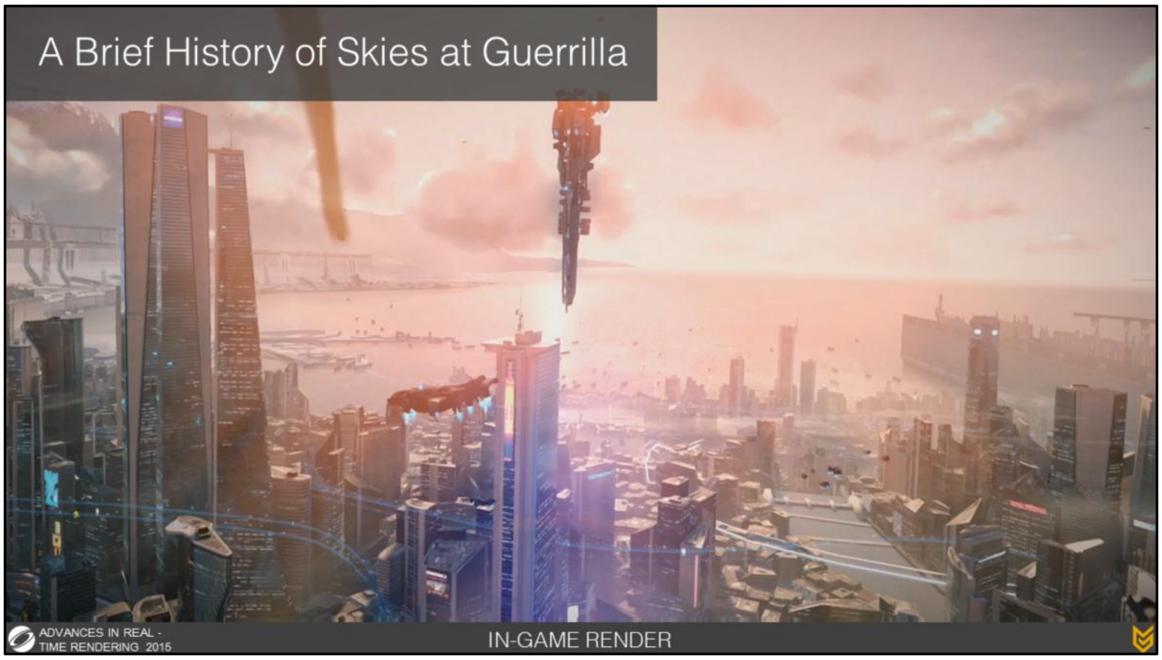
As Natasha mentioned, my background is in Animated film VFX, with experience programming for voxel systems including clouds.

This was co-developed between myself and a programmer named Nathan Vos. He could not be here today, but his work is an important part of what we were able to achieve with this.

Horizon was just announced at E3 this year, and this is the first time that we are sharing some of our new tech with the community. What you are seeing here renders in about 2 milliseconds, takes 20 mb of ram and completely replaces our asset based cloud solutions in previous games.

Before I dive into our approach and justification for those 2 milliseconds, let me give you a little background to explain why we ended up developing a procedural volumetric system for skies in the first place.

In the past, Guerrilla has been known for the KILLZONE series of games, which are first person shooters .



A Brief History of Skies at Guerrilla

FPS usually restrict the player to a predefined track, which means that we could hand place elements like clouds using billboards and highly detailed sky domes to create a heavily art directed sky.

These domes and cards were built in Photoshop by one artist using stock photography. As Time of day was static in the KILLZONE series, we could pre-bake our lighting to one set of images, which kept ram usage and processing low.

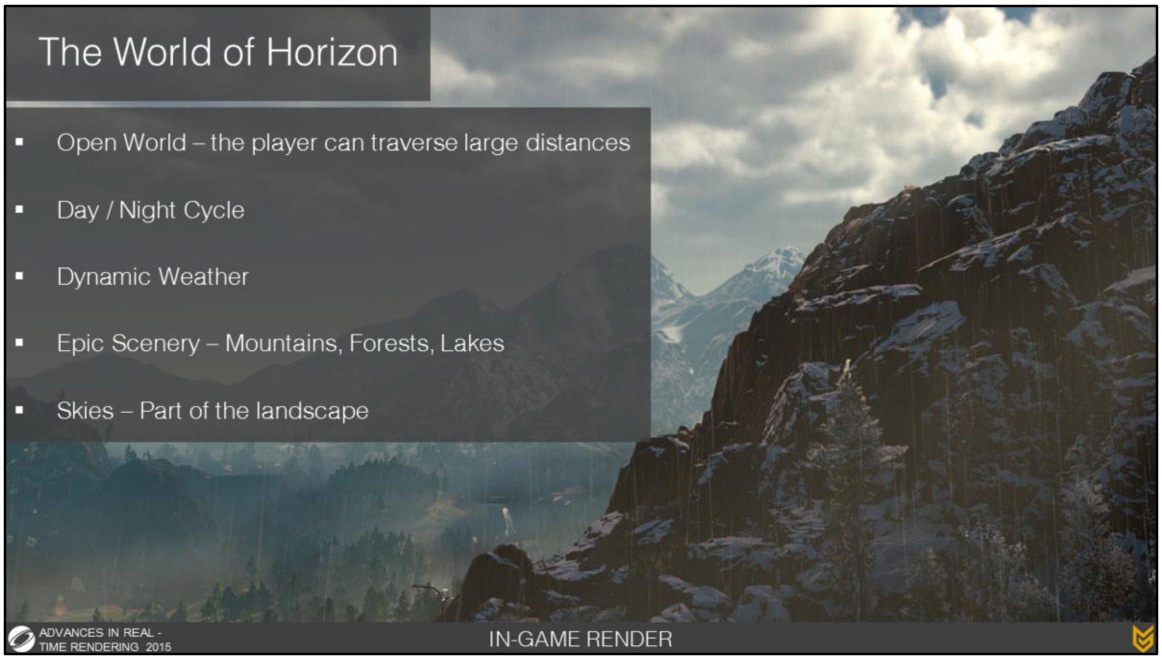
By animating these dome shaders we could create some pretty detailed and epic sky scapes for our games.

Horizon is a very different kind of game…

ADVANCES IN REAL - TIME RENDERING 2015

IN-GAME RENDER

Horizon trailer



The World of Horizon

- Open World – the player can traverse large distances
- Day / Night Cycle
- Dynamic Weather
- Epic Scenery – Mountains, Forests, Lakes
- Skies – Part of the landscape

ADVANCES IN REAL - TIME RENDERING 2015

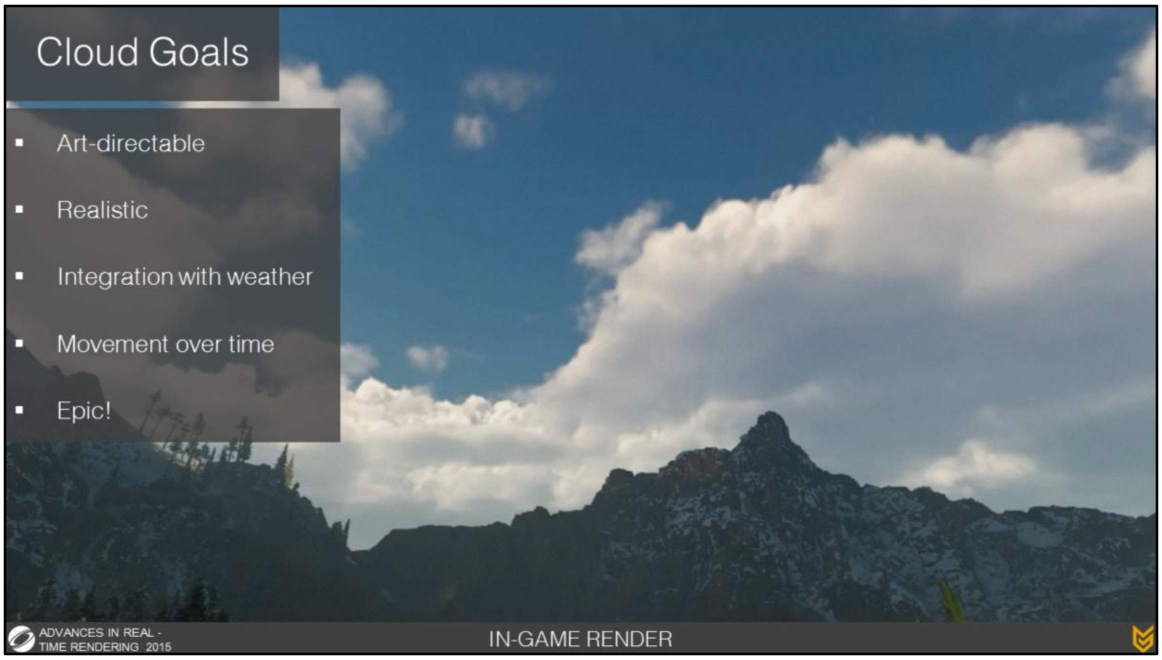
IN-GAME RENDER

So, from that you could see that we have left the world of Killzone behind.

- Horizon is a vastly open world where you can pretty much go anywhere that you see, including the tops of mountains.
- Since this is a living real world, we simulate the spinning of the earth by having a time of day cycle.
- Weather is part of the environment so it will be changing and evolving as well.
- There's lots of epic scenery: Mountains, forests, plains, and lakes.
- Skies are a big part of the landscape of horizon. They make up half of the screen. Skies are also are a very important part of storytelling as well as world building.

ADVANCES IN REAL - TIME RENDERING 2015
CONCEPT ART

They are used to tell us where we are, when we are, and they can also be used as thematic devices in storytelling.



Cloud Goals

- Art-directable
- Realistic
- Integration with weather
- Movement over time
- Epic!

ADVANCES IN REAL - TIME RENDERING 2015

IN-GAME RENDER

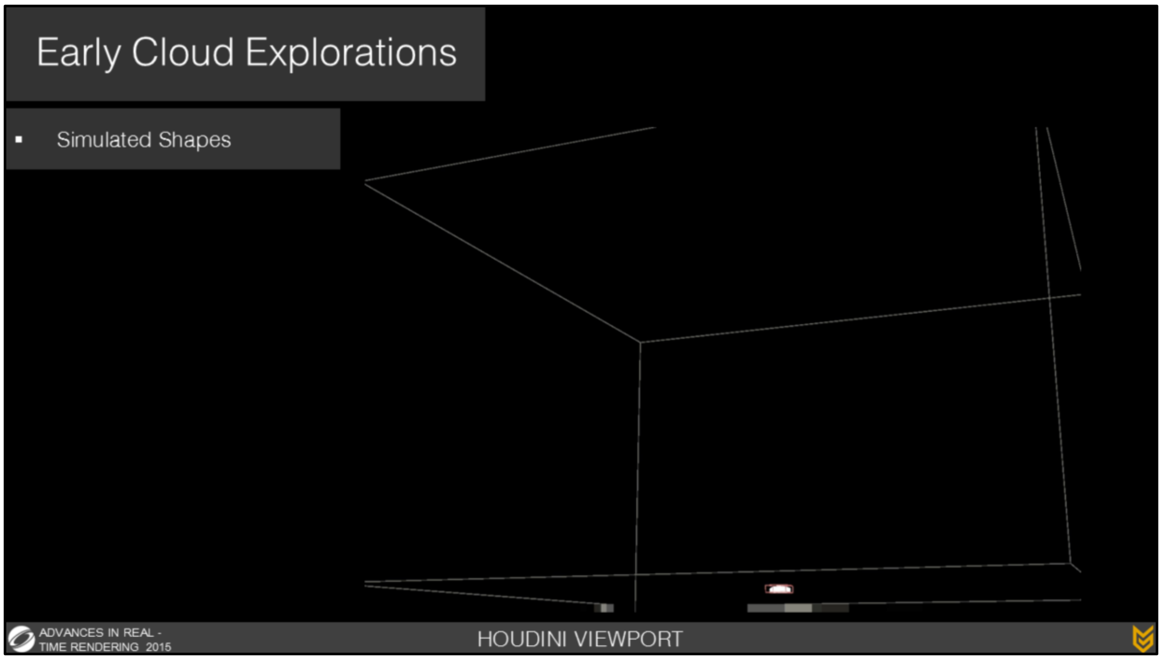
For Horizon, we want the player to really experience the world we are building. So we decided to try something bold. We prioritized some goals for our clouds.

- Art direct-able
- Realistic Representing multiple cloud types
- Integrate with weather
- Evolve in some way
- And of course, they needed to be Epic!

Early Cloud Explorations

ADVANCES IN REAL - TIME RENDERING 2015
IN-GAME RENDER

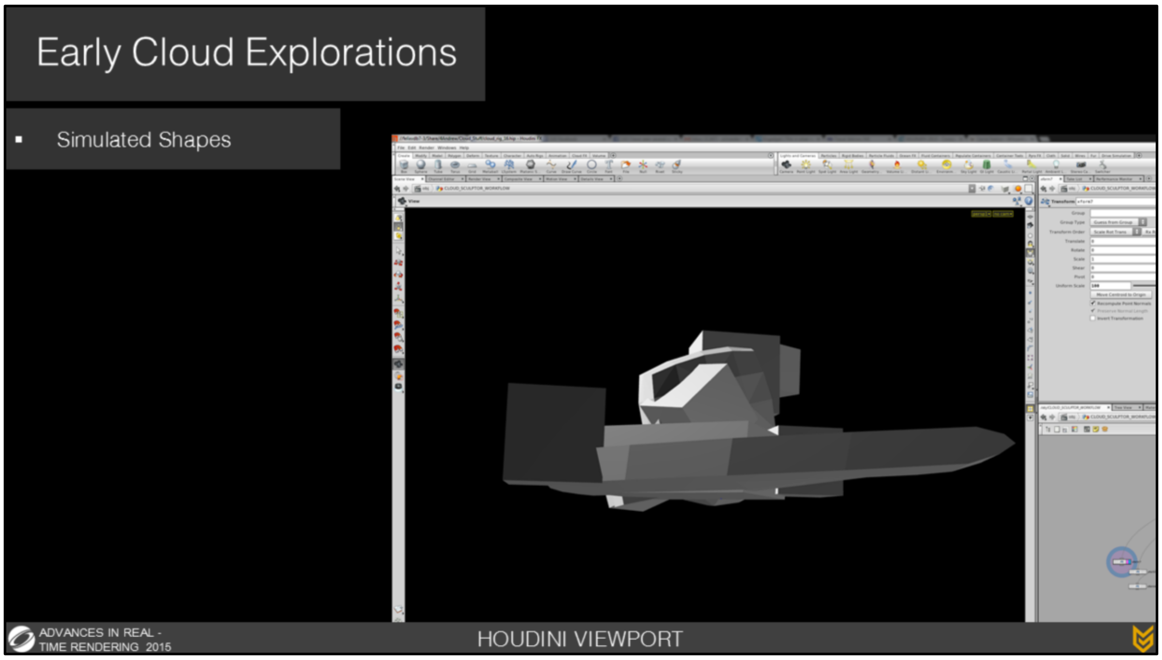
Realistic CG clouds are not an easy nut to crack. So, before we tried to solve the whole problem of creating a sky full them, we thought it would be good to explore different ways to make and light individual cloud assets.



Early Cloud Explorations

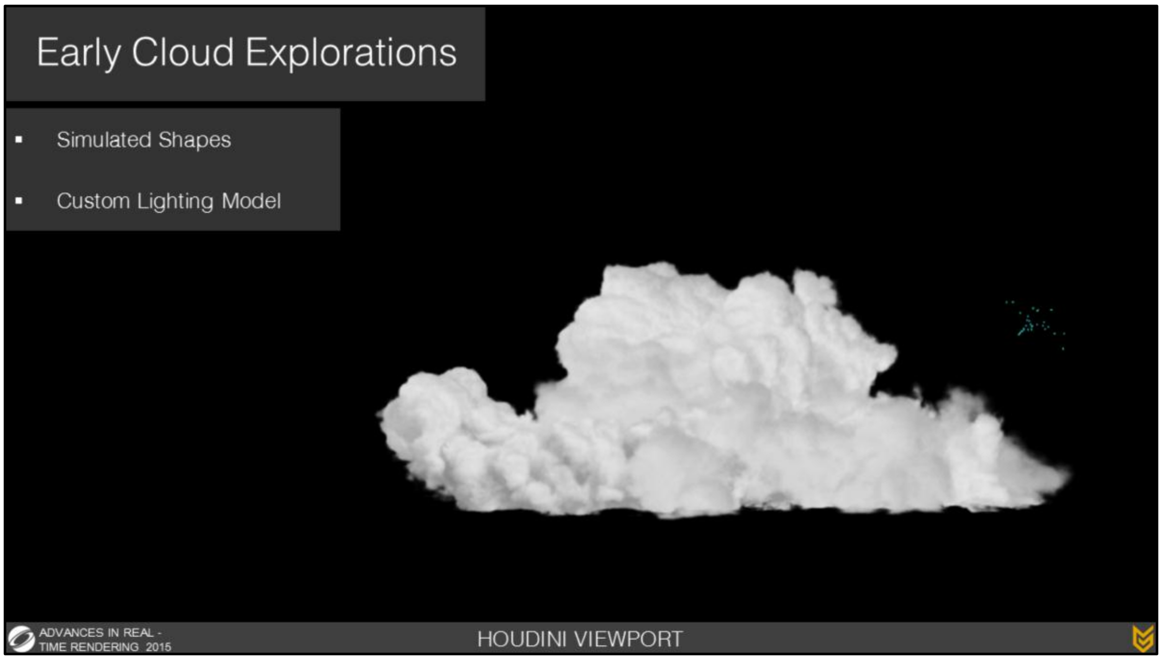
- Simulated Shapes

- Our earliest successful modeling approach was to use a custom fluid solver to grow clouds. The results were nice, but this was hard for artists to control if they had not had any fluid simulation experience. Guerrilla is a game studio after all.


Early Cloud Explorations

- Simulated Shapes

We ended up modeling clouds from simple shapes,
- voxelizing them and then …
- Running them through our fluid solver …
- Until we got a cloud like shape .



Early Cloud Explorations

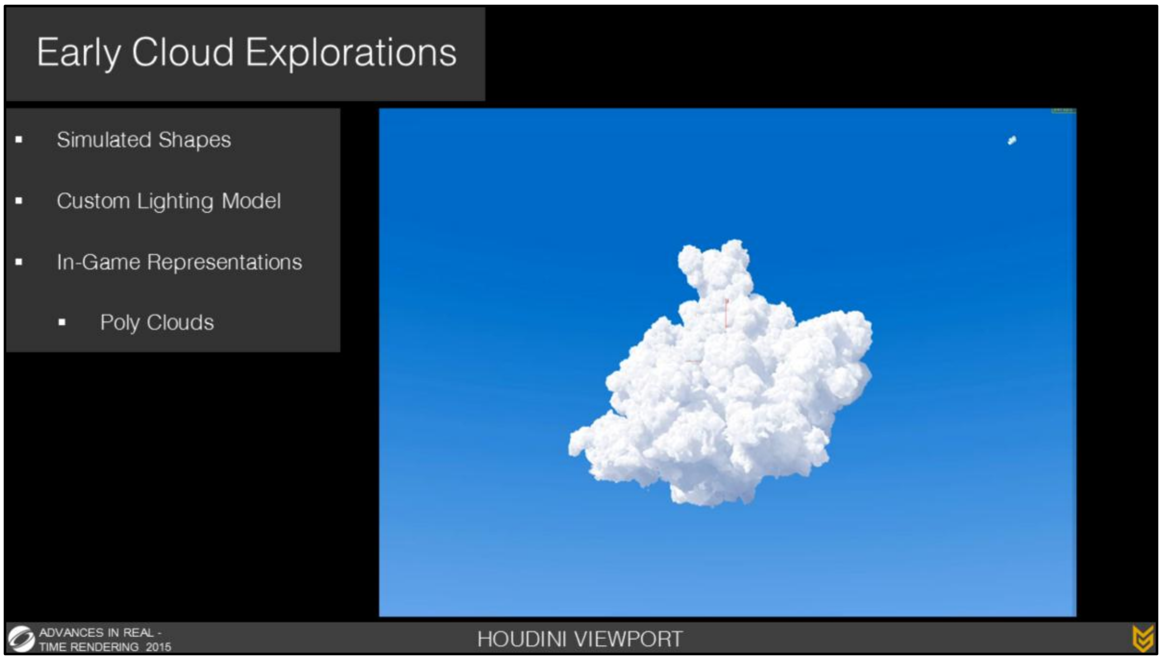
- Simulated Shapes
- Custom Lighting Model

ADVANCES IN REAL - TIME RENDERING 2015

HOUDINI VIEWPORT

And then we developed a lighting model that we used to pre-compute primary and secondary scattering,

- Ill get into our final lighting model a little later, but the result you see here is computed on the cpu in Houdini in 10 seconds.



Early Cloud Explorations

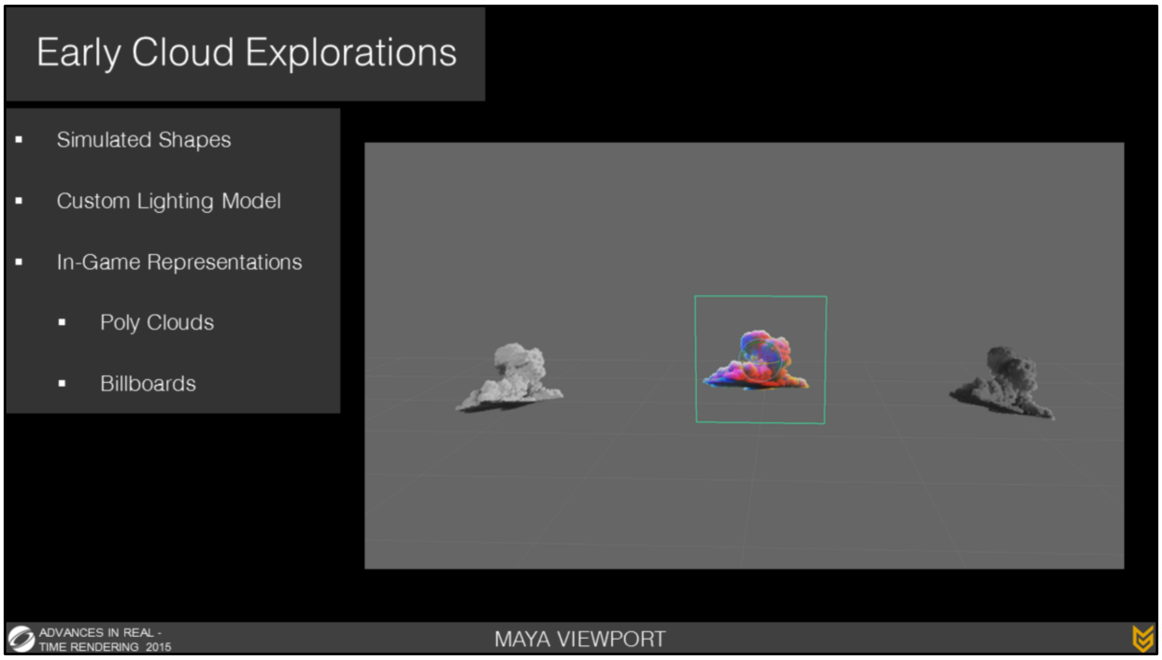
- Simulated Shapes
- Custom Lighting Model
- In-Game Representations
  - Poly Clouds

ADVANCES IN REAL - TIME RENDERING 2015

HOUDINI VIEWPORT

We explored 3 ways to get these cloud assets into game.

- For the first, we tried to treat our cloud as part of the landscape, literally modeling them as polygons from our fluid simulations and baking the lighting data using spherical harmonics. This only worked for the thick clouds and not whispy ones …



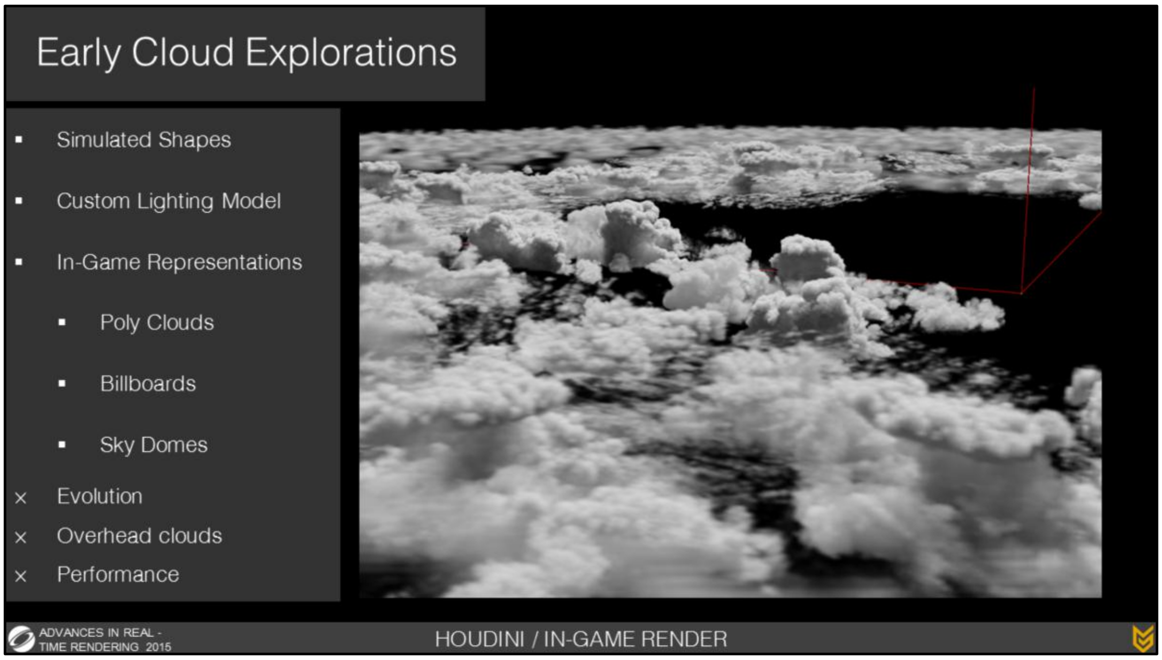
Early Cloud Explorations

- Simulated Shapes
- Custom Lighting Model
- In-Game Representations
  - Poly Clouds
  - Billboards

ADVANCES IN REAL - TIME RENDERING 2015

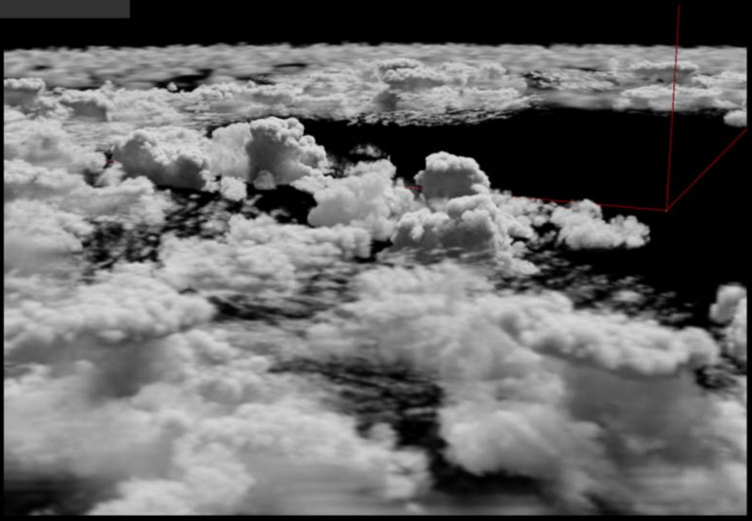
MAYA VIEWPORT

So, we though we should try to enhance the billboard approach to support multiple orientations and times of day . We succeeded but we found that we couldn't easily re-produce inter cloud shadowing. So…



Early Cloud Explorations

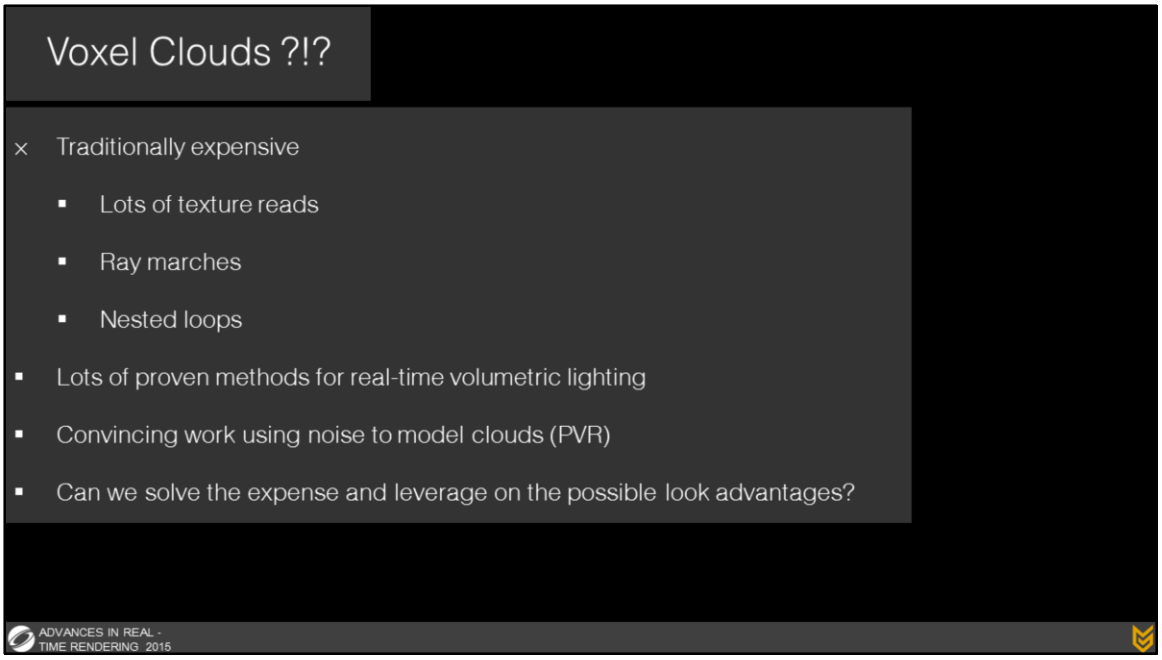
- Simulated Shapes
- Custom Lighting Model
- In-Game Representations
  - Poly Clouds
  - Billboards
  - Sky Domes
- × Evolution
- × Overhead clouds
- × Performance



ADVANCES IN REAL - TIME RENDERING 2015

HOUDINI / IN-GAME RENDER

- We tried rendering all of our voxel clouds as one cloud set to produce skydomes that could also blend into the atmosphere over depth. Sort of worked.
- At this point we took a step back to evaluate what didn't work. None of the solutions made the clouds evolve over time. There was not a good way to make clouds pass overhead. And there was high memory usage and overdraw for all methods.
- So maybe a traditional asset based approach was not the way to go.



Voxel Clouds ?!?

× Traditionally expensive
  - Lots of texture reads
  - Ray marches
  - Nested loops
- Lots of proven methods for real-time volumetric lighting
- Convincing work using noise to model clouds (PVR)
- Can we solve the expense and leverage on the possible look advantages?

ADVANCES IN REAL - TIME RENDERING 2015

Well, What about voxel clouds?
OK we are crazy we are actually considering voxel clouds now…
As you can imagine this idea was not very popular with the programmers.

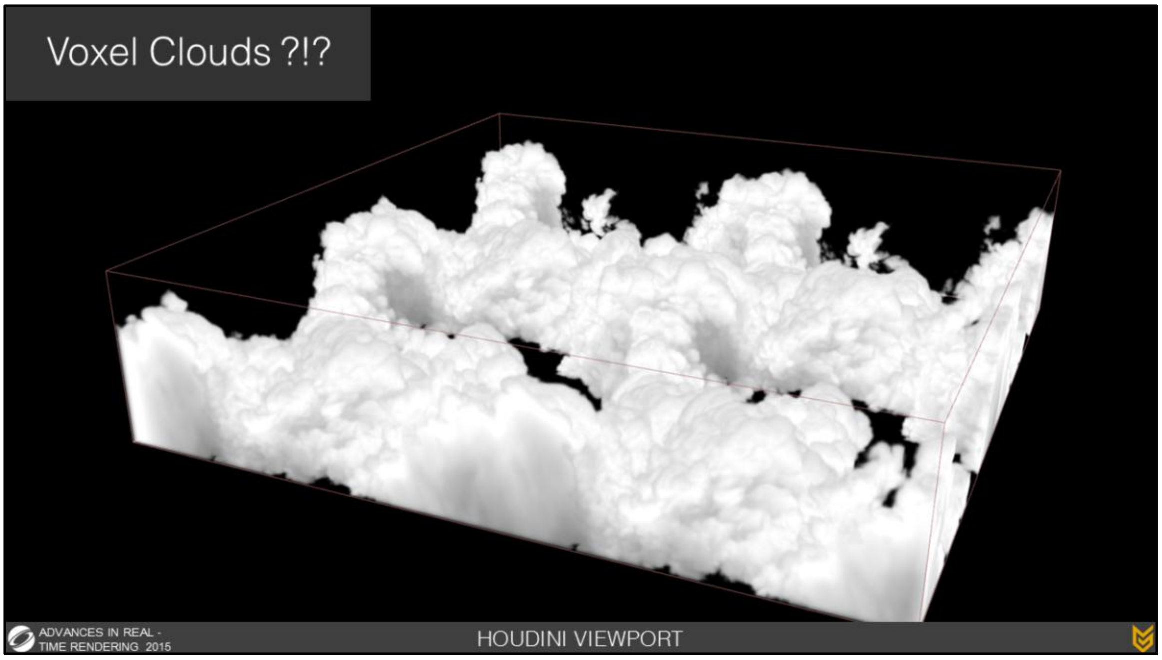
- Volumetrics are traditionally very expensive
- With lots of texture reads
- Ray marches
- Nested loops
- However, there are many proven methods for fast, believable volumetric lighting
- There is convincing work to use noise to model clouds . I can refer you to the 2012 Production Volume Rendering course.
- Could we solve the expense somehow and benefit from all of the look advantages of volumetrics?

Voxel Clouds ?!?

ADVANCES IN REAL - TIME RENDERING 2015

IN-GAME RENDER

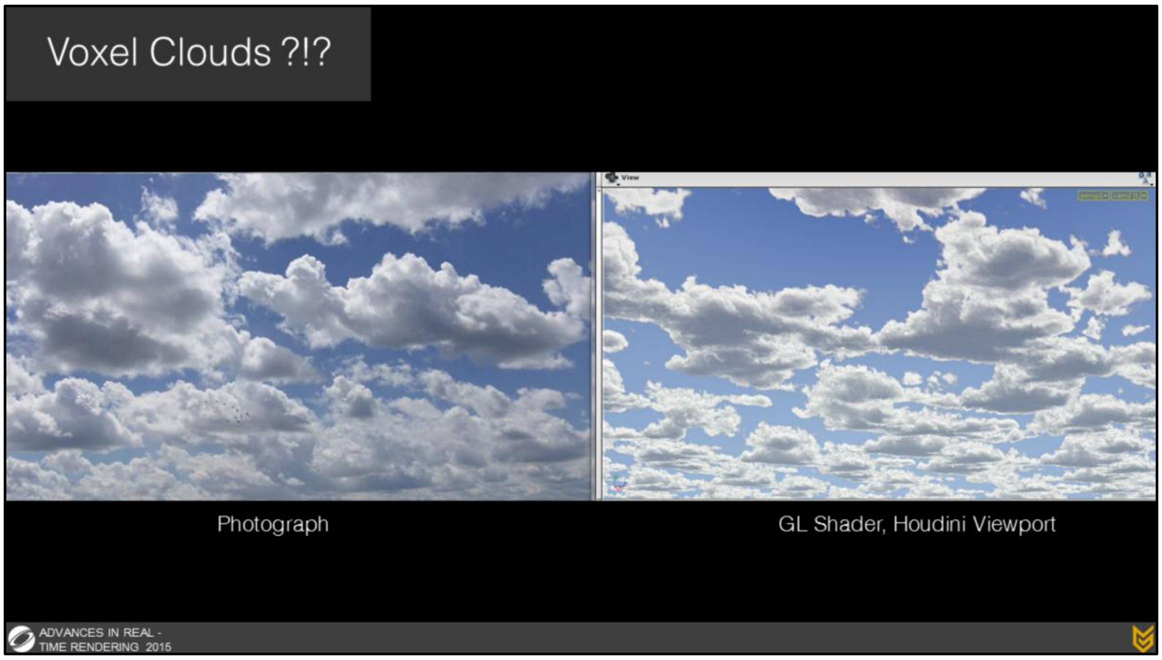
Our first test was to stack up a bunch of polygons in front of the camera and sample 3d Perlin noise with them. While extremely slow, This was promising, but we want to represent multiple clouds types not just these bandy clouds.


Voxel Clouds ?!?

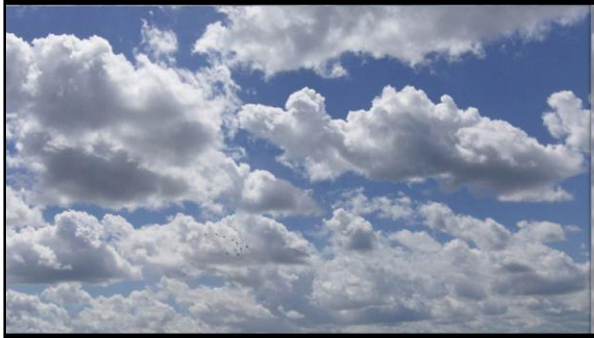
ADVANCES IN REAL - TIME RENDERING 2015

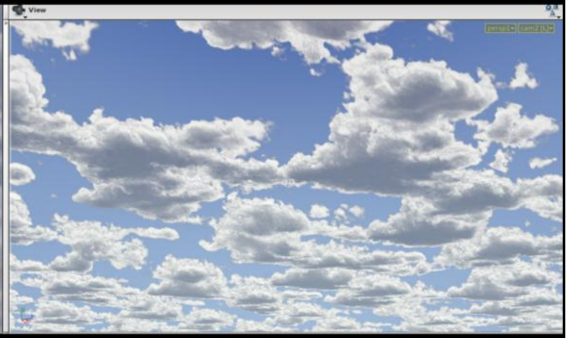
HOUDINI VIEWPORT

So we went into Houdini and generated some tiling 3d textures out of the simulated cloud shapes. Using Houdini's GL extensions, we built a prototype GL shader to develop a cloud system and lighting model.



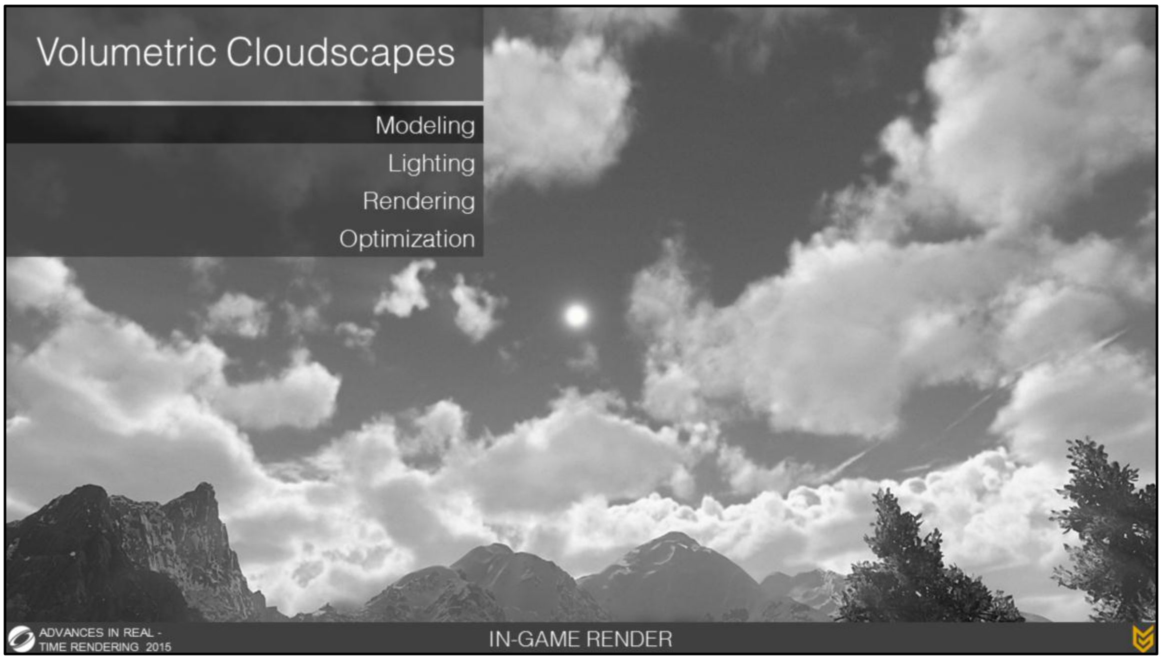
Voxel Clouds ?!?


Photograph


GL Shader, Houdini Viewport

ADVANCES IN REAL - TIME RENDERING 2015

In The end, with a LOT of hacks, we got very close to mimicking our reference. However, it all fell apart when we put the clouds in motion. It also took 1 second per frame to compute. For me coming from animated vfx, this was pretty impressive, but my colleagues were still not impressed.
So I thought, Instead of explicitly defining clouds with pre determined shapes, what if we could develop some good noises at lower resolutions that have the characteristics we like and then find a way to blend between them based on a set of rules. There has been previous work like this but none of it came close to our look goals.

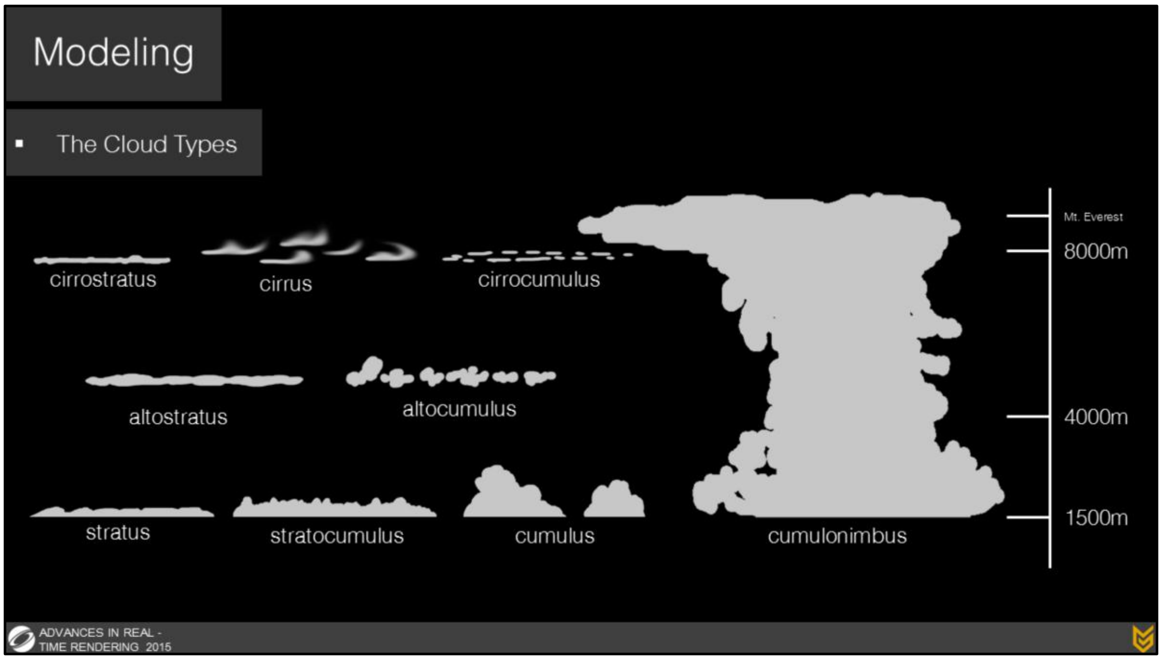


Volumetric Cloudscapes

- Modeling
- Lighting
- Rendering
- Optimization

This brings us to the clouds system for horizon. To explain it better I have broken it down into 4 sections: Modeling, Lighting, Rendering and Optimization.

Before I get into how we modeled the cloud scapes, it would be good to have a basic understanding of what clouds are and how they evolve into different shapes.

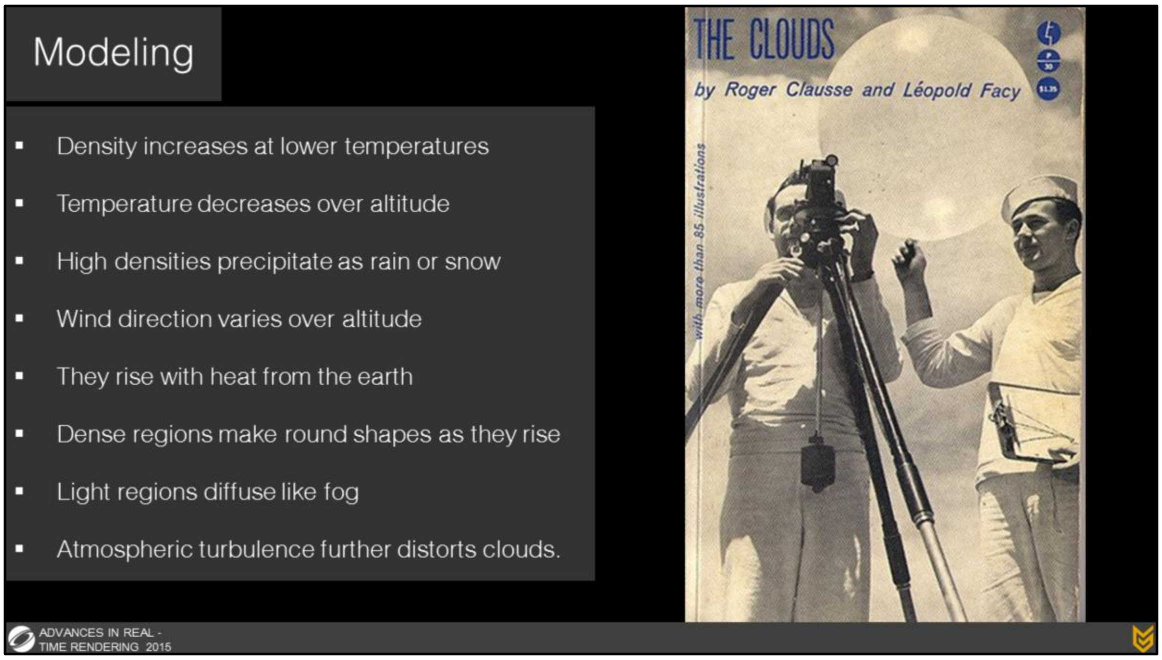


Modeling

- The Cloud Types

Classifying clouds helped us better communicate what we were talking about and

- Define where we would draw them. The basic cloud types are as follows.
- The strato clouds including stratus, cumulus and stratocumulus
- The alto clouds, which are those bandy or puffy clouds above the strato layer
- And the cirro clouds those big arcing bands and little puffs in the upper atmosphere.
- Finally there is the granddaddy of all cloud types, the Cumulonimbus clouds which go high into the atmosphere.
- For comparison, mount Everest is above 8,000 meters.



Modeling

- Density increases at lower temperatures
- Temperature decreases over altitude
- High densities precipitate as rain or snow
- Wind direction varies over altitude
- They rise with heat from the earth
- Dense regions make round shapes as they rise
- Light regions diffuse like fog
- Atmospheric turbulence further distorts clouds.

THE CLOUDS
by Roger Clausse and Léopold Facy
with more than 85 illustrations
$1.35

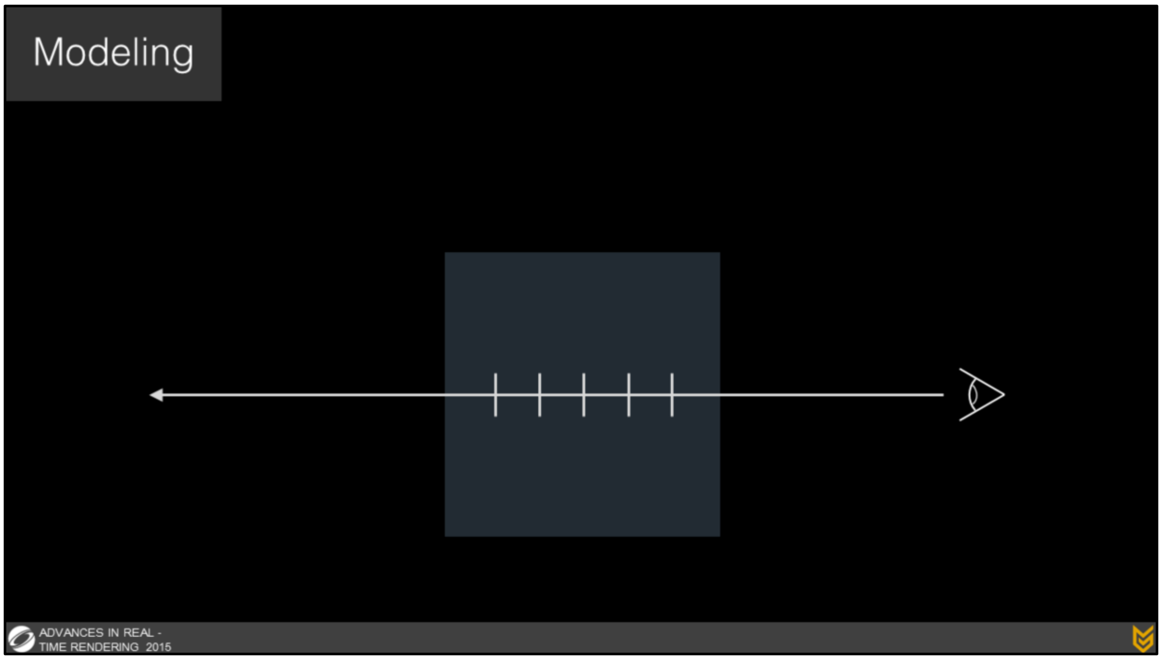
ADVANCES IN REAL - TIME RENDERING 2015

After doing research on cloud types, we had a look into the forces that shape them. The best source we had was a book from 1961 by two meteorologists, called “The Clouds” as creatively as research books from the 60’s were titled. What it lacked in charm it made up for with useful empirical results and concepts that help with modeling a cloud system.

- Density increases at lower temperatures
- Temperature decreases over altitude
- High densities precipitate as rain or snow
- Wind direction varies over altitude
- They rise with heat from the earth
- Dense regions make round shapes as they rise

- Light regions diffuse like fog
- Atmospheric turbulence further distorts clouds.

These are all abstractions that are useful when modeling clouds.

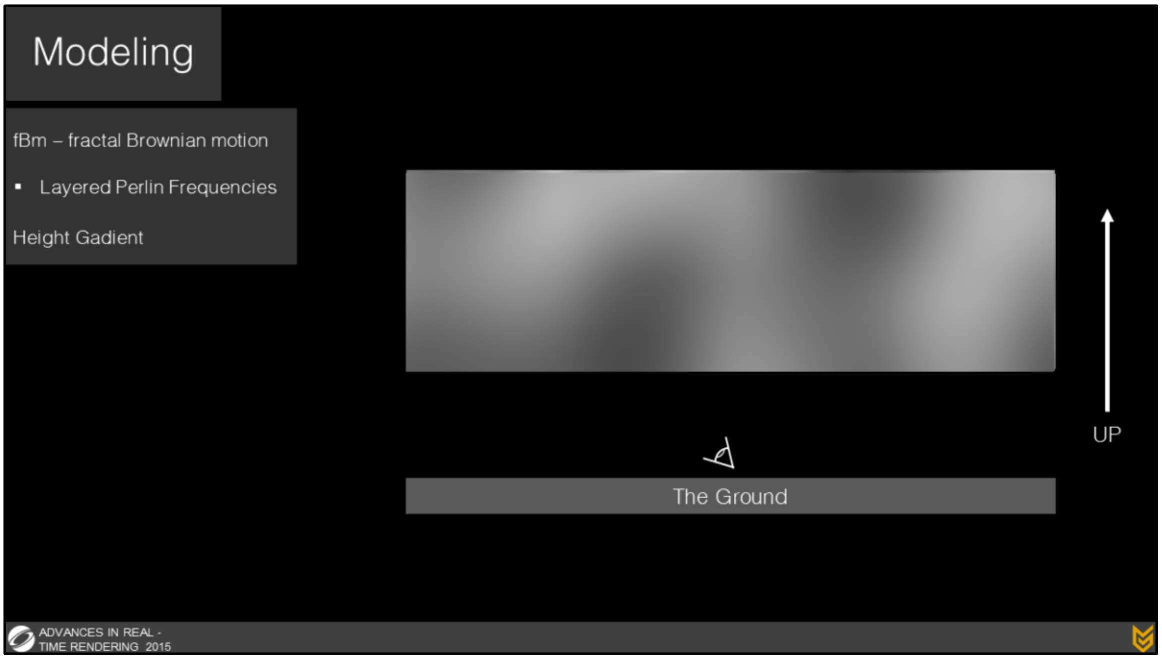

Modeling

Our modeling approach uses ray marching to produce clouds.

- We march from the camera and sample noises and a set of gradients to define our cloud shapes using a sampler

In a ray march you use a sampler to…

- Build up an alpha channel….
- And calculate lighting



Modeling

fBm – fractal Brownian motion

- Layered Perlin Frequencies

Height Gadient

There are many examples of real-time volume clouds on the internet. The usual approach involves drawing them in a height zone above the camera using something called fBm, Fractal Brownian Motion. This is done by layering Perlin noises of different frequencies until you get something detailed.

•

•

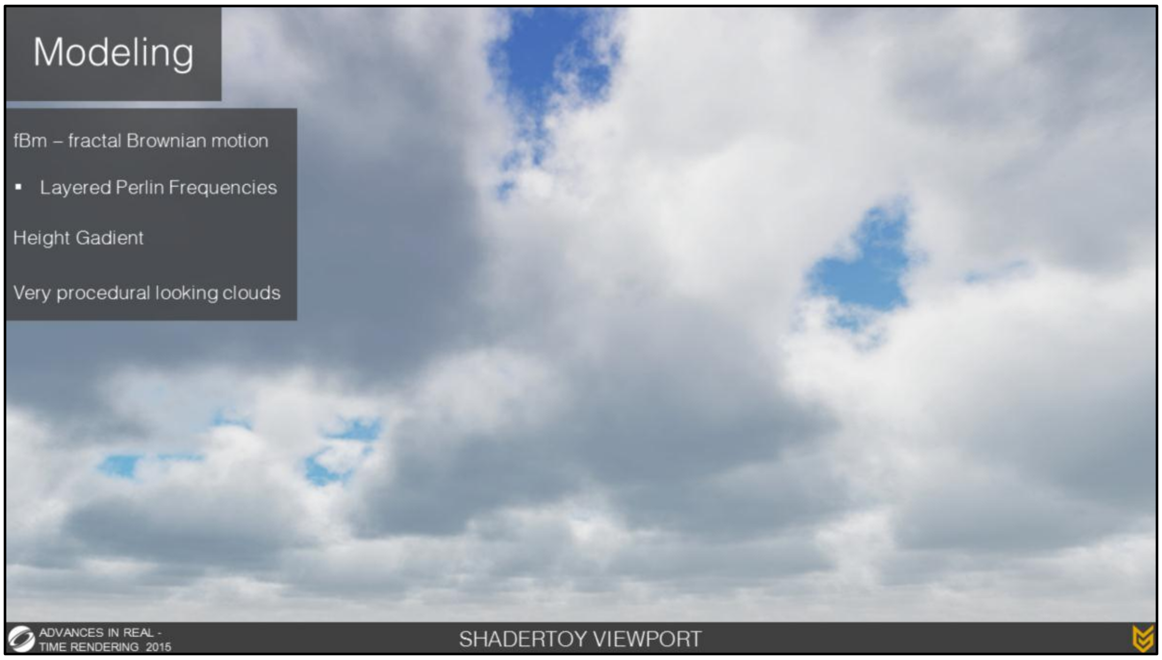
•

•

•

(pause)

- This noise is then usually combined somehow with a gradient to define a change in cloud density over height.



Modeling

fBm – fractal Brownian motion

- Layered Perlin Frequencies

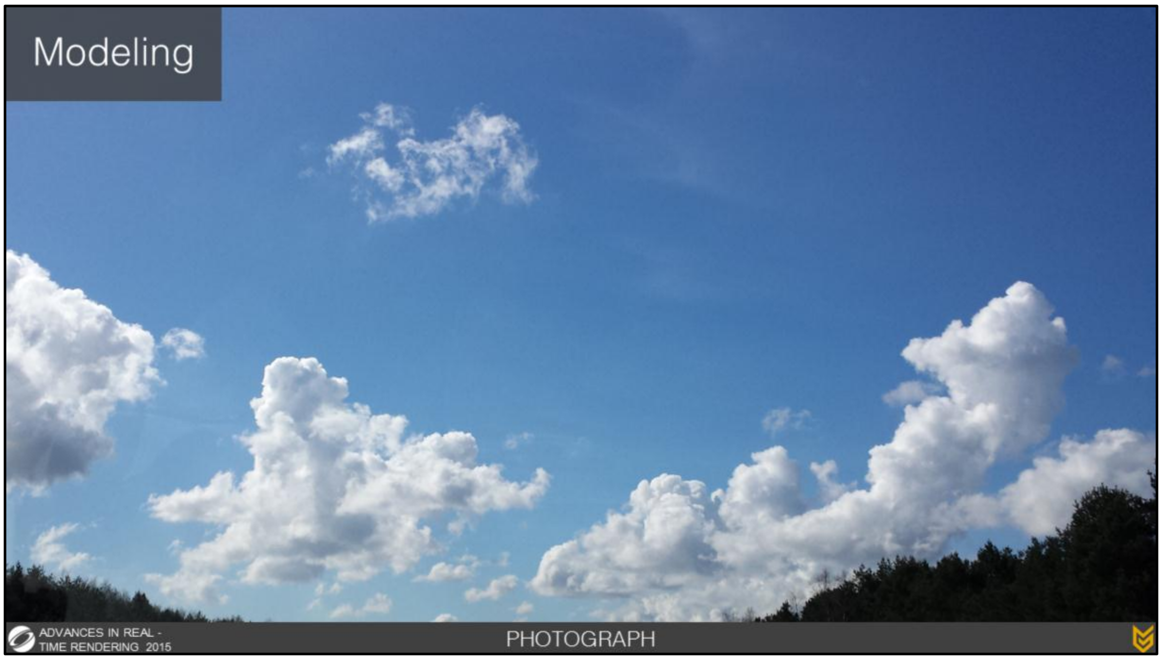
Height Gadient

Very procedural looking clouds

ADVANCES IN REAL - TIME RENDERING 2015

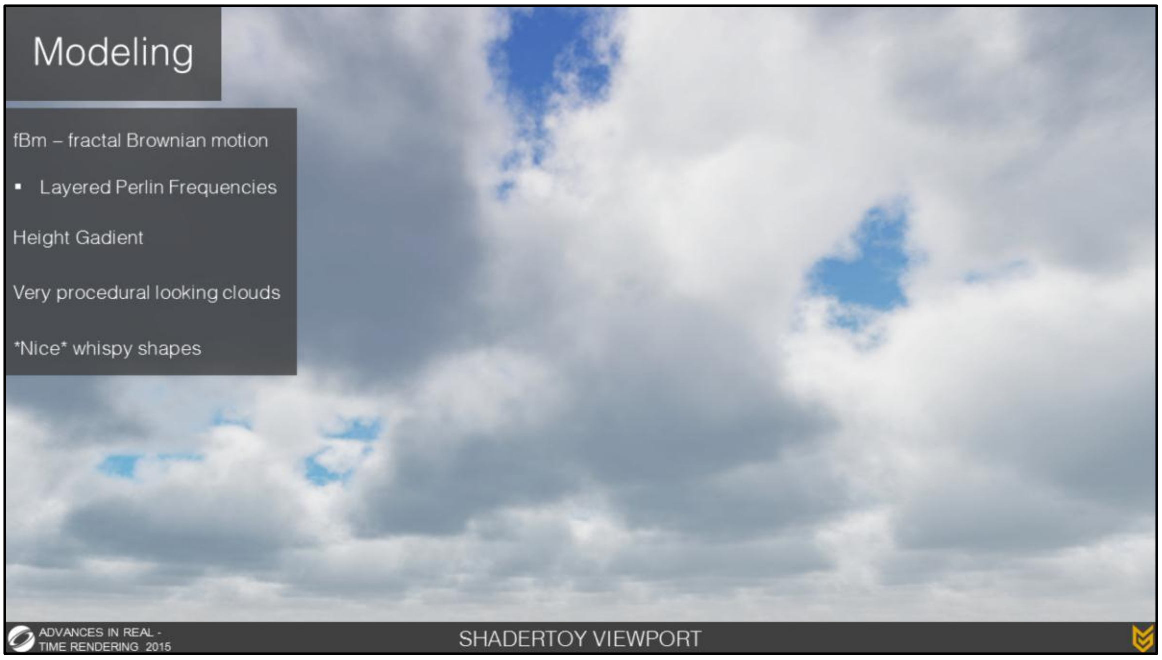
SHADERTOY VIEWPORT

This makes some very nice but very procedural looking clouds. What’s wrong? There are no larger governing shapes or visual cues as to what is actually going on here. We don’t feel the implied evolution of the clouds from their shapes.



Modeling

By contrast, in this photograph we can tell what is going on here. These clouds are rising like puffs of steam from a factory. Notice the round shapes at the tops and whispy shapes at the bottoms.



Modeling

fBm – fractal Brownian motion

- Layered Perlin Frequencies

Height Gadient

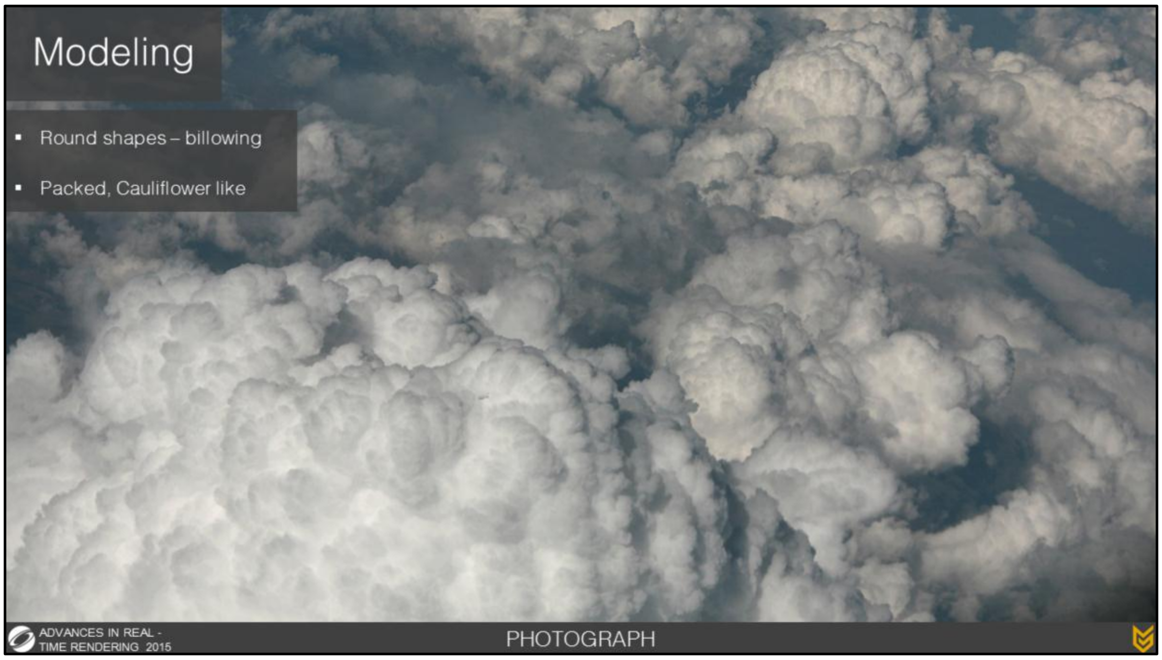
Very procedural looking clouds

Nice whispy shapes

ADVANCES IN REAL - TIME RENDERING 2015

SHADERTOY VIEWPORT

This fBm approach has some nice whispy shapes, but it lacks those bulges and billows that give a sense of motion. We need to take our shader beyond what you would find on something like ShaderToy.



Modeling

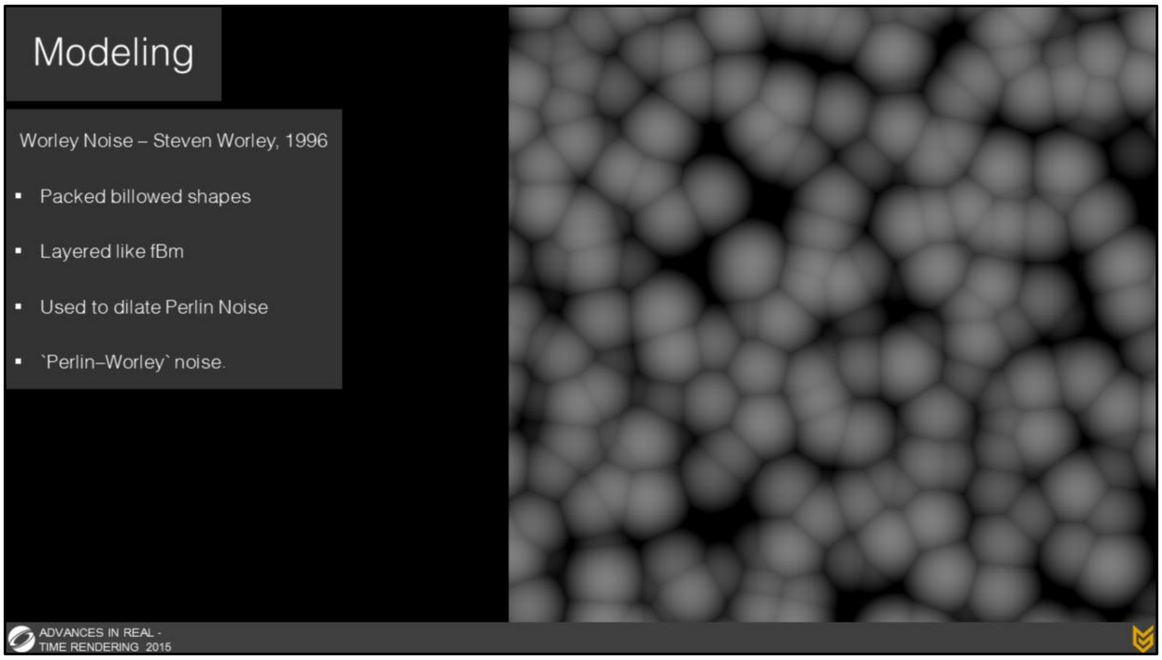
- Round shapes – billowing
- Packed, Cauliflower like

ADVANCES IN REAL - TIME RENDERING 2015

PHOTOGRAPH

- These billows, as Ill call them…
- …are packed, sometimes taking on a cauliflower shape.

Since Perlin noise alone doesn’t cut it, we developed our own layered noises.



Modeling

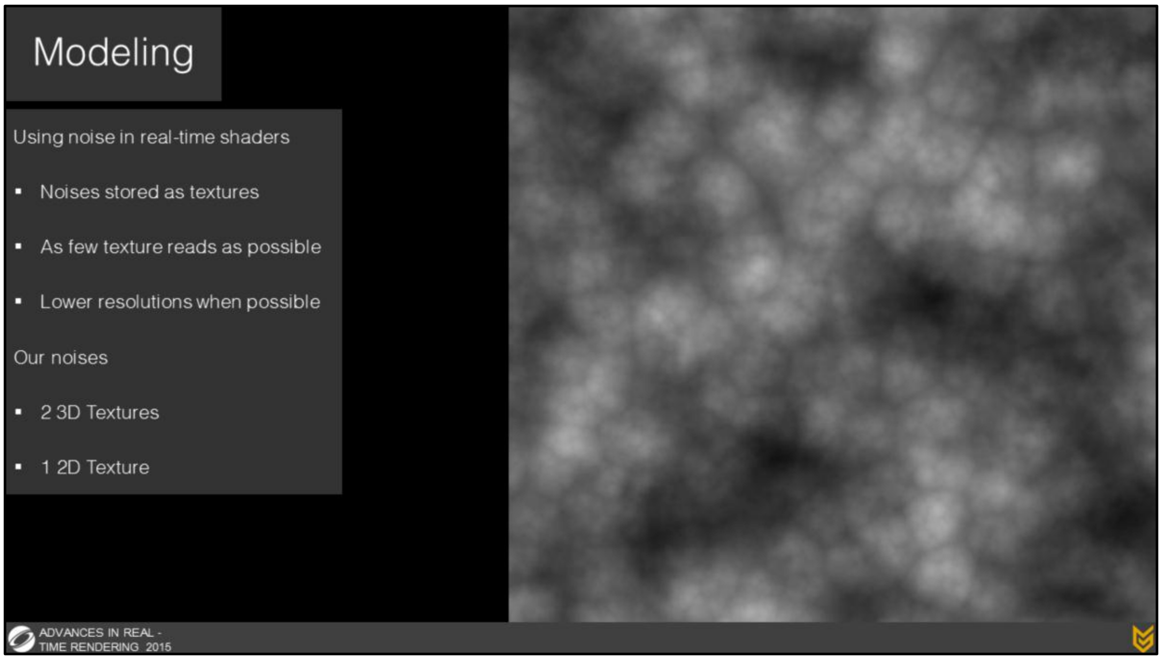
Worley Noise – Steven Worley, 1996

- Packed billowed shapes
- Layered like fBm
- Used to dilate Perlin Noise
- `Perlin–Worley` noise.

ADVANCES IN REAL - TIME RENDERING 2015

Worley noise was introduced in 1996 by Steven Worley and is often used for caustics and water effects. If it is inverted as you see here:

- It makes tightly packed billow shapes.
- We layered it like the standard Perlin fBm approach
-
-
- Then we used it as an offset to dilate Perlin noise. this allowed us to keep the connectedness of Perlin noise but add some billowy shapes to it.
- We referred to this as `Perlin-Worley` noise



Modeling

Using noise in real-time shaders

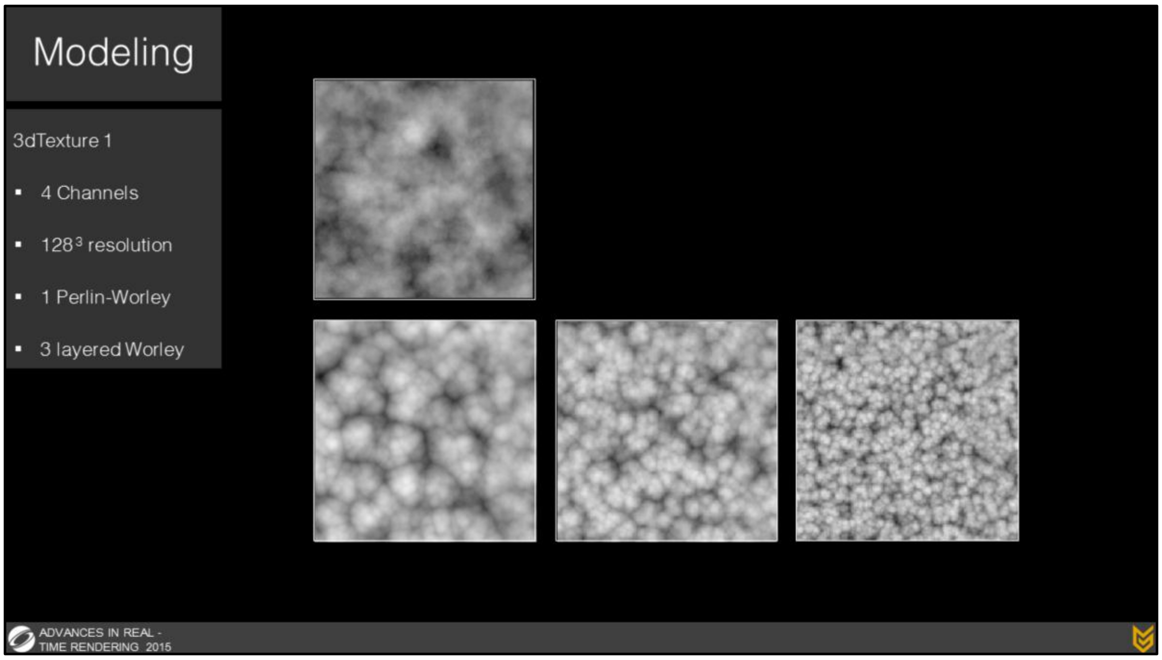
- Noises stored as textures
- As few texture reads as possible
- Lower resolutions when possible

Our noises

- 2 3D Textures
- 1 2D Texture

ADVANCES IN REAL-TIME RENDERING 2015

- In games, it is often best for performance to store noises as tiling 3d textures.
- You want to keep texture reads to a minimum…
- and keep the resolutions as small as possible.
- In our case we have compressed our noises to…
- two 3d textures…
- And 1 2d texture.

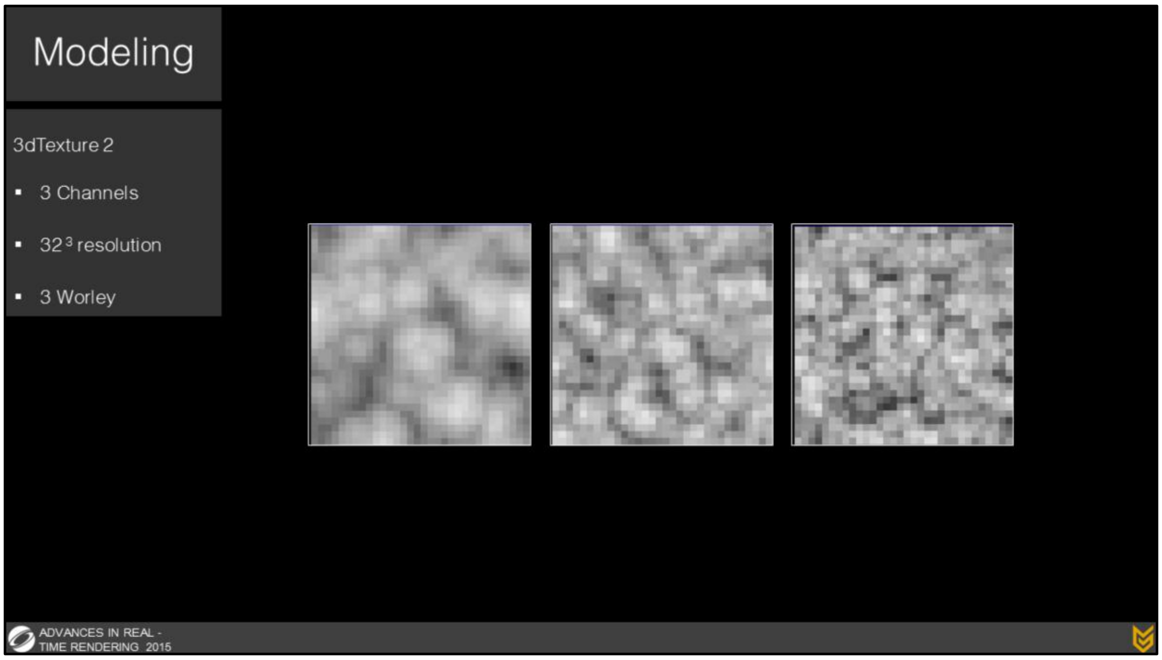


Modeling

3dTexture 1

- 4 Channels
- 128^3 resolution
- 1 Perlin-Worley
- 3 layered Worley

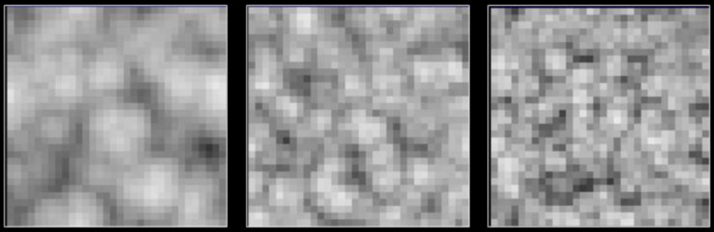
- The first 3d Texture…
- has 4 channels…
- it is 128^3 resolution…
- The first channel is the Perlin - Worley noise I just described.
- The other 3 are Worley noise at increasing frequencies. Like in the standard approach, This 3d texture is used to define the base shape for our clouds.



Modeling

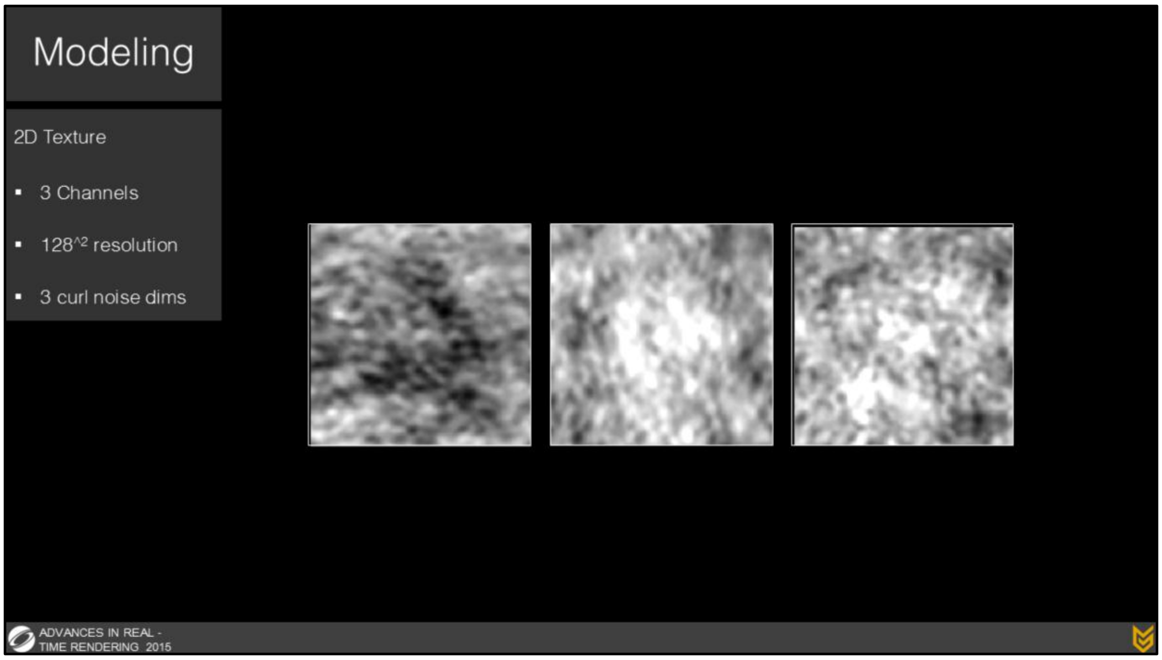
3dTexture 2

- 3 Channels
- 32^3 resolution
- 3 Worley



ADVANCES IN REAL - TIME RENDERING 2015

- Our second 3d texture…
- has 3 channels…
- it is 32^3 resolution…
- and uses Worley noise at increasing frequencies. This texture is used to add detail to the base cloud shape defined by the first 3d noise.



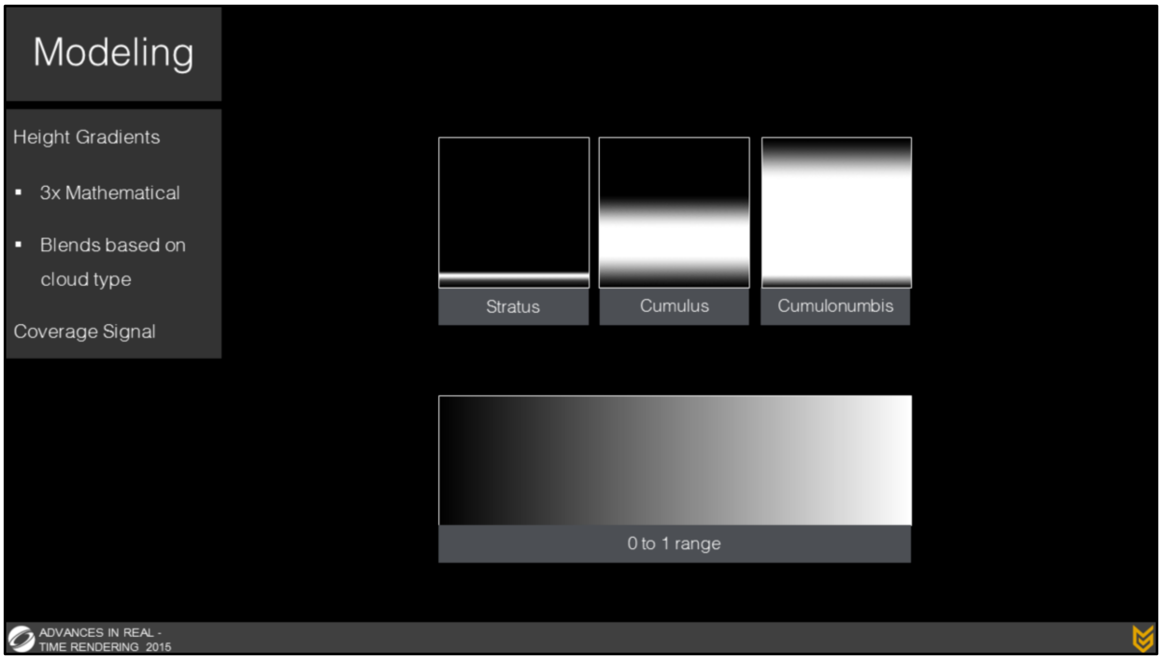
Modeling

2D Texture

- 3 Channels
- 128^2 resolution
- 3 curl noise dims

ADVANCES IN REAL - TIME RENDERING 2015

- Our 2D texture…
- has 3 channels…
- it is 128^2 resolution…
- and uses curl noise. Which is non divergent and is used to fake fluid motion. We use this noise to distort our cloud shapes and add a sense of turbulence.



Modeling

Height Gradients

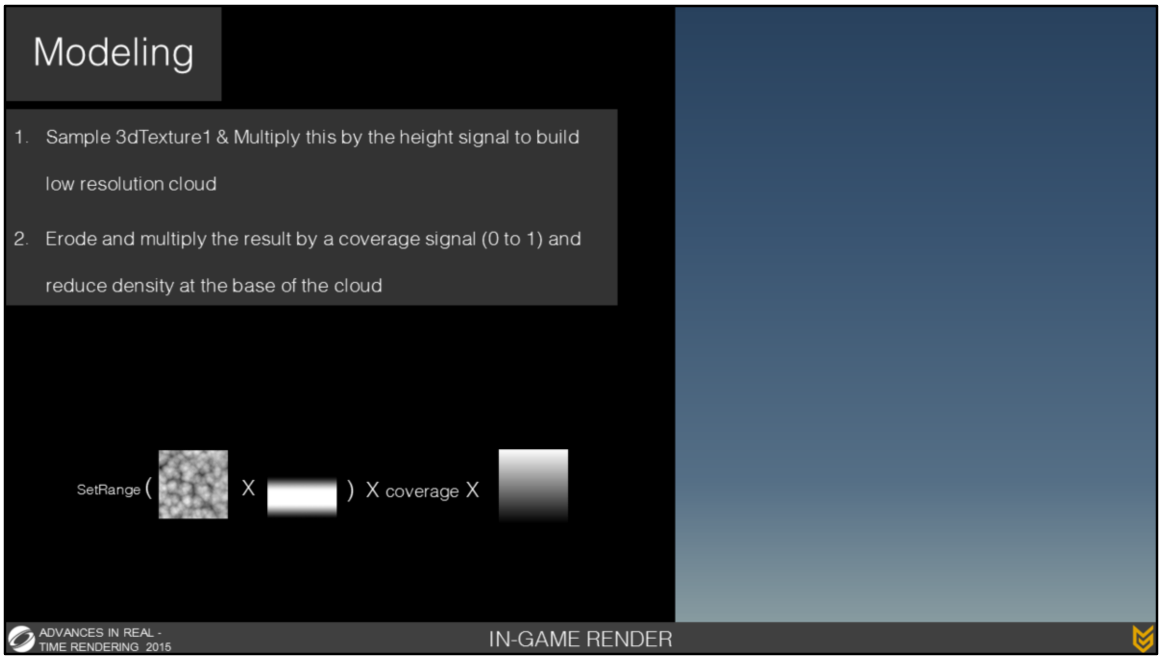
- 3x Mathematical
- Blends based on cloud type

Coverage Signal

Stratus
Cumulus
Cumulonumbis
0 to 1 range

ADVANCES IN REAL - TIME RENDERING 2015

- Recall that the standard solution calls for a height gradient to change the noise signal over altitude. Instead of 1, we use…
- 3 mathematical presets that represent the major low altitude…
- cloud types when we blend between them at the sample position.
- We also have a value telling us how much cloud coverage we want to have at the sample position. This is a value between zero and 1.



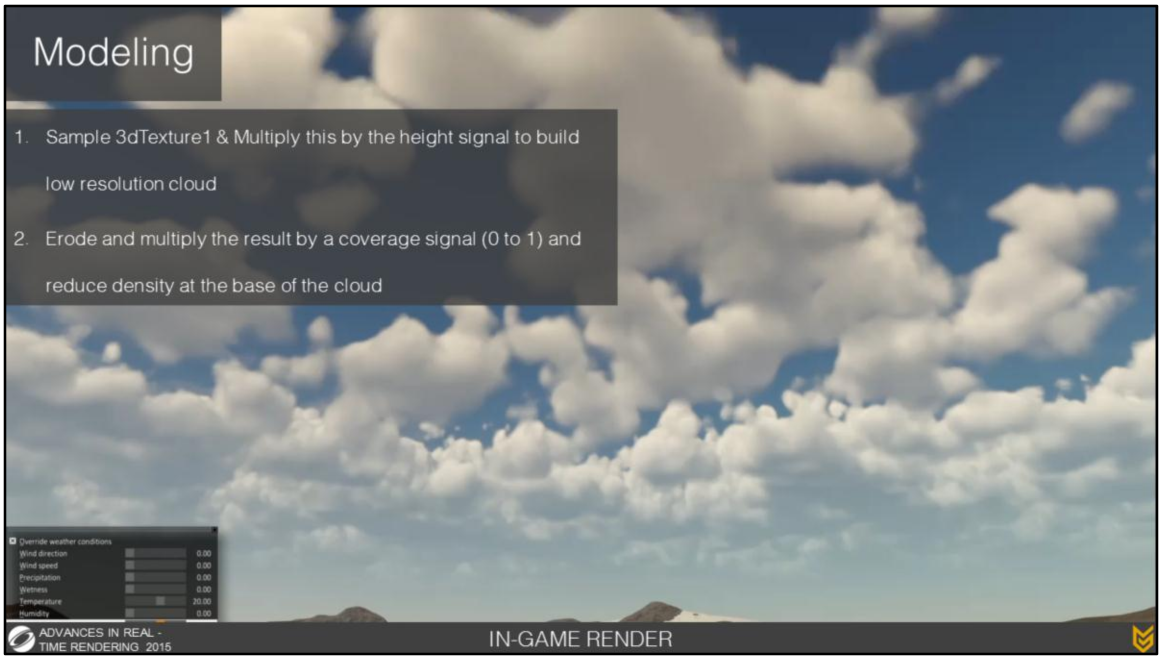
Modeling

1. Sample 3dTexture1 & Multiply this by the height signal to build low resolution cloud
2. Erode and multiply the result by a coverage signal (0 to 1) and reduce density at the base of the cloud

SetRange (
X
) X coverage X
ADVANCES IN REAL - TIME RENDERING 2015
IN-GAME RENDER

What we are looking at on the right side of the screen is a view rotated about 30 degrees above the horizon. We will be drawing clouds per the standard approach in a zone above the camera.

- First, we build a basic cloud shape by sampling our first 3dTexture and multiplying it by our height signal.
- The next step is to multiply the result by the coverage and reduce density at the bottoms of the clouds.



Modeling

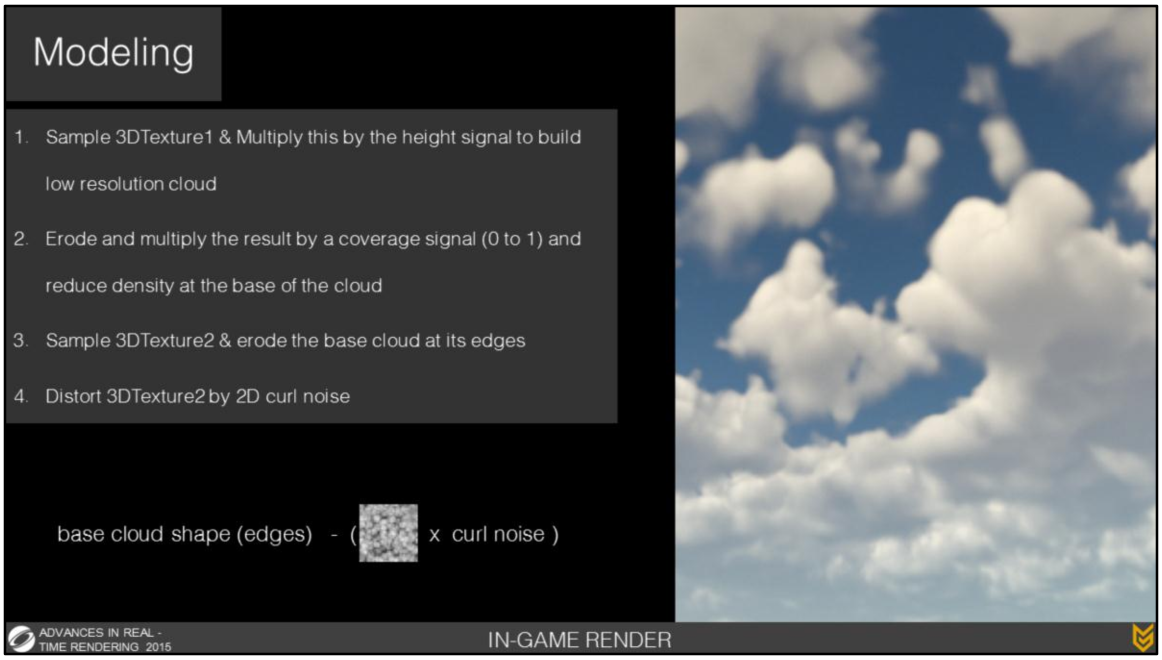
1. Sample 3dTexture1 & Multiply this by the height signal to build low resolution cloud
2. Erode and multiply the result by a coverage signal (0 to 1) and reduce density at the base of the cloud

```
☒ Override weather conditions
Wind direction    0.00
Wind speed        0.00
Precipitation     0.00
Wetness           0.00
Temperature      20.00
Humidity          0.00
```

ADVANCES IN REAL - TIME RENDERING 2015

IN-GAME RENDER

This ensures that the bottoms will be whispy and it increases the presence of clouds in a more natural way. Remember that density increases over altitude. Now that we have our base cloud shape, we add details.



Modeling

1. Sample 3DTexture1 & Multiply this by the height signal to build low resolution cloud
2. Erode and multiply the result by a coverage signal (0 to 1) and reduce density at the base of the cloud
3. Sample 3DTexture2 & erode the base cloud at its edges
4. Distort 3DTexture2 by 2D curl noise

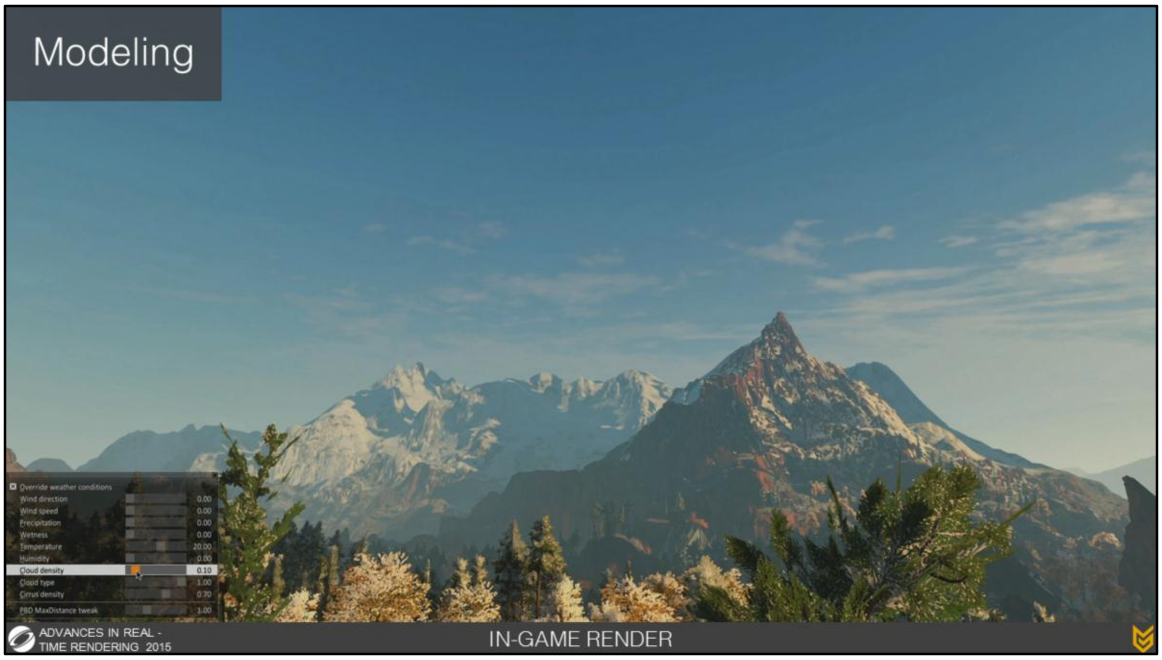
base cloud shape (edges) - (x curl noise)

ADVANCES IN REAL - TIME RENDERING 2015

IN-GAME RENDER

The next step is to…

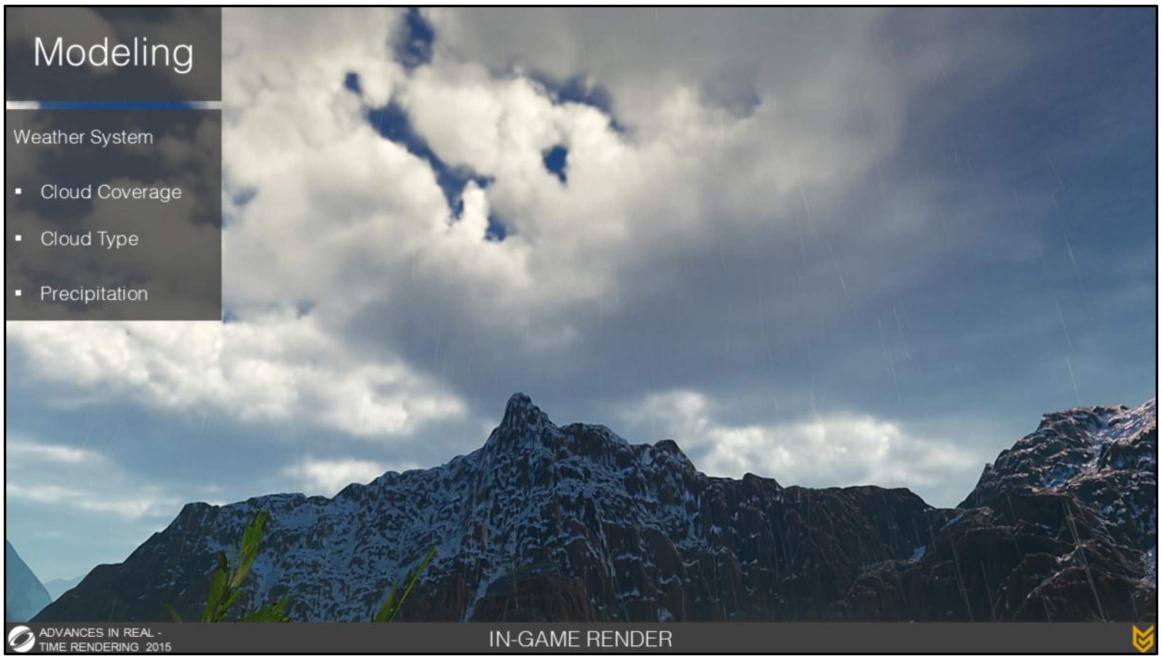
- erode the base cloud shape by subtracting the second 3d texture at the edges of the cloud. Little tip, If you invert the Worley noise at the base of the clouds you get some nice whispy shapes.
- We also distort this second noise texture by our 2d curl noise to fake the swirly distortions from atmospheric turbulence as you can see here…
-


Modeling

```
☒ Override weather conditions
  Wind direction          0.00
  Wind speed              0.00
  Precipitation           0.00
  Wetness                 0.00
  Temperature            20.00
  Humidity                0.00
  Cloud density           0.10
  Cloud type              1.00
  Cirrus density          0.70
  PBD MaxDistance tweak   1.00
```

Here's that it looks like in game. I'm adjusting the coverage signal to make them thicker and then transitioning between the height gradients for cumulus to stratus.

Now that we have decent stationary clouds we need to start working on making them evolve as part of our weather system.



Modeling

Weather System

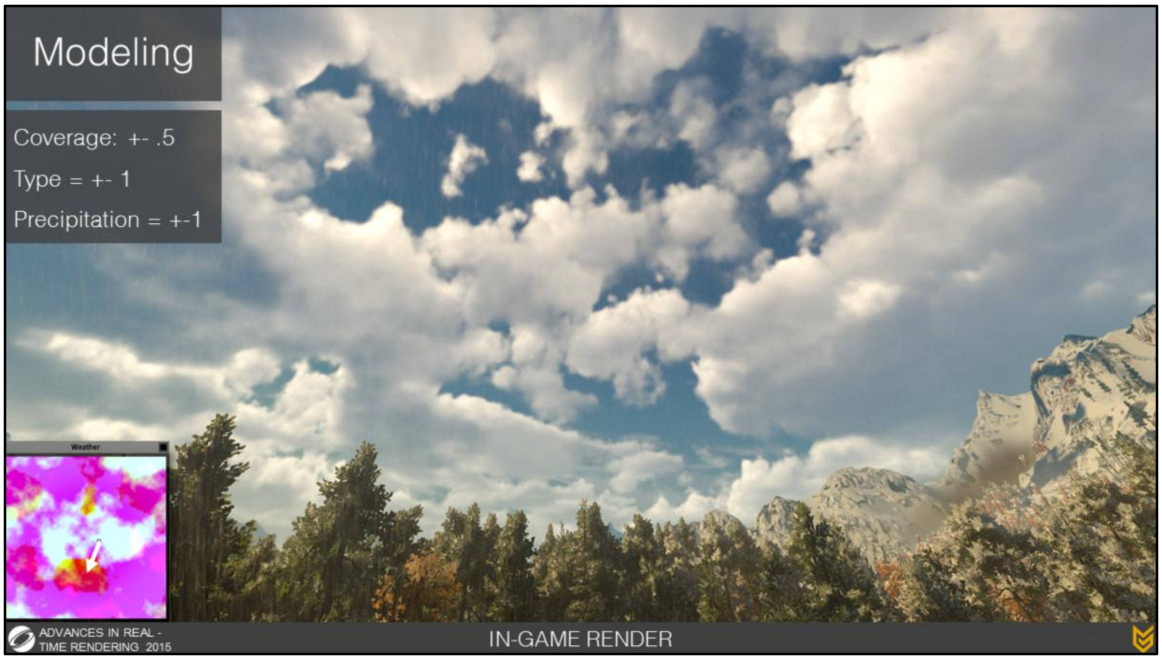
- Cloud Coverage
- Cloud Type
- Precipitation

ADVANCES IN REAL - TIME RENDERING 2015

IN-GAME RENDER

These two controls, cloud coverage and cloud type are a FUNCTION of our weather system.

- There is an additional control for Precipitation that we use to draw rain clouds.



Modeling

Coverage: +- .5

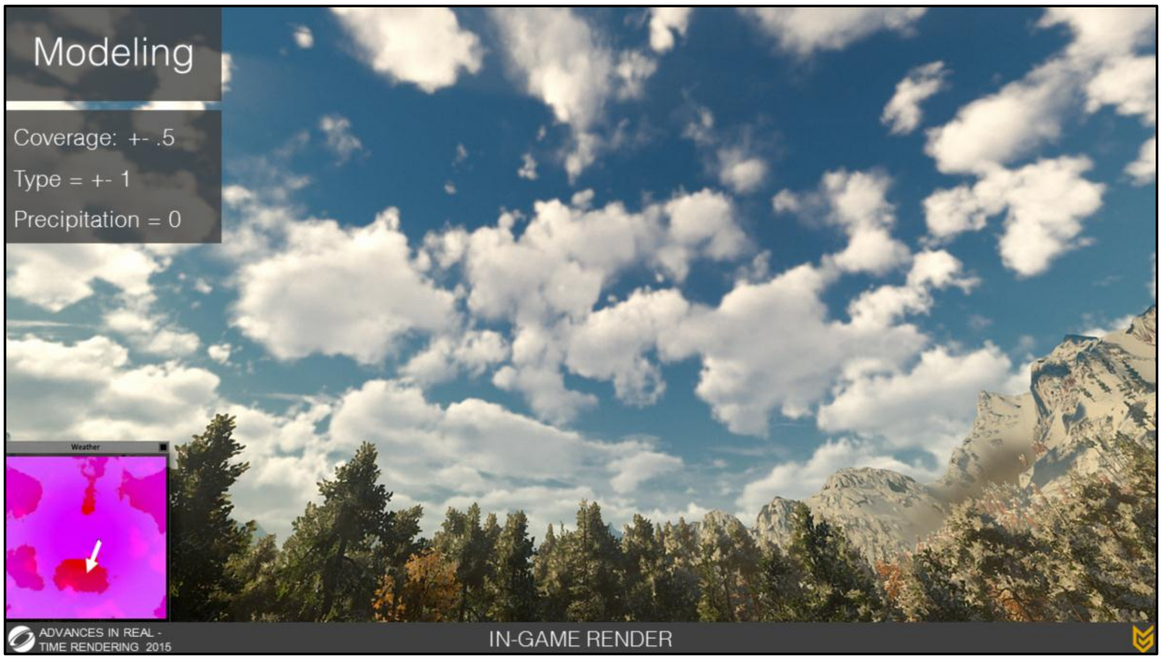
Type = +- 1

Precipitation = +-1

ADVANCES IN REAL - TIME RENDERING 2015

IN-GAME RENDER

Here in this image you can see a little map down in the lower left corner. This represents the weather settings that drive the clouds over our section of world map. The pinkish white pattern you see is the output from our weather system. Red is coverage, Green is precipitation and blue is cloud type. The weather system modulates these channels with a simulation that progresses during gameplay. The image here has Cumulus rain clouds directly overhead (white) and regular cumulus clouds in the distance. We have controls to bias the simulation to keep things art direct-able in a general sense.



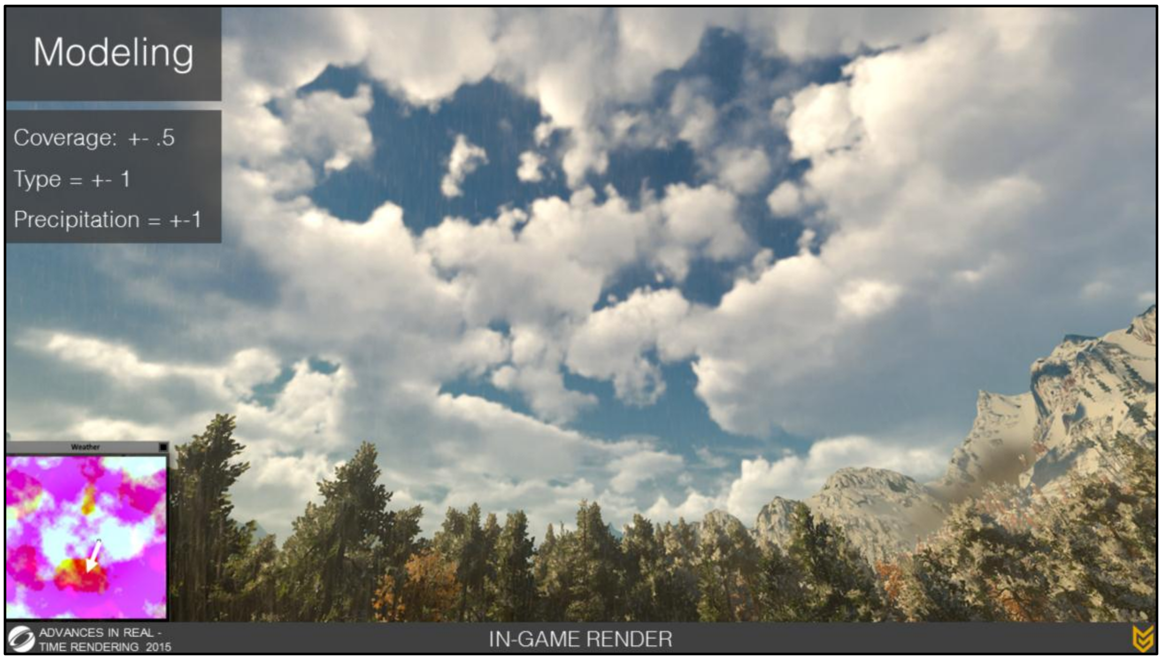
Modeling

Coverage: +- .5

Type = +- 1

Precipitation = 0

The default condition is a combination of cumulus and stratus clouds. The areas that are more red have less of the blue signal, making them stratus clouds. You can see them in the distance at the center bottom of the image.



Modeling

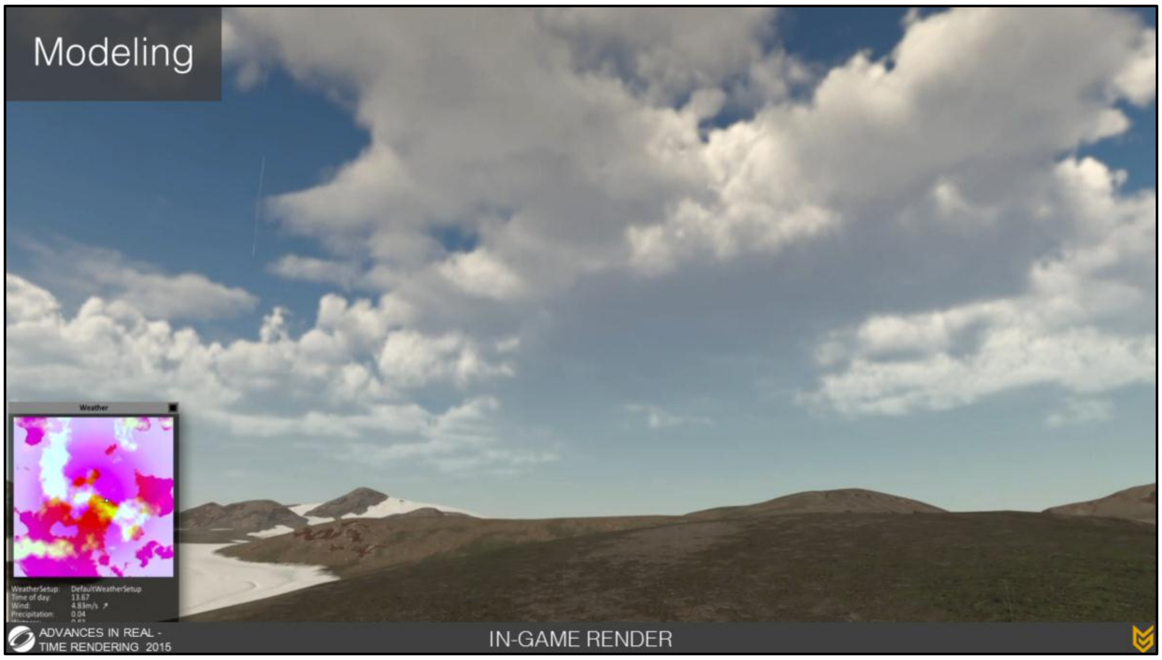
Coverage: +- .5

Type = +- 1

Precipitation = +-1

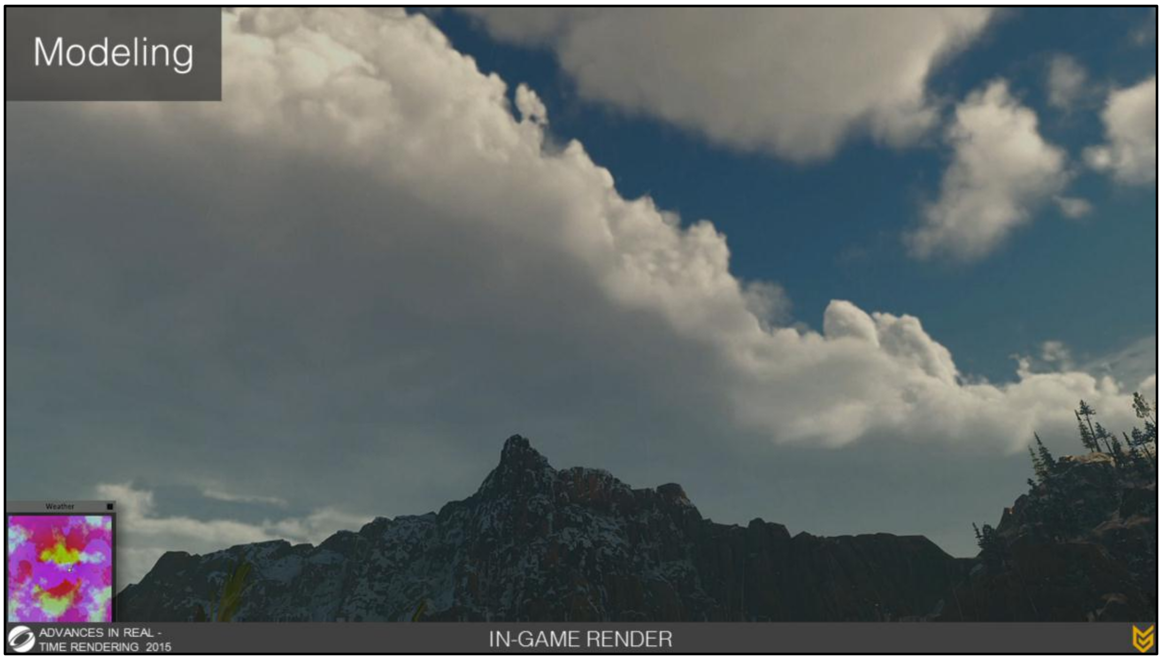
The precipitation signal transitions the map from whatever it is to cumulonimbus clouds at 70% coverage.

Modeling

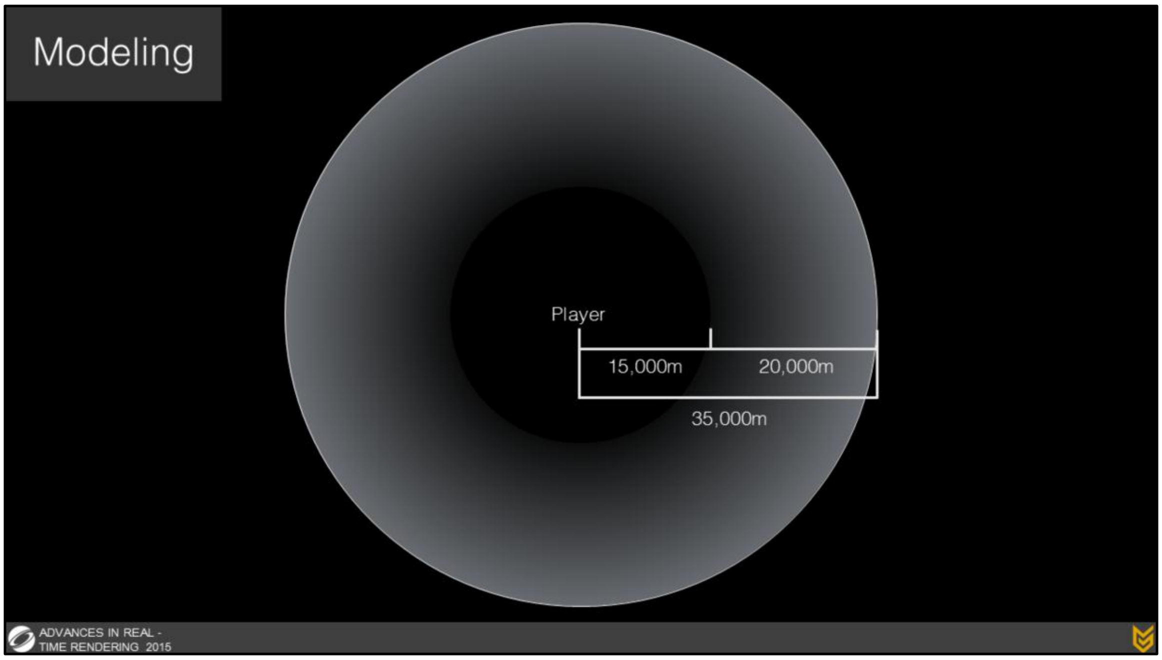


```
WeatherSetup:    DefaultWeatherSetup
Time of day:     13.67
Wind:            4.83m/s ↗
Precipitation:   0.04
```

The precipitation control not only adjusts clouds but it creates rain effects. In this video I am increasing the chance of precipitation gradually to 100%

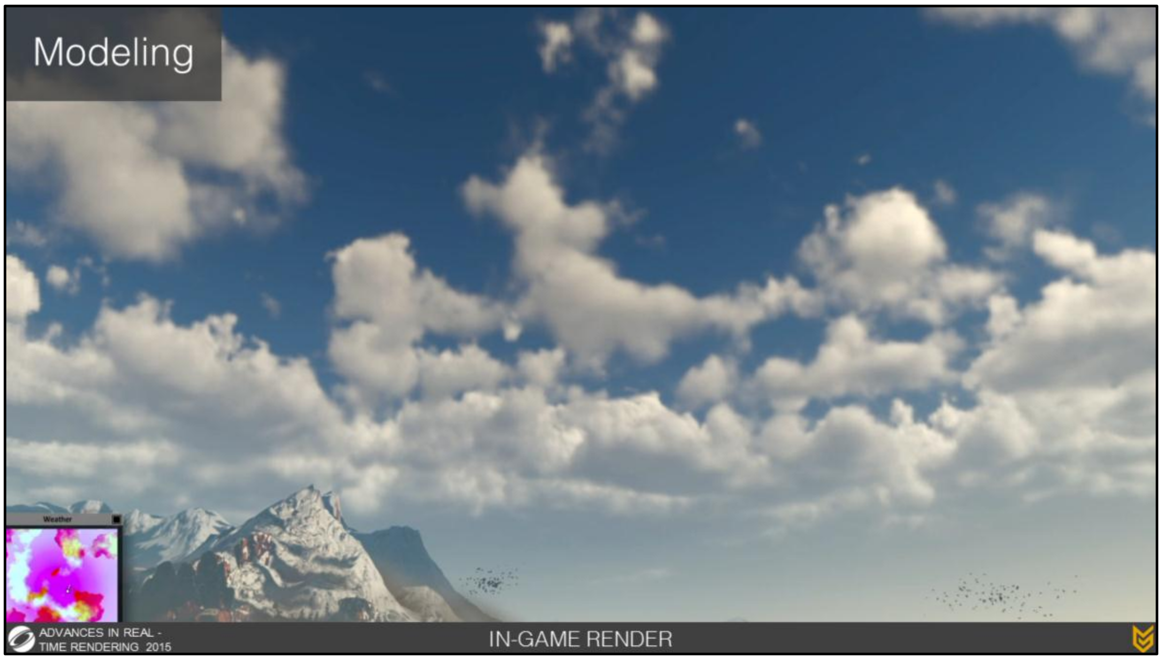


If we increase the wind speed and make sure that there is a chance of rain, we can get Storm clouds rolling in and starting to drop rain on us. This video is sped up, for effect, btw. Ahhh… Nature Sounds.

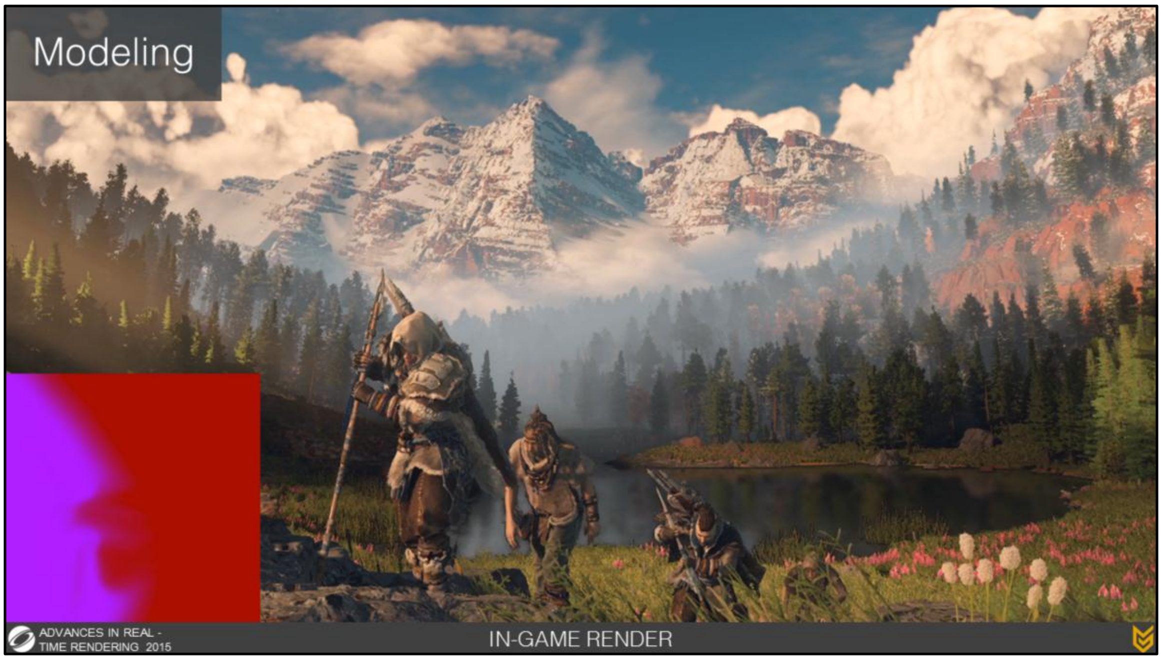


Modeling

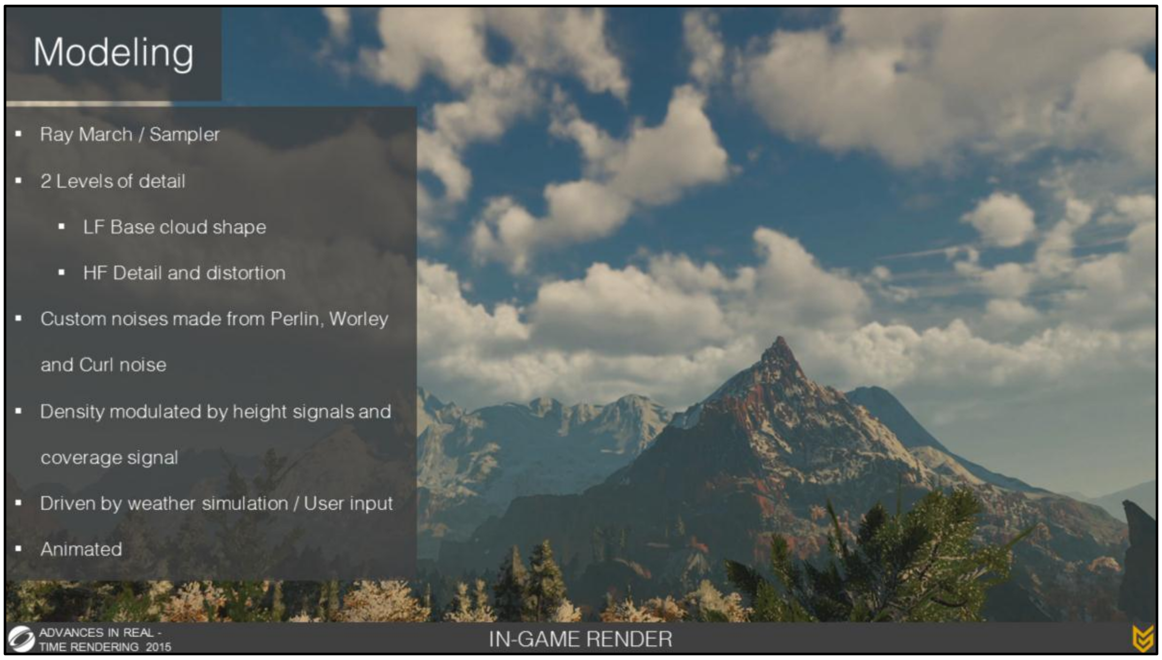
- We also use our weather system to make sure that clouds are the horizon are always interesting and poke above mountains.
- We draw the cloudscapes within a 35,000 meter radius around the player….
- and Starting at a distance of 15,000 meters…
- we start transitioning to cumulus clouds at around 50% coverage.



This ensures that there is always some variety and ‘epicness’ to the clouds on the horizon.
So, as you can see, the weather system produces some nice variation in cloud type and coverage.



In the case of the e3 trailer, We overrode the signals from the weather system with custom textures. You can see the corresponding textures for each shot in the lower left corner. We painted custom skies for each shot in this manner.

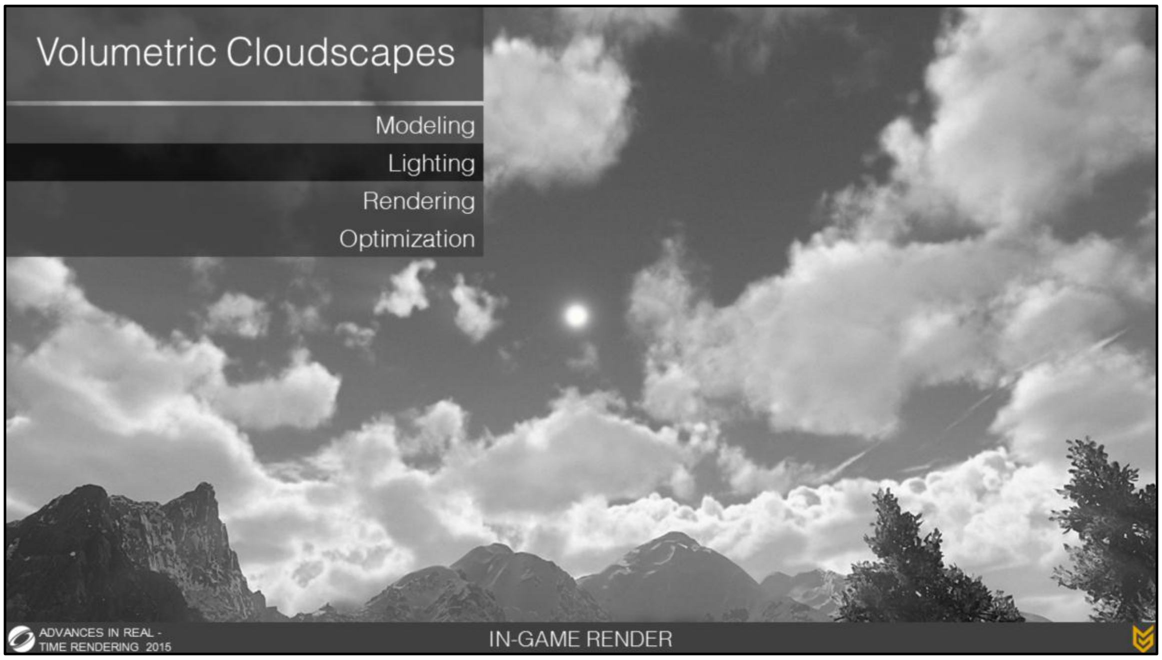

Modeling

- Ray March / Sampler
- 2 Levels of detail
  - LF Base cloud shape
  - HF Detail and distortion
- Custom noises made from Perlin, Worley and Curl noise
- Density modulated by height signals and coverage signal
- Driven by weather simulation / User input
- Animated

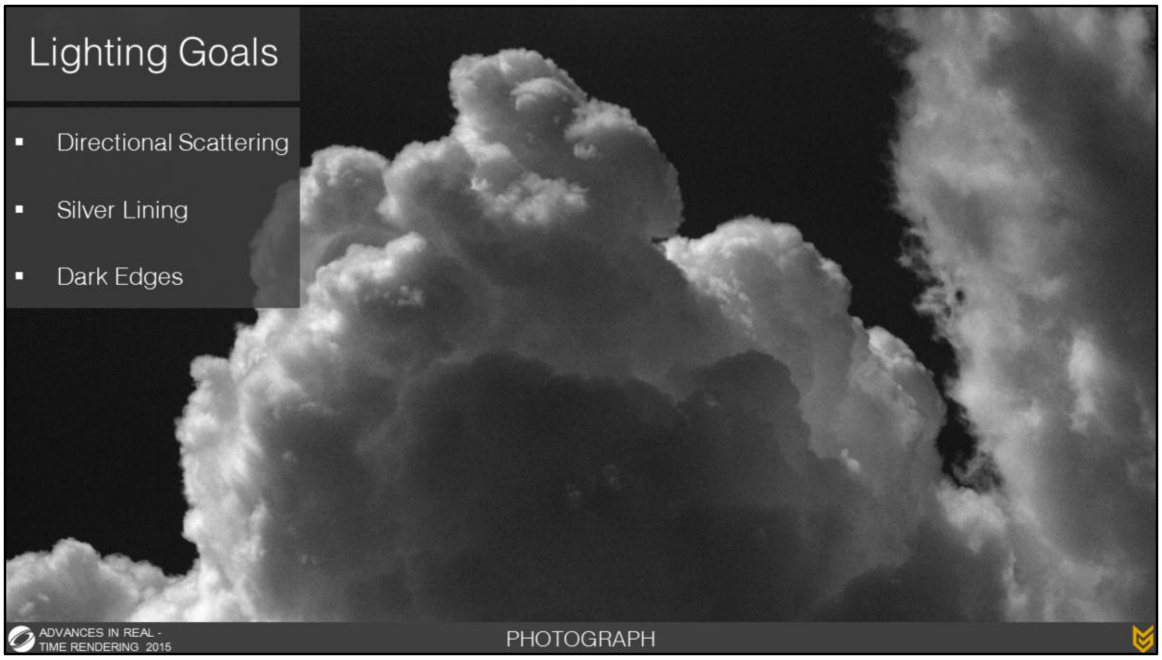
ADVANCES IN REAL - TIME RENDERING 2015

IN-GAME RENDER

So to sum up our modeling approach…
we follow the standard ray-march/ sampler framework
but we build the clouds with two levels of detail
a low frequency cloud base shape
and high frequency detail and distortion
Our noises are custom and made from Perlin, Worley and Curl noise
We use a set of presets for each cloud type to control density over height and cloud coverage
These are driven by our weather simulation or by custom textures for use with cutscenes
and it is all animated in a given wind direction.



Cloud lighting is a very well researched area in computer graphics. The best results tend to come from high numbers of samples. In games, when you ask what the budget will be for lighting clouds, you might very well be told “Zero”. We decided that we would need to examine the current approximation techniques to reproduce the 3 most important lighting effects for us.



Lighting Goals

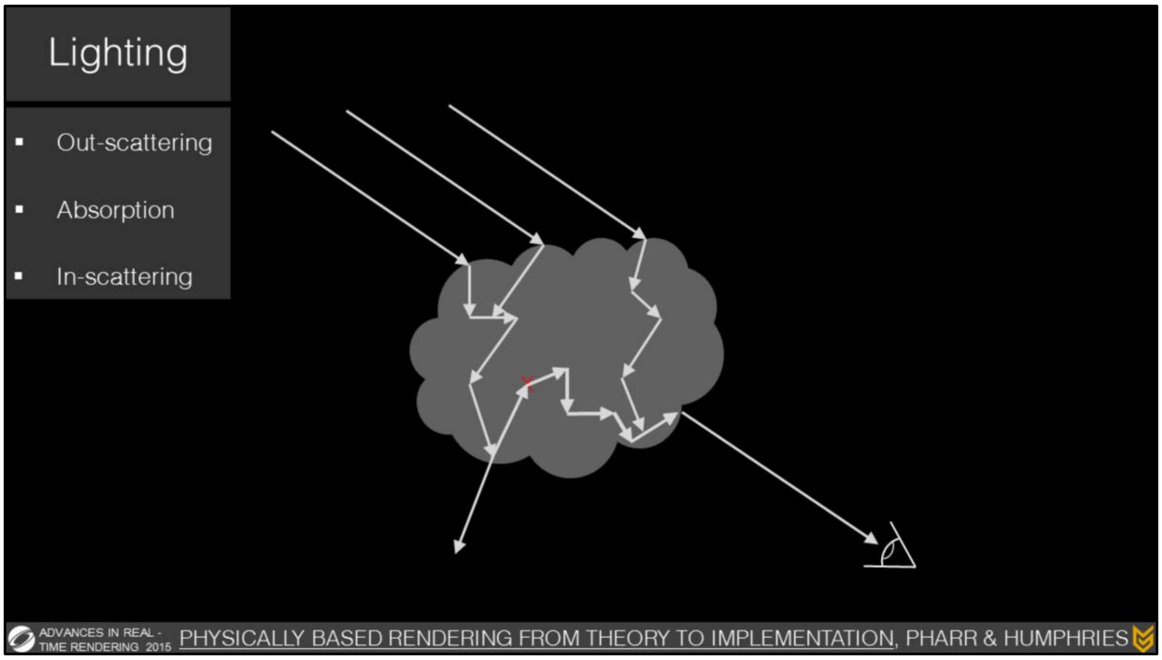
- Directional Scattering
- Silver Lining
- Dark Edges

ADVANCES IN REAL - TIME RENDERING 2015

PHOTOGRAPH

- The directional scattering or luminous quality of clouds…
- The sliver lining when you look toward the sun through a cloud…
- And the dark edges visible on clouds when you look away from the sun.

The first two have standard solutions but the third is something we had to solve ourselves.



Lighting

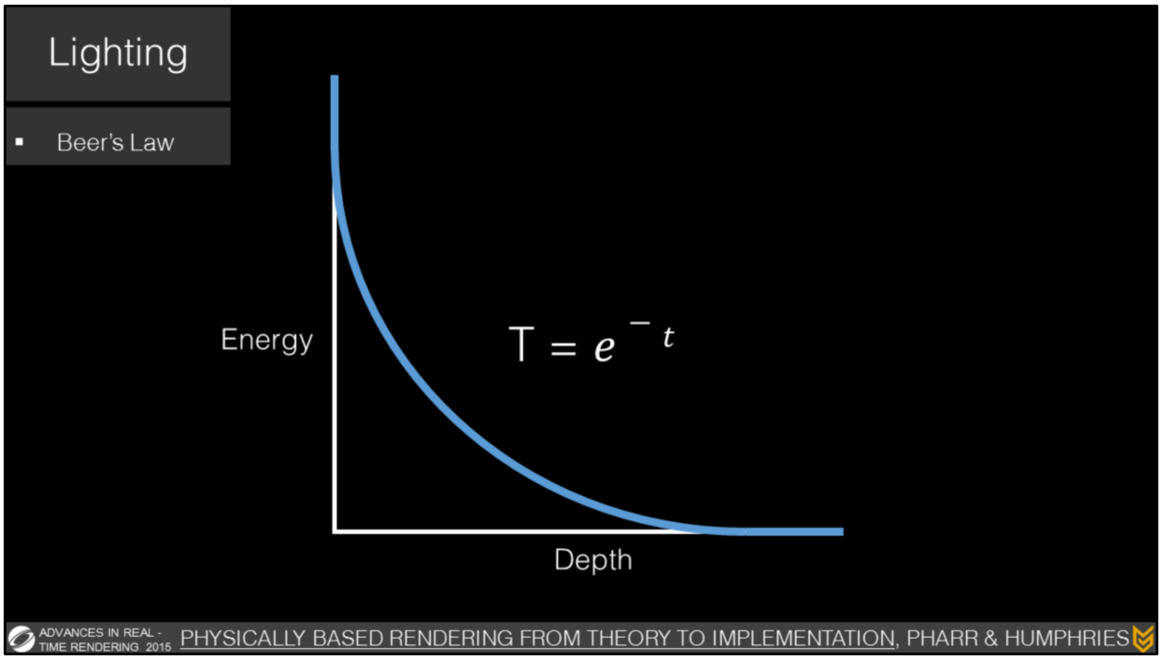
- Out-scattering
- Absorption
- In-scattering

ADVANCES IN REAL-TIME RENDERING 2015 PHYSICALLY BASED RENDERING FROM THEORY TO IMPLEMENTATION, PHARR & HUMPHRIES

When light enters a cloud
- The majority of the
- light rays spend their
- time refracting
- off of water
- droplets and
- ice inside of
- the cloud
- before heading
- to our eyes.

(pause)
- By the time the light ray finally exits the cloud it could have been out scattered…
- absorbed by the cloud…
- or combined with…
- other light rays in what is called in-scattering..

In film vfx we can afford to spend time gathering light and accurately reproducing this, but in games we have to use approximations. These three behaviors can be thought of as probabilities and there is a
Standard way to approximate the result you would get.



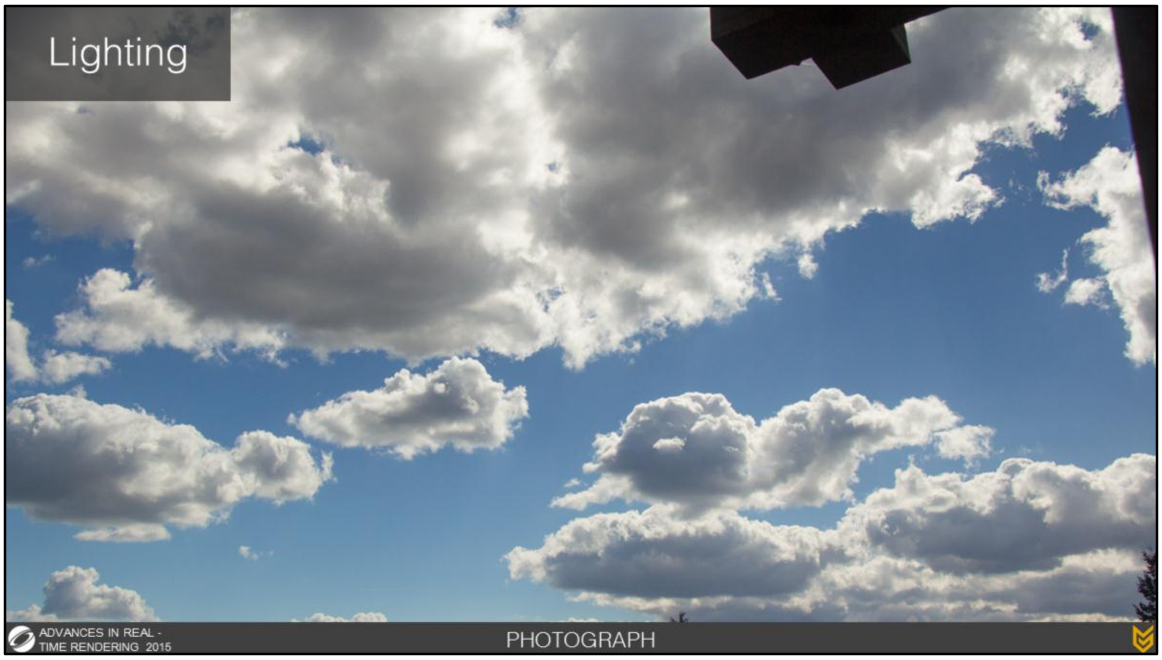
Lighting

- Beer's Law

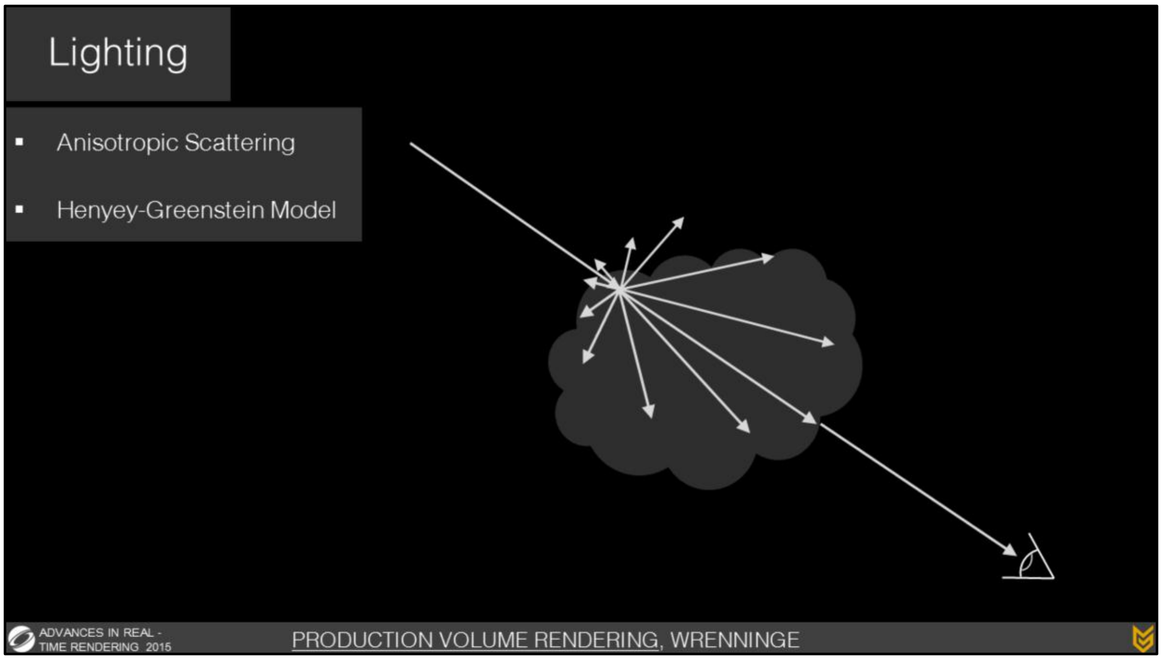
ADVANCES IN REAL - TIME RENDERING 2015 PHYSICALLY BASED RENDERING FROM THEORY TO IMPLEMENTATION, PHARR & HUMPHRIES

Beer’s law states that we can determine the amount of light reaching a point based on the optical thickness of the medium that it travels through. With Beers law, we have a basic way to describe the amount of light at a given point in the cloud.

- If we substitute energy for transmittance ad depth in the cloud for thickness, and draw this out you can see that energy exponentially decreases over depth. This forms the foundation of our lighting model.



but there is a another component contributing to the light energy at a point. It is the probability of light scattering forward or backward in the cloud. This is responsible for the silver lining in clouds, one of our look goals.



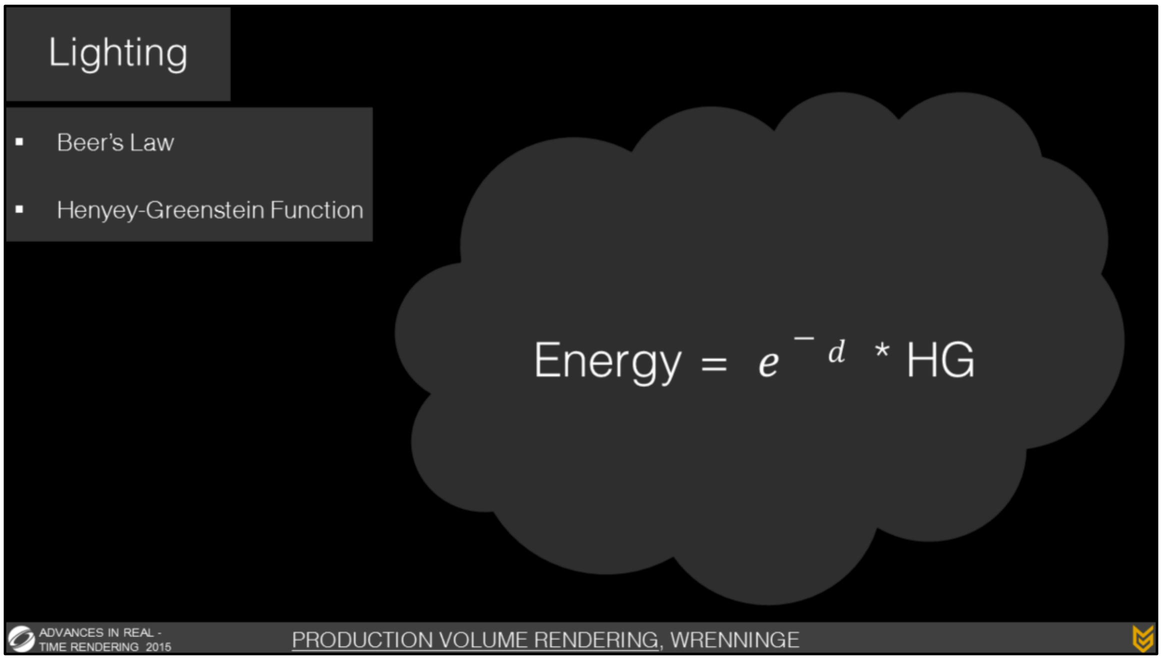
Lighting

- Anisotropic Scattering
- Henyey-Greenstein Model

ADVANCES IN REAL - TIME RENDERING 2015

PRODUCTION VOLUME RENDERING, WRENNINGE

- In clouds, there is a higher probability of light scattering forward. This is called Anisotropic scattering.
- In 1941, the Henyey-Greenstein model was developed to help astronomers with light calculations at galactic scales, but today it is used to reliably reproduce Anisotropy in cloud lighting.



Lighting

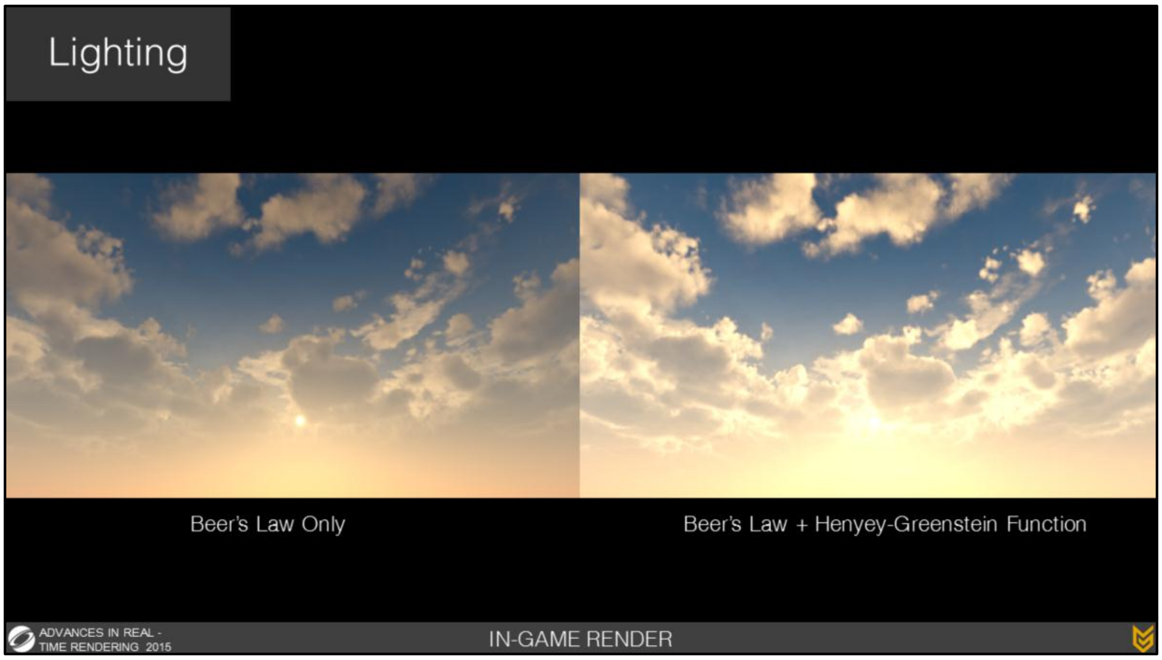
- Beer's Law
- Henyey-Greenstein Function

$$\text{Energy} = e^{-d} * \text{HG}$$

ADVANCES IN REAL-TIME RENDERING 2015

PRODUCTION VOLUME RENDERING, WRENNINGE

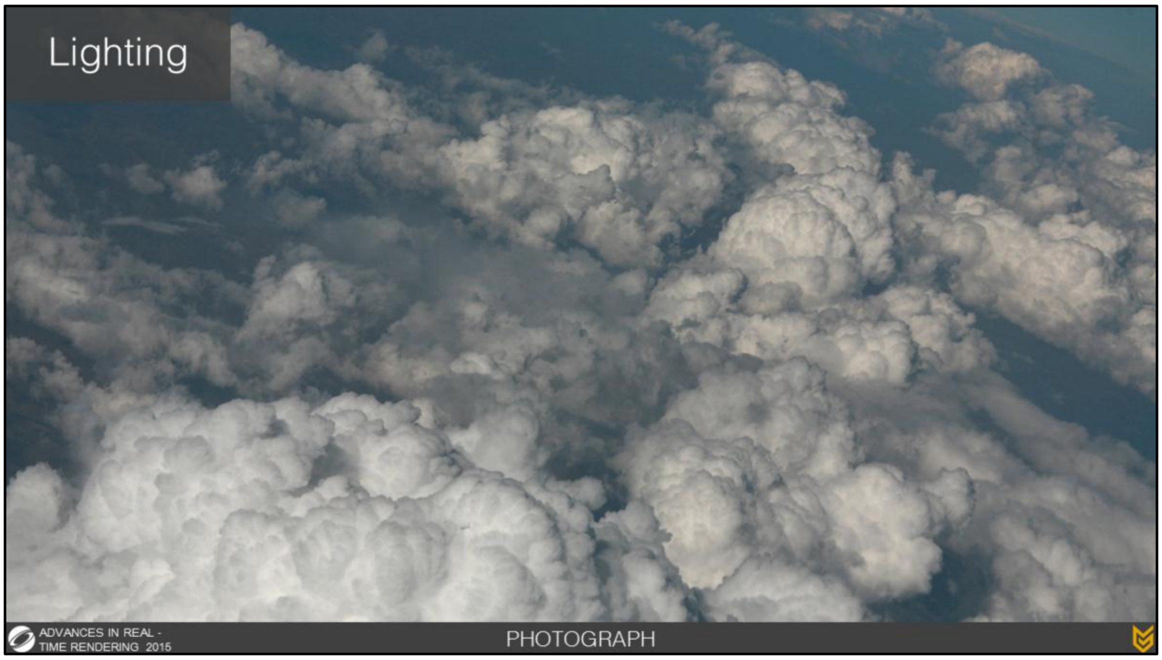
Each time we sample light energy, we multiply it by The Henyey-Greenstein phase function.



Lighting

Beer's Law Only
Beer's Law + Henyey-Greenstein Function
ADVANCES IN REAL - TIME RENDERING 2015
IN-GAME RENDER

Here you can see the result. On the left is Just the beers law portion of our lighting model. On the right we have applied the Henyey-Greenstein phase function. Notice that the clouds are brighter around the sun on the right.

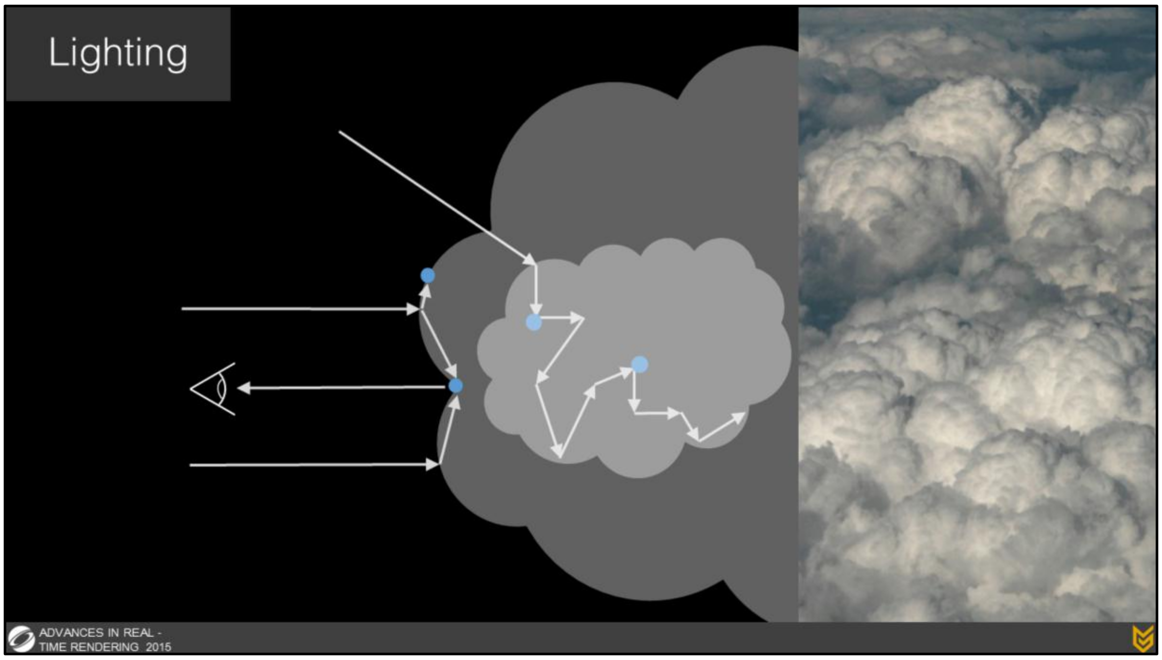


Lighting

ADVANCES IN REAL - TIME RENDERING 2015

PHOTOGRAPH

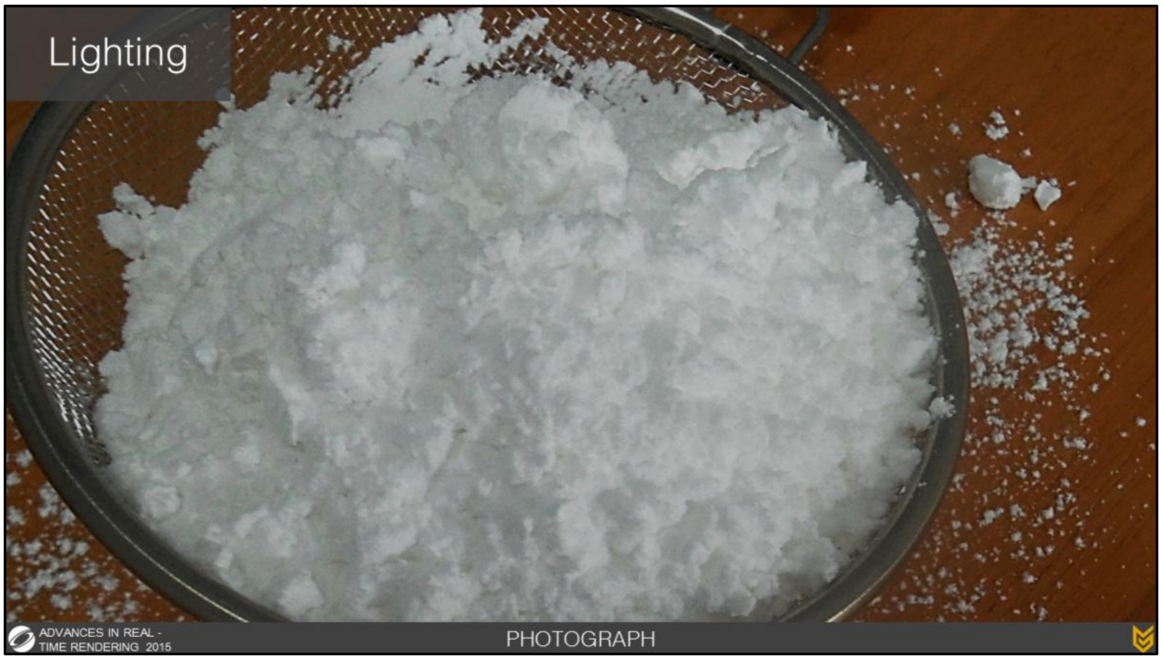
But we are still missing something important, something that is often forgotten. The dark edges on clouds. This is something that is not as well documented with solutions so we had to do a thought experiment to understand what was going on.



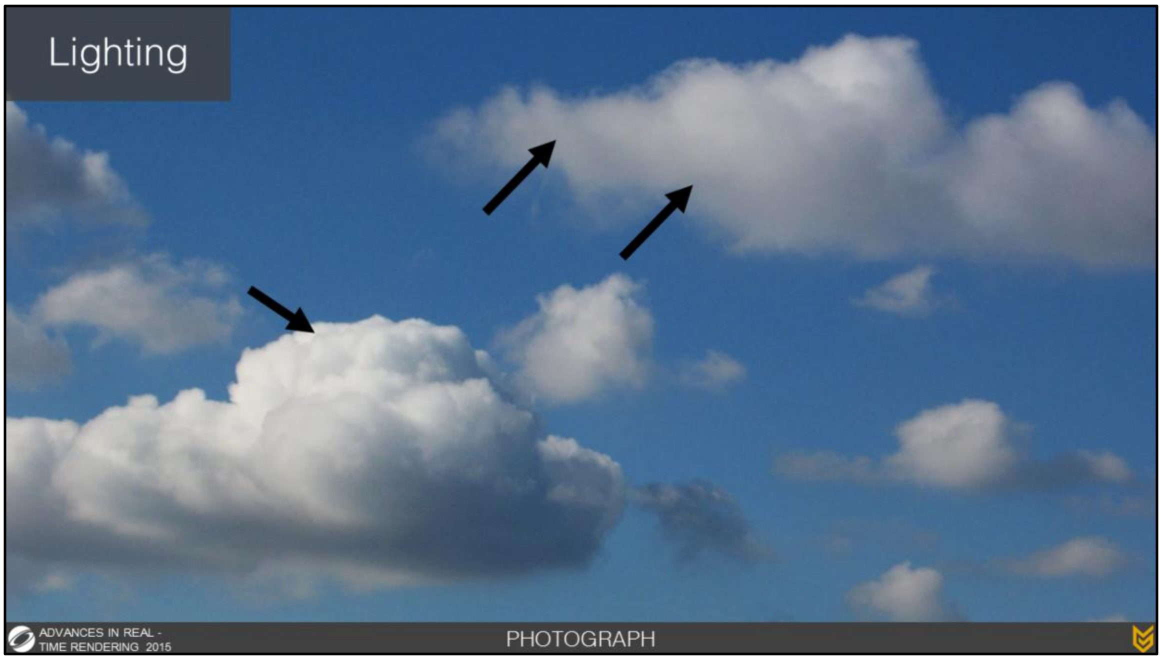
Think back to the random walk of a light ray through a cloud.

- If we compare a point inside of the cloud to one near the surface, the one inside would receive more in scattered light. In other words, Cloud material, if you want to call it that, is a collector for light. The deeper you are in the surface of a cloud, the more potential there is for gathered light from nearby regions until the light begins to attenuate, that is.
-
- This is extremely pronounced in round formations on clouds, so much so that the crevices appear…
- to be lighter that the bulges and edges because they receive a small boost of in-scattered light.
-

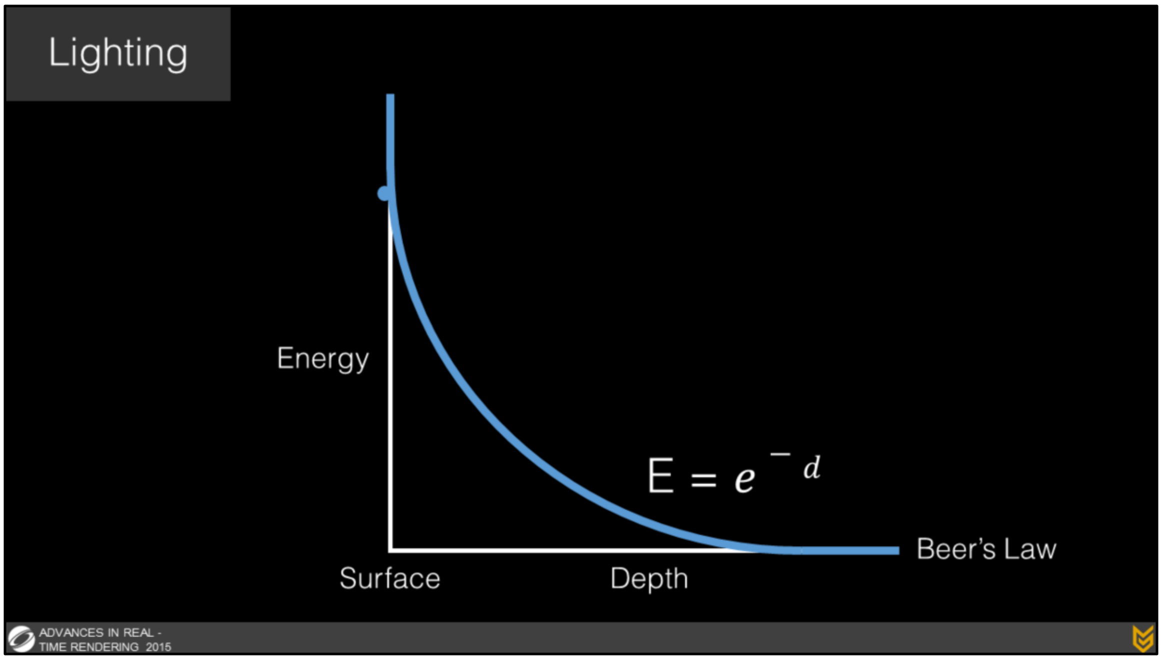
Normally in film, we would take many many samples to gather the contributing light at a point and use a more expensive phase function. You can get this result with brute force. If you were in Magnus Wrenninge's multiple scattering talk yesterday there was a very good example of how to get this. But in games we have to find a way to approximate this.



A former colleague of mine, Matt Wilson, from Blue Sky, said that there is a similar effect in piles of powdered sugar. So, I’ll refer to this as the powdered sugar look.

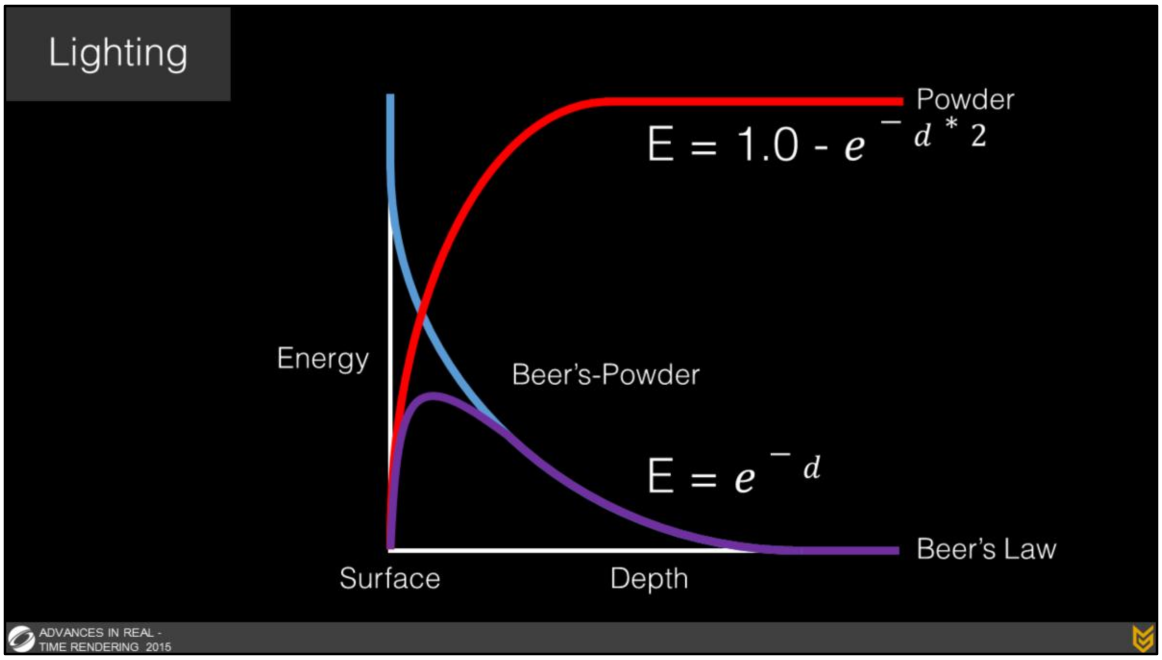

Lighting

Once you understand this effect, you begin to see it everywhere. It can not be un-seen.
Even in light whispy clouds. The dark gradient is just wider.



The reason we do not see this effect automatically is because our transmittance function is an approximation and doesn't take it into account.

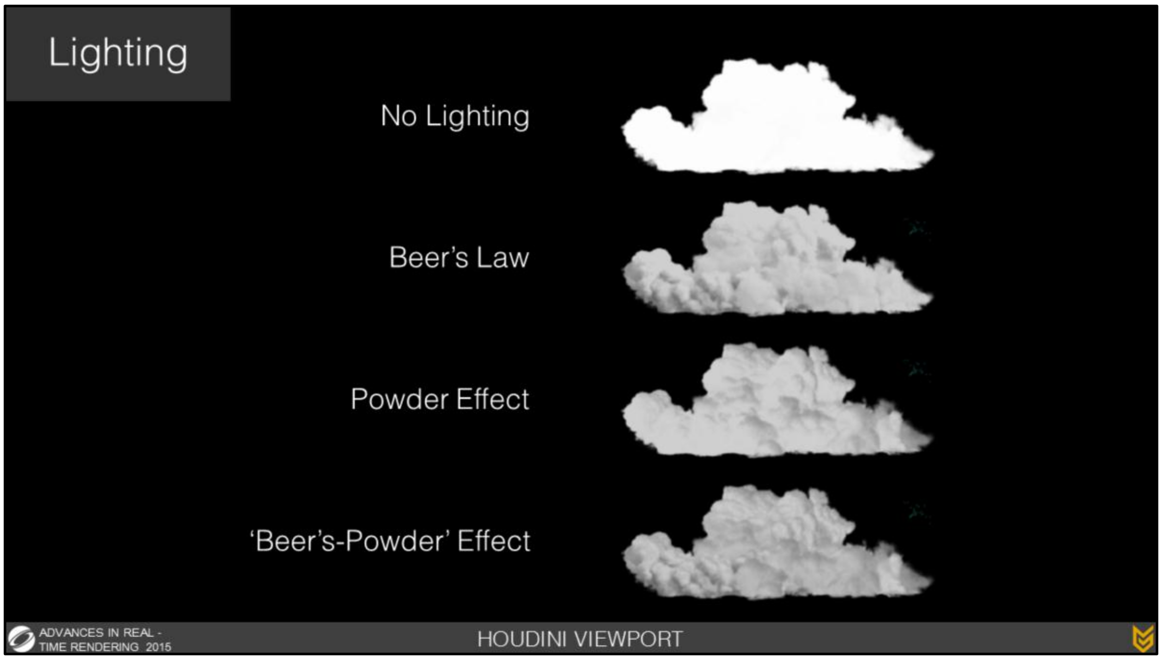
- The surface of the cloud is always going to have the same light energy that it receives. Lets think of this effect as a statistical probability based on depth.



As we go deeper in the cloud, our potential for in scattering increases and more of it will reach our eye.

- If you combine the two functions you get something that describes this…
- effect as well as the traditional approach.

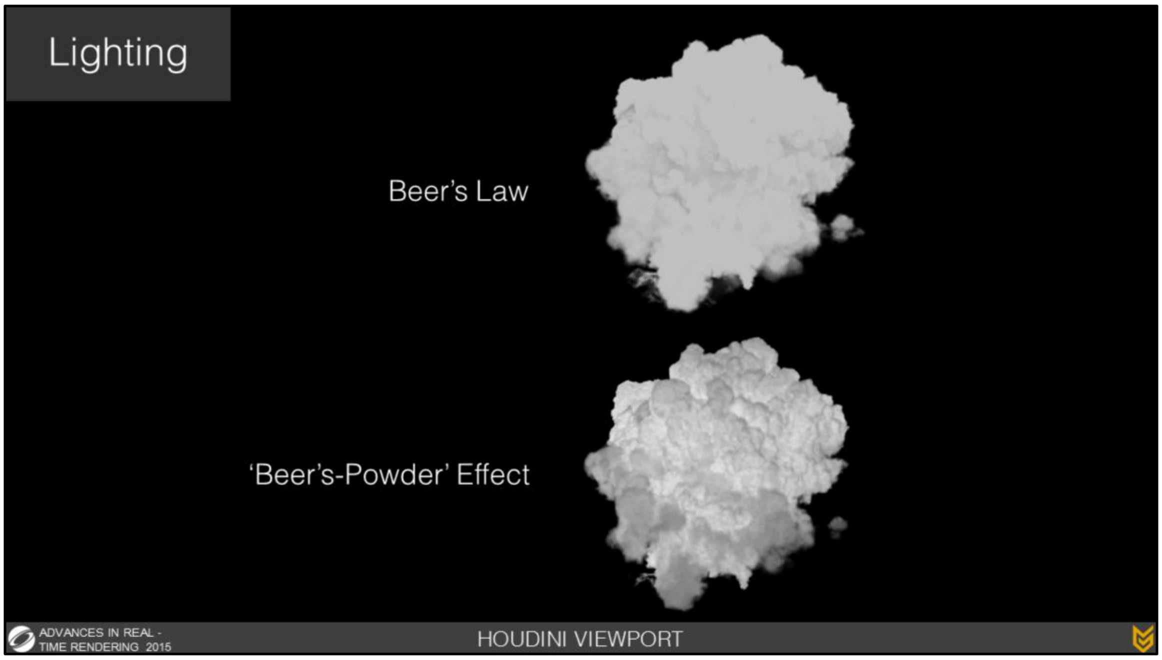
I am still looking for the Beer's-Powder approximation method in the ACM digital library and I haven't found anything mentioned with that name yet.



Lighting

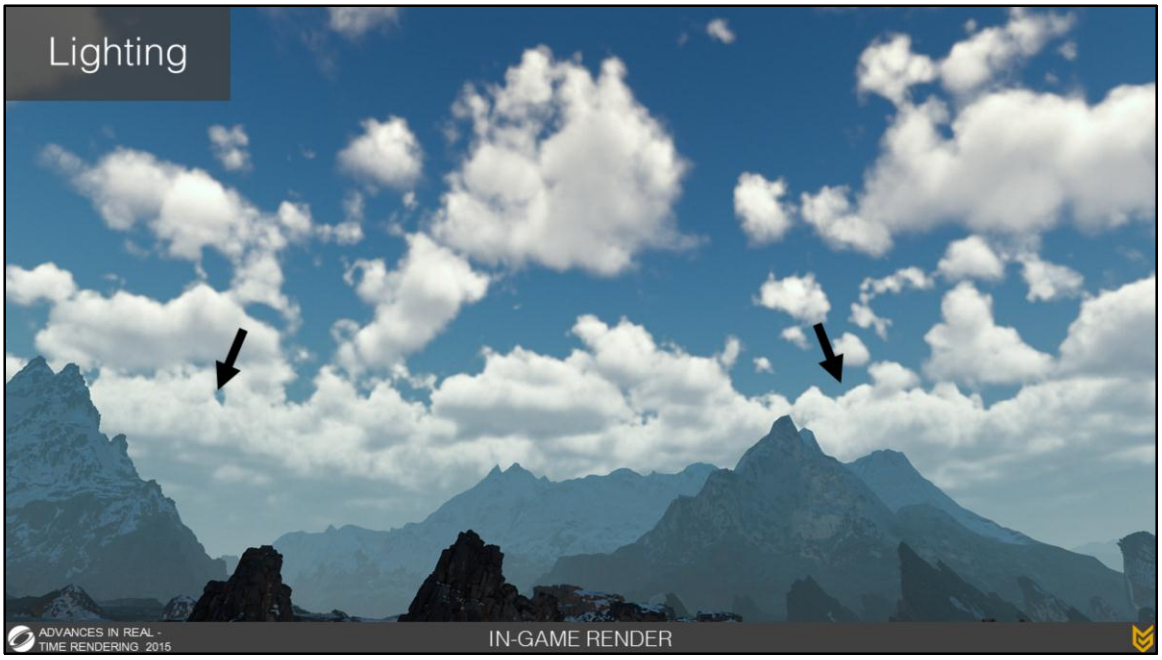
Lets visually compare the components of our directional lighting model

- The beer's law component which handles the primary scattering…
- the powder sugar effect which produces the dark edges facing the light…
- And their combination in our final result.



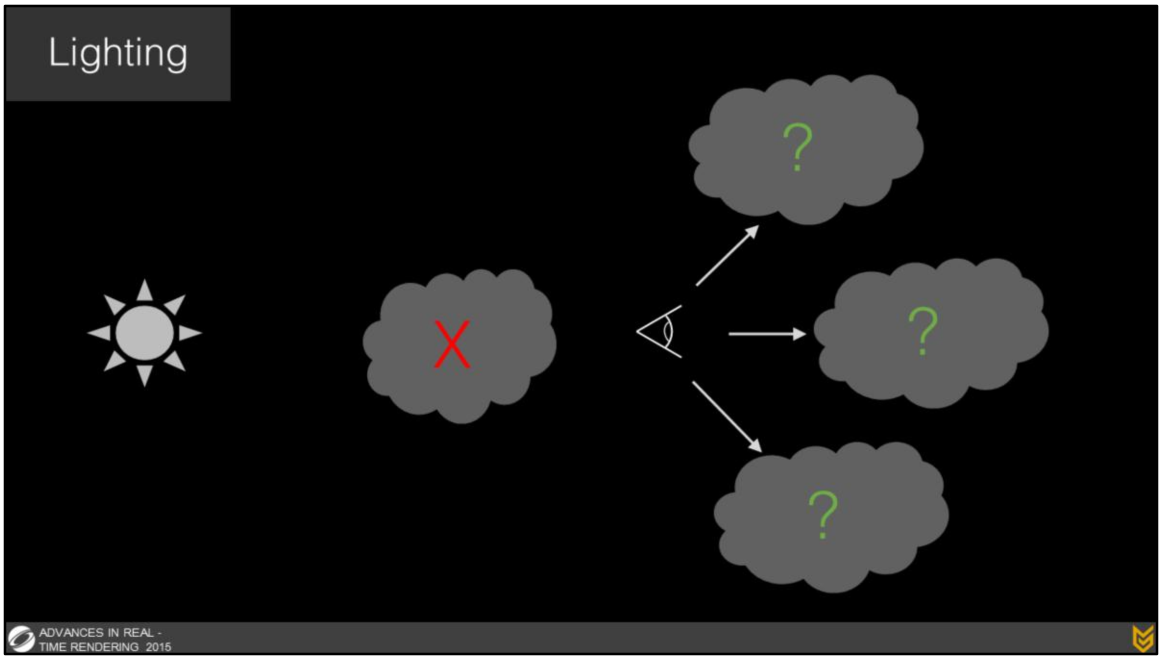
Lighting

Here you can see what the beer's law and combined beer's law and powder effect look like when viewed from the light source. This is a pretty good approximation of our reference.



In game, it adds a lot of realism to the

- thicker clouds and helps sell the scale of the scene.


Lighting

But we have to remember that this is a view dependent effect. We only see it where our view vector approaches the light vector, so the powder function should account for this gradient as well.

Lighting

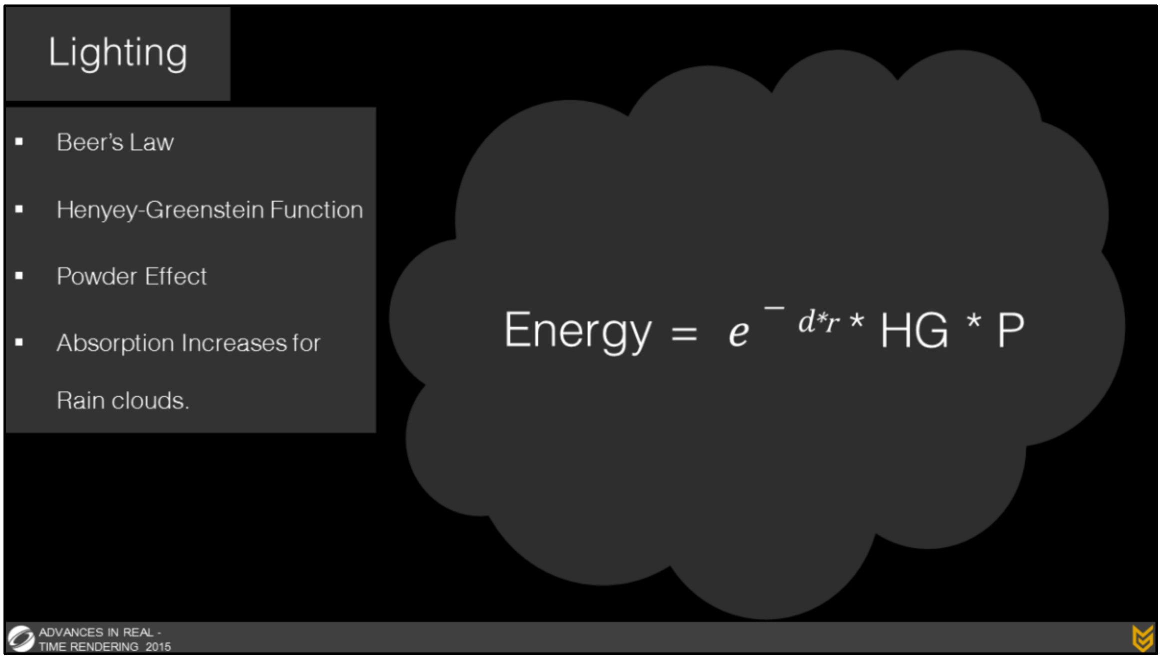
ADVANCES IN REAL - TIME RENDERING 2015

IN-GAME RENDER

Here is a panning camera view that shows this effect increasing as we look away from the sun.

Lighting
Weather
ADVANCES IN REAL - TIME RENDERING 2015
IN-GAME RENDER

The last part of our lighting model is that we artificially darken the rain clouds by …
- increasing the light absorption where they exist.



Lighting

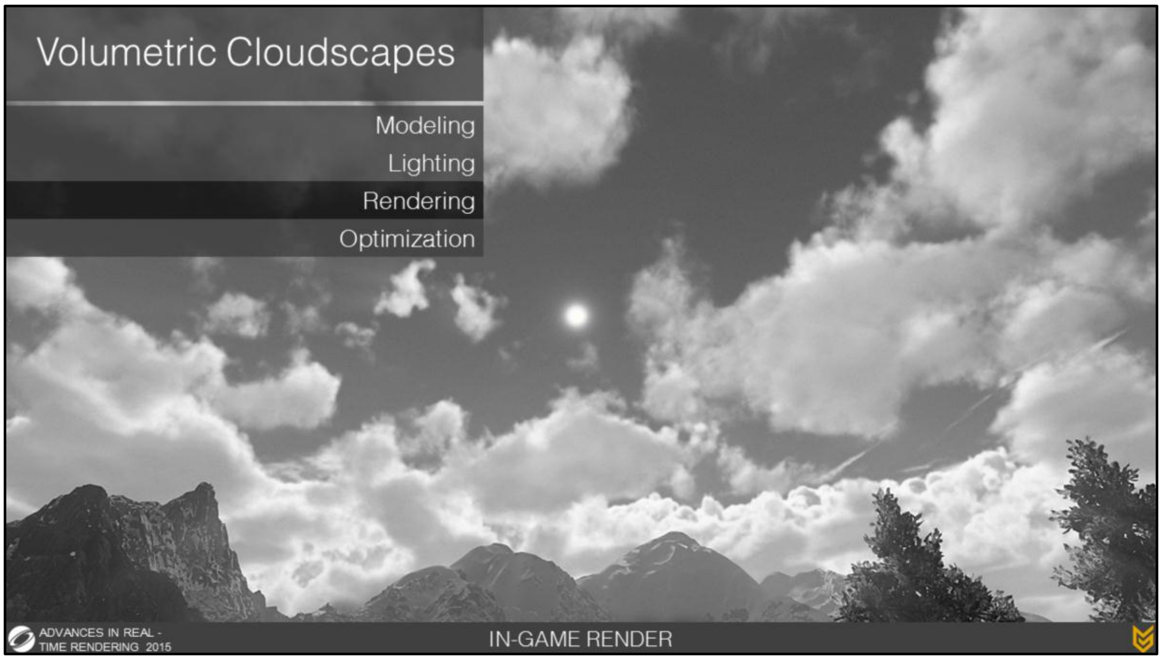
- Beer's Law
- Henyey-Greenstein Function
- Powder Effect
- Absorption Increases for Rain clouds.

$$\text{Energy} = e^{-d*r} * \text{HG} * \text{P}$$

ADVANCES IN REAL - TIME RENDERING 2015

So, in review our model has 3 components:

- Beer's Law
- Henyen-Greenstein
- our powder sugar effect
- And Absorption increasing for rain clouds

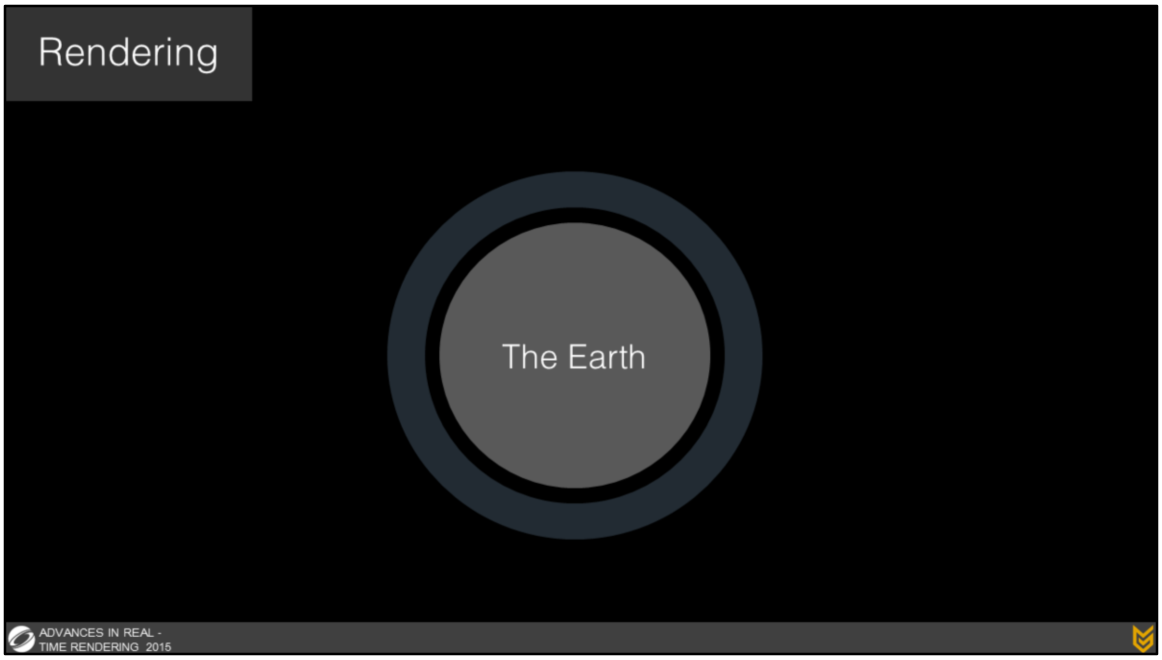

Volumetric Cloudscapes

- Modeling
- Lighting
- Rendering
- Optimization

ADVANCES IN REAL - TIME RENDERING 2015

IN-GAME RENDER

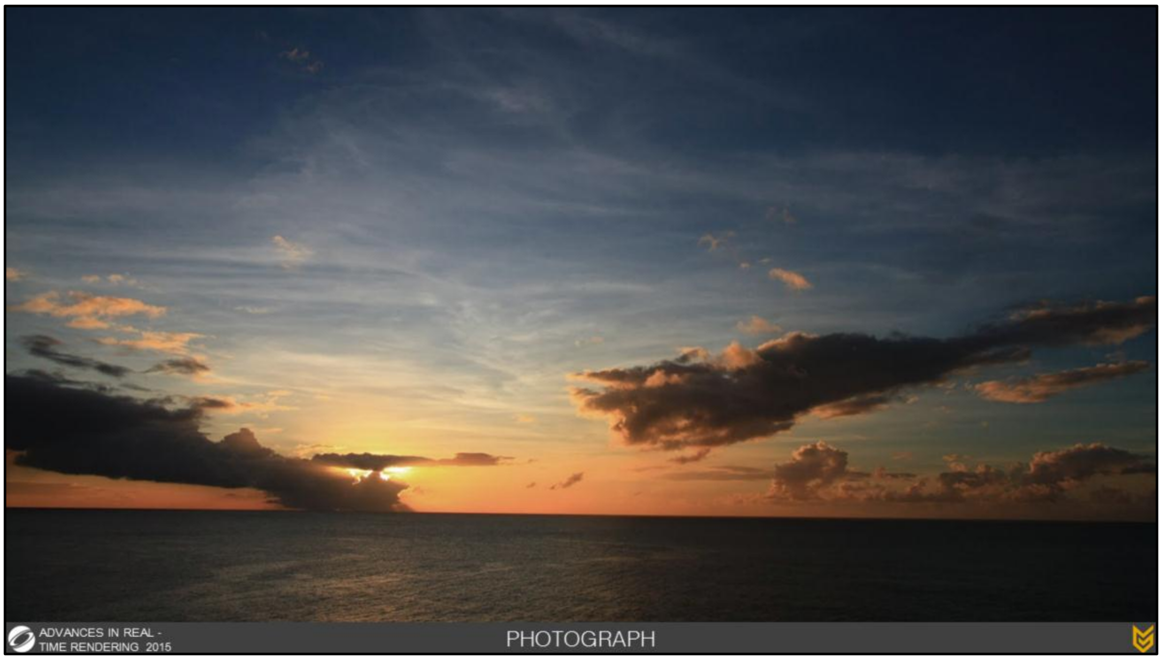
I have outlined How our sampler is used to model clouds and how our lighting algorithm simulates the lighting effects associated with them. Now I am going to describe how and where we take samples to build an image. And how we integrate our clouds into atmosphere and our time of day cycle.



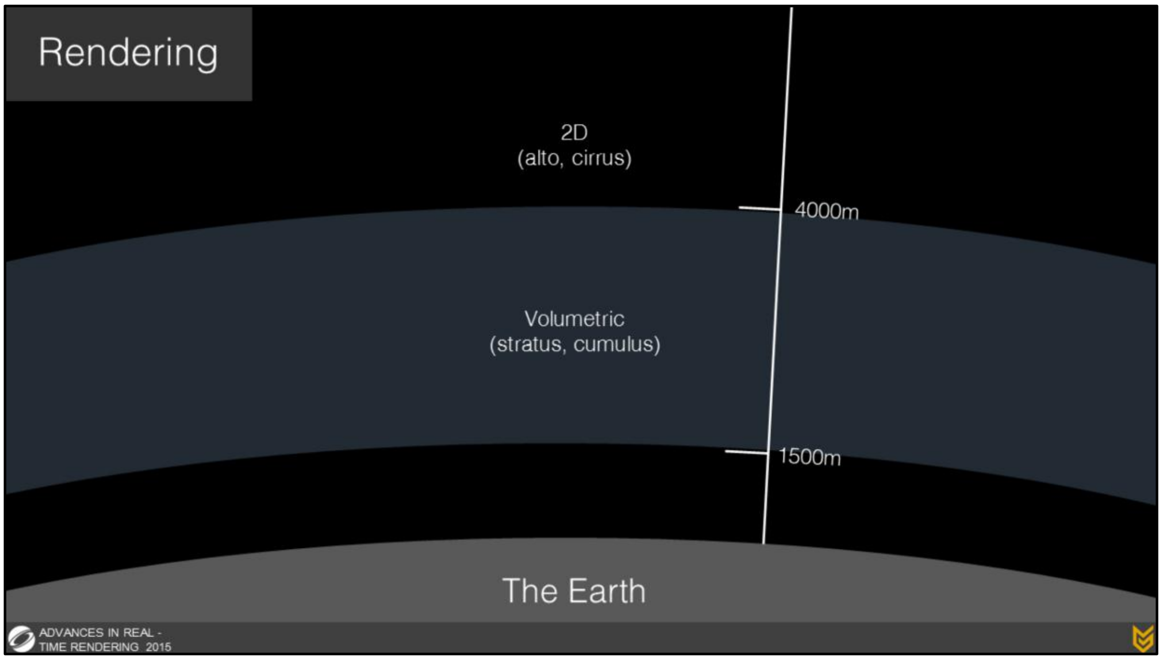
Rendering

The first part of rendering with a ray march is deciding where to start. In our situation, Horizon takes place on Earth and as most of you are aware… the earth ….. Is round.

- The gases that make up our atmosphere wrap around the earth and clouds exists in different layers of the atmosphere.

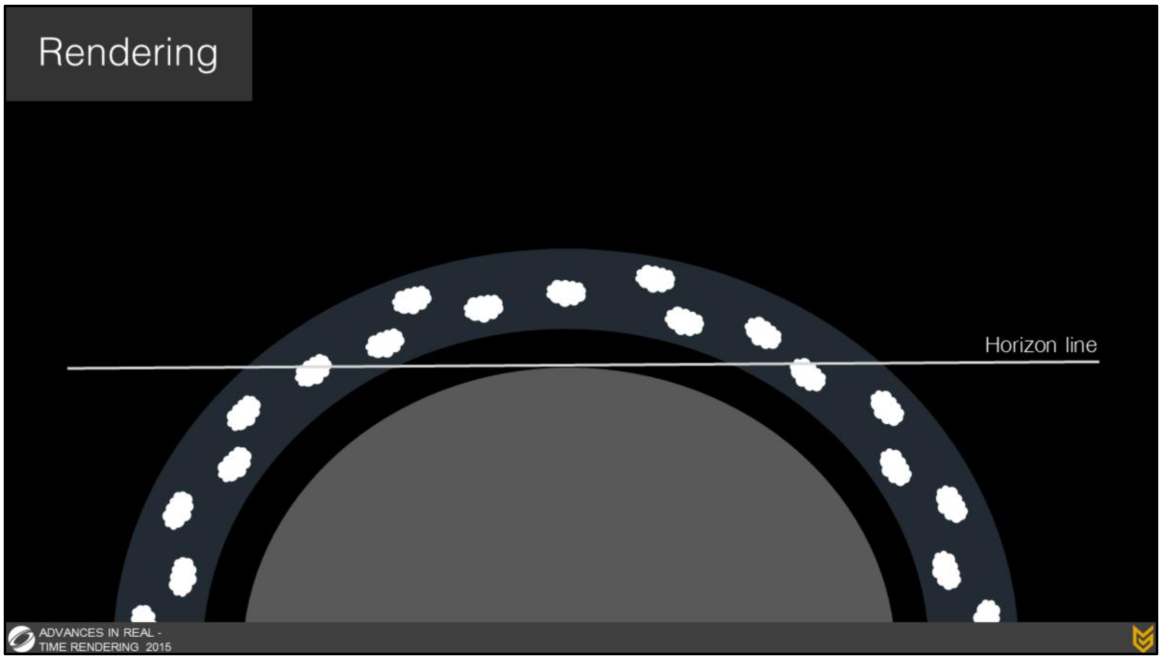


When you are on a “flat” surface such as the ocean, you can clearly see how the curvature of the earth causes clouds to descend into the horizon.


Rendering

For the purposes of our game we divide the clouds into two types in this spherical atmosphere.

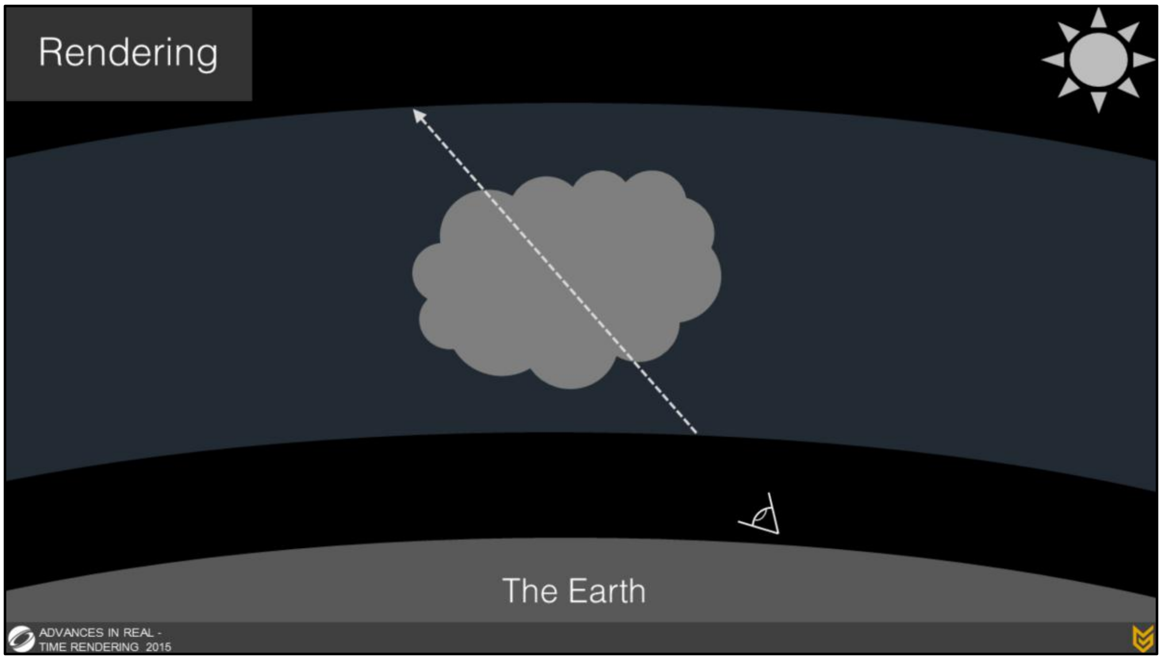
- The low altitude volumetric strato class clouds between 1500 and 4000 meters…
- and the high altitude 2D alto and cirro class clouds above 4000 meters. The upper level clouds are not very thick so this is a good area to reduce expense of the shader by making them scrolling textures instead of multiple samples in the ray march.



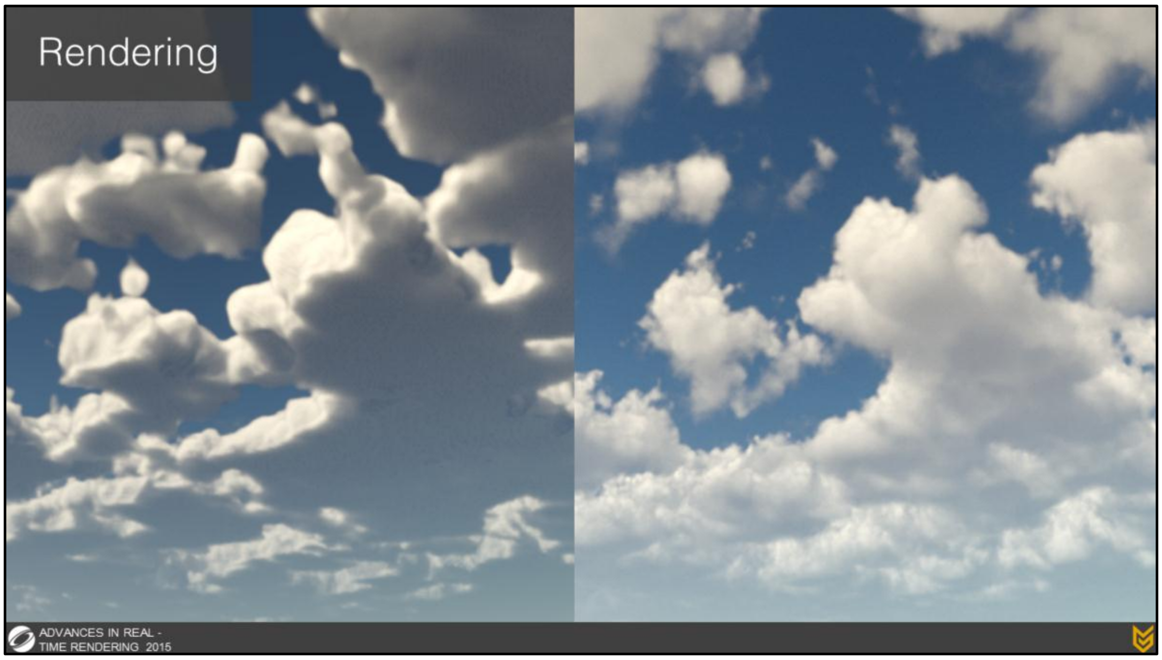
Rendering

By ray marching through spherical atmosphere we can…

- ensure that clouds properly descend into the horizon.
- It also means we can force the scale of the scene by shrinking the radius of the atmosphere.

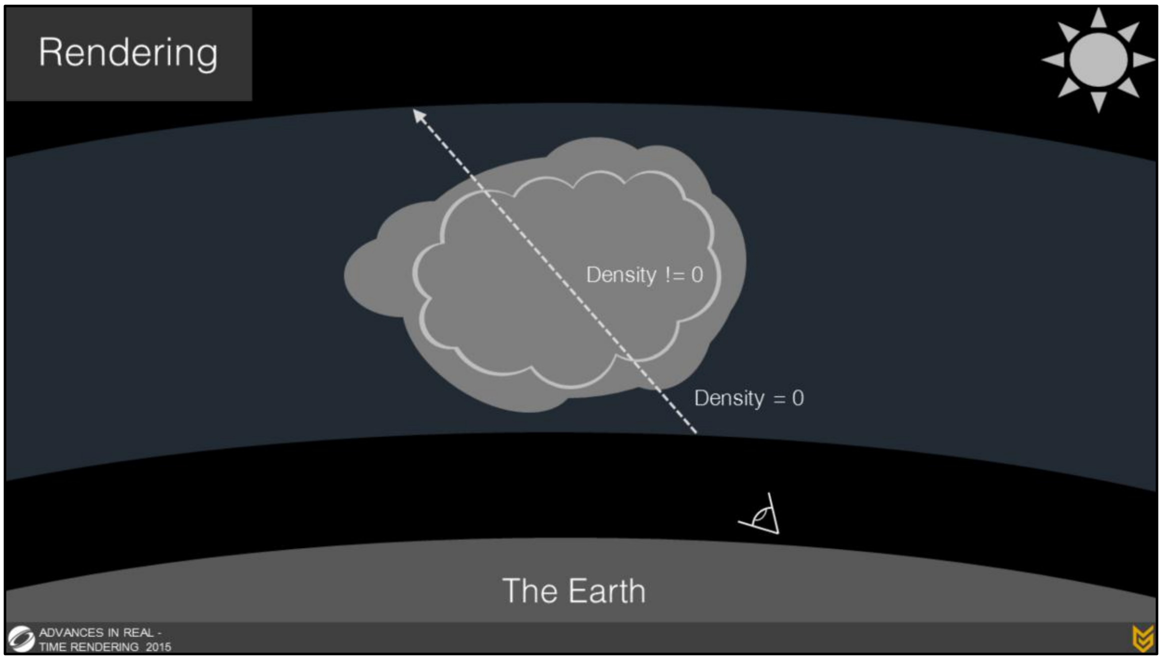


In our situation we do not want to do any work or any expensive work where we don't need to. So instead of sampling every point along the ray, we use our samplers two levels of detail as a way to do cheaper work until we actually hit a cloud.



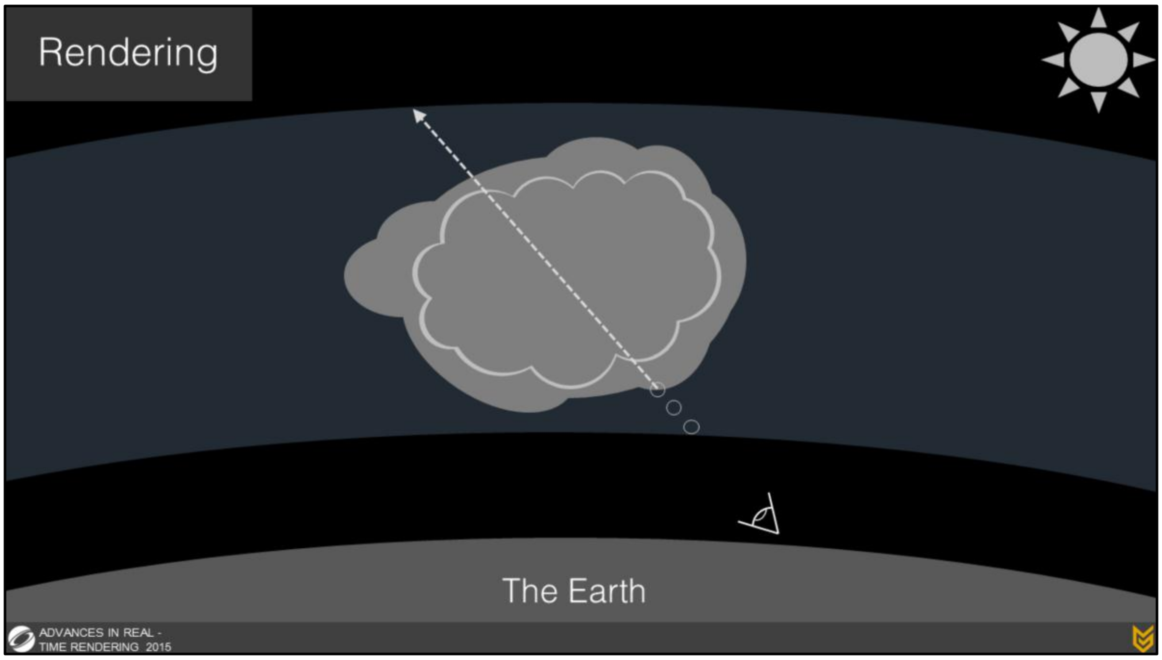
- Recall that the sampler has a low detail noise that make s basic cloud shape
- and a high detail noise that adds the realistic detail we need.

The high detail noise is always applied as an erosion from the edge of the base cloud shape.

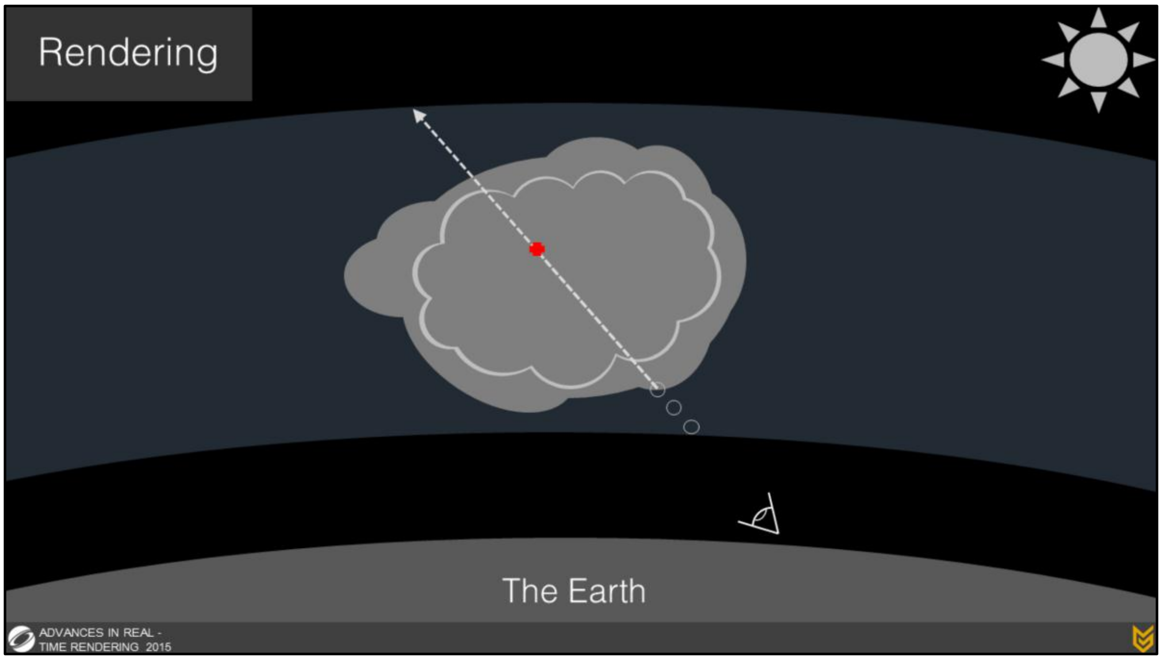


This means that we only need to do the high detail noise and all of its associated instructions where the low detail sample returns a non zero result.

- This has the effect of producing an isosurface that surrounds the area that our cloud will be that could be.

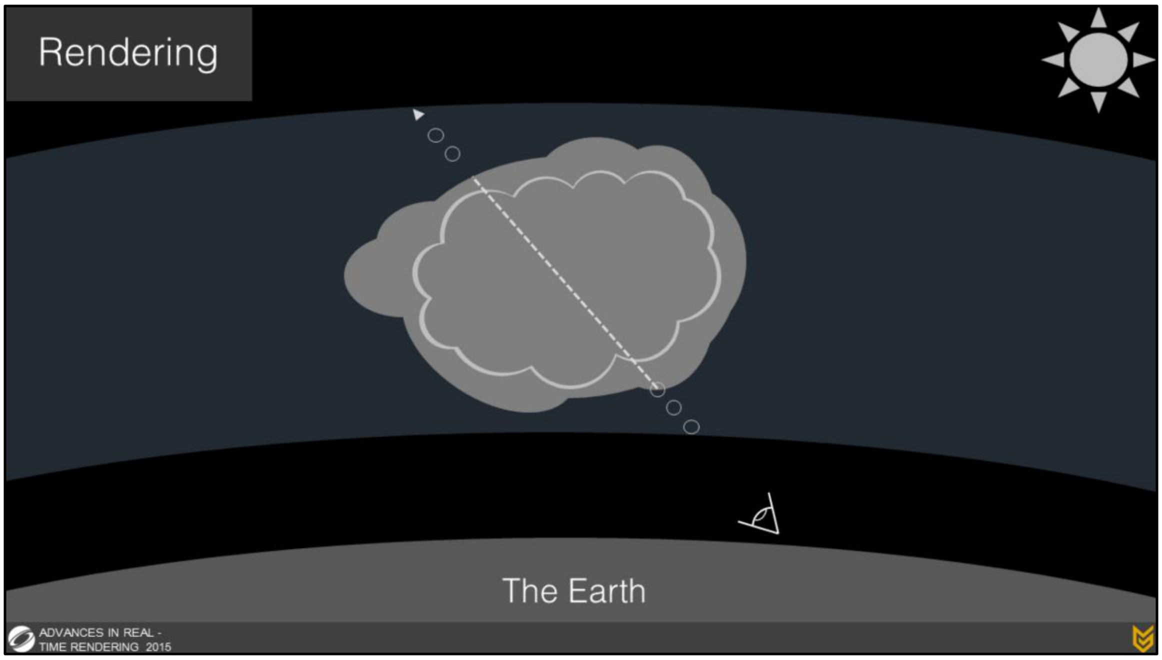


So, when we take samples through the atmosphere, we do these cheaper samples at a larger step size until we hit a cloud iso surface. Then we switch to full samples with the high detail noise and all of its associated instructions. To make sure that we do not miss any high res samples, we always take a step backward before switching to high detail samples.



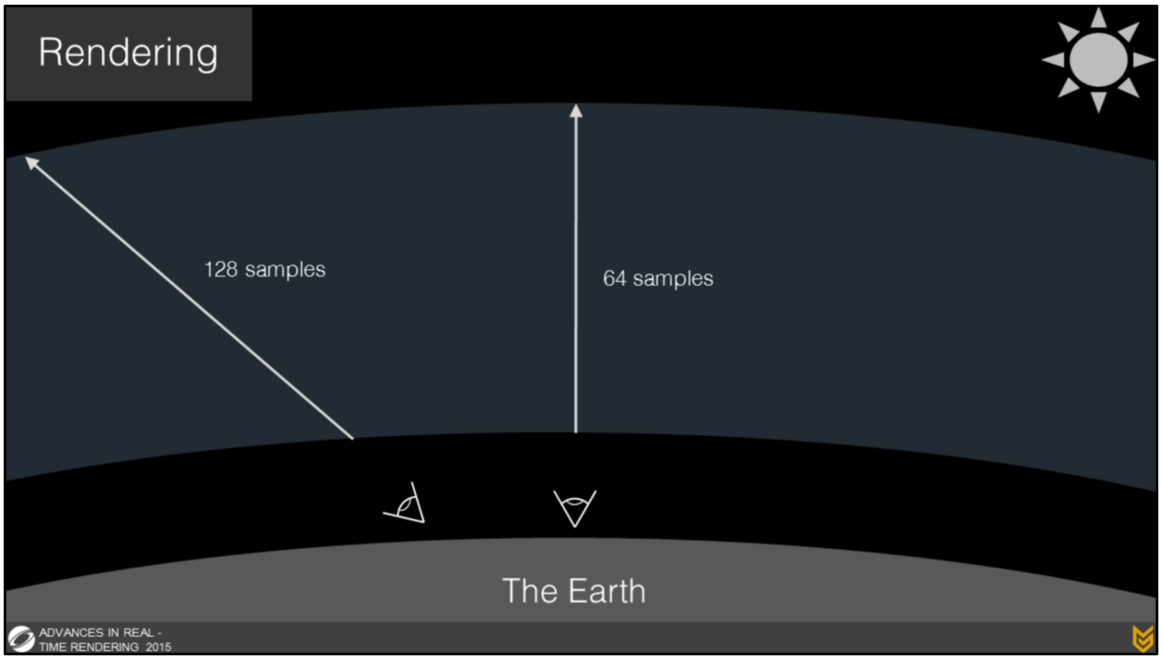
Rendering

- Once the alpha of the image reaches 1 we don’t need to keep sampling so we stop the march early.



If we don’t reach an alpha of one we have another optimization.

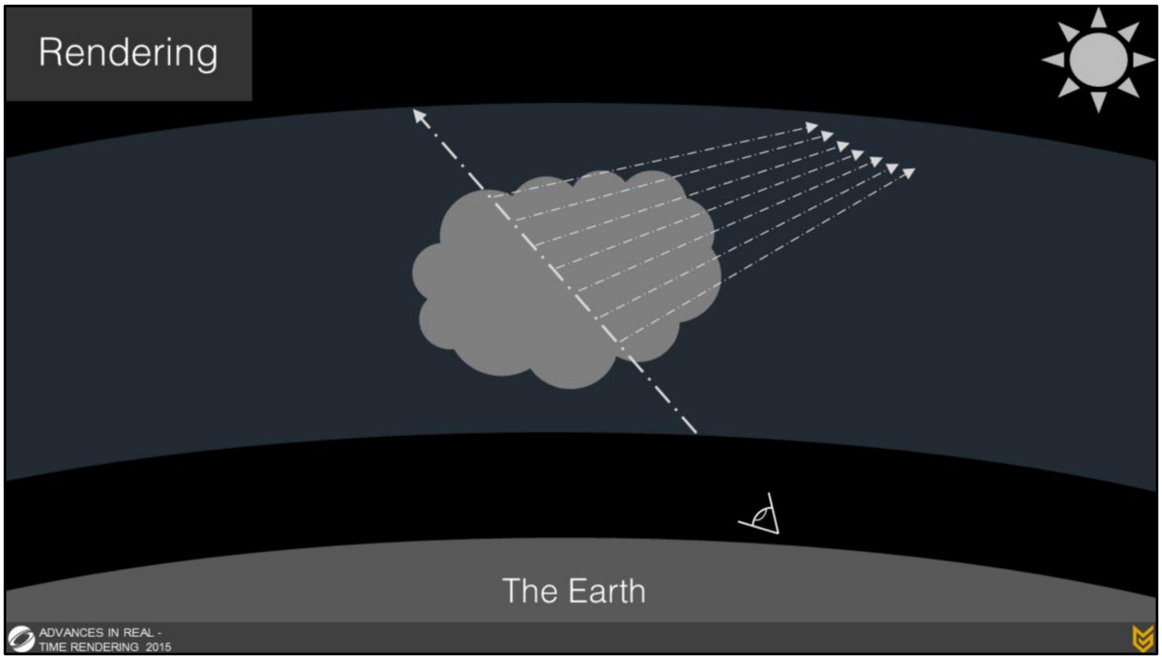
- After several consecutive samples that return zero density, we switch back to the cheap march behavior until we hit something again or reach the top of the cloud layer.



Because of the fact that the ray length increases as we look toward the horizon, we start with …

- an initial potential 64 samples and end with a …
- potential 128 at the horizon. I say potential because of the optimizations which can cause the march to exit early. And we really hope they do.

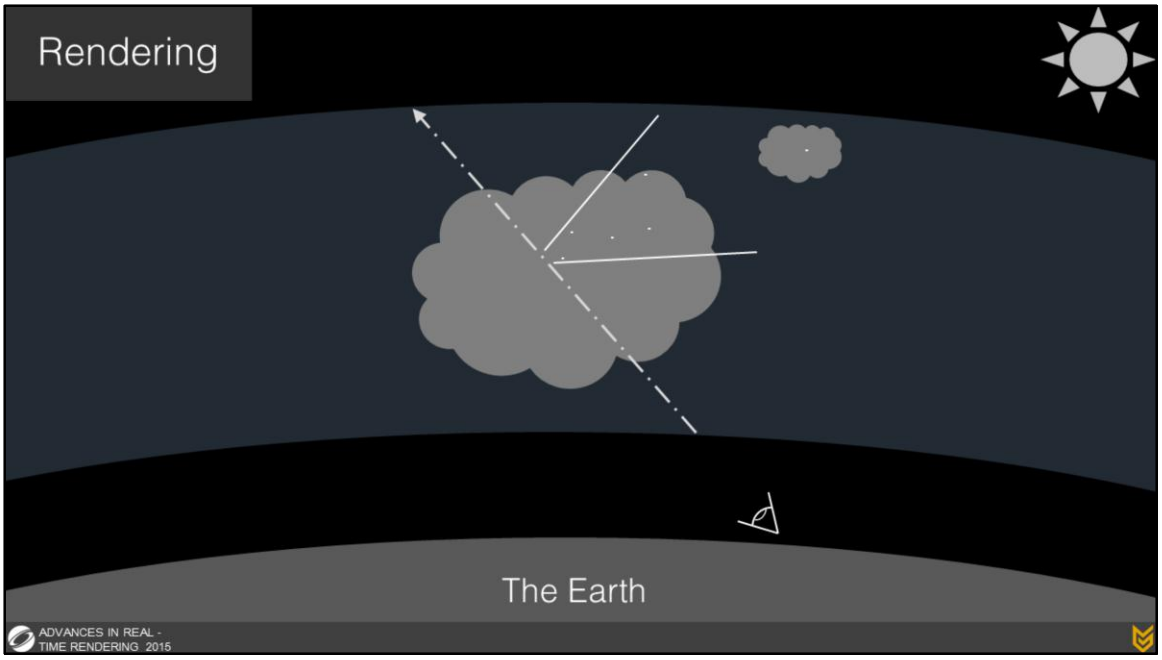
This is how we take the samples to build up the alpha channel of our image. To calculate light intensity we need to take more samples.



Rendering

Normally what you do in a ray march like this is to …

- take samples toward the light source, plug the sum into your lighting equation and then attenuate this value using the alpha channel until you hopefully exit the march early because your alpha has reached 1.



In our approach, we sample 6 times in a cone toward the sun. This smooth's the banding we would normally get with 6 simples and weights our lighting function with neighboring density values, which creates a nice ambient effect. The last sample is placed far away from the rest …

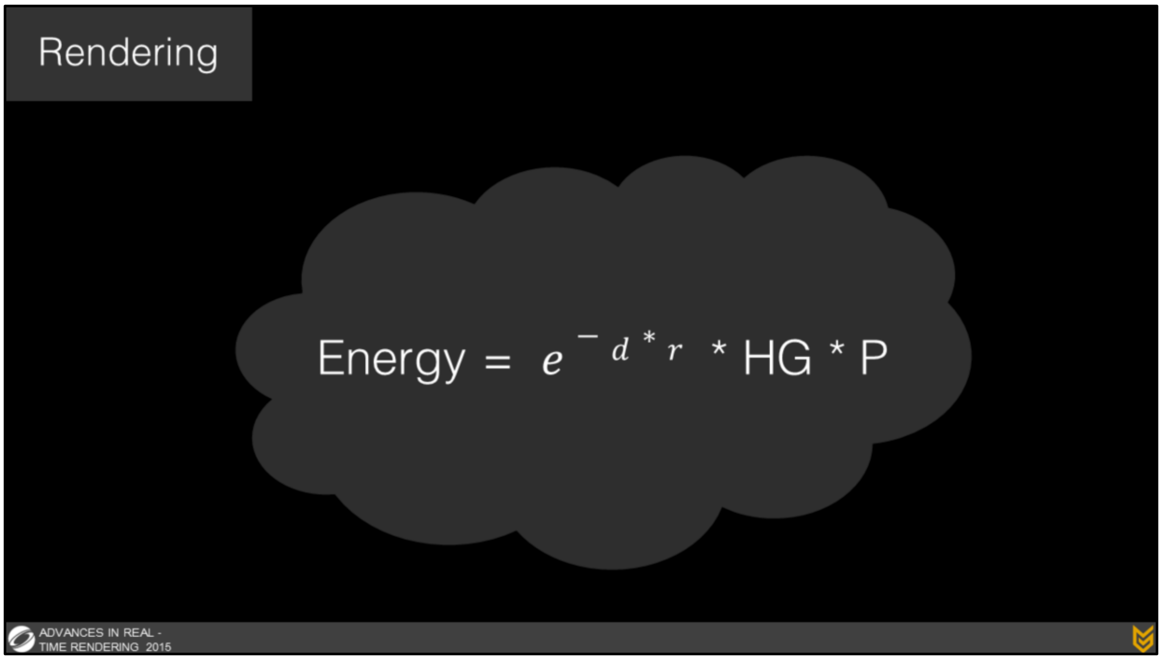
- in order to capture shadows cast by distant clouds.

Rendering
ADVANCES IN REAL - TIME RENDERING 2015

Here you can see what our clouds look like with just alpha samples….

- With our 5 cone samples for lighting.
- And the long distance cone sample.

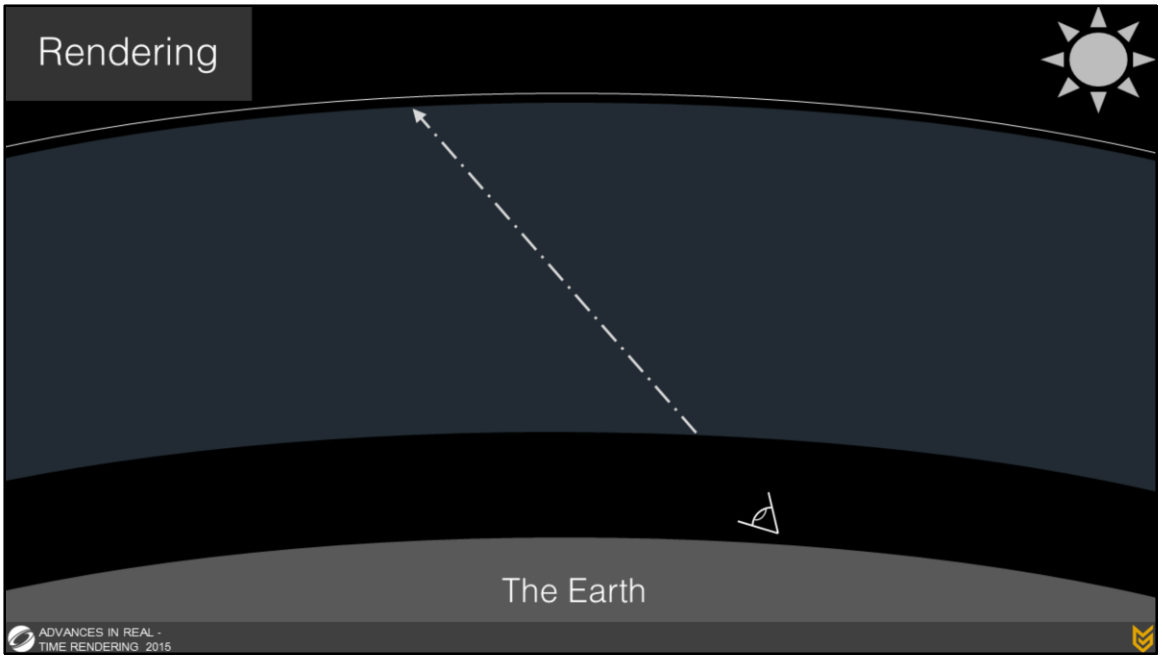
To improve performance of these light samples, we switched to sampling the cheap version of our shader once the alpha of the image reached 0.3. , this made the shader 2x faster



Rendering

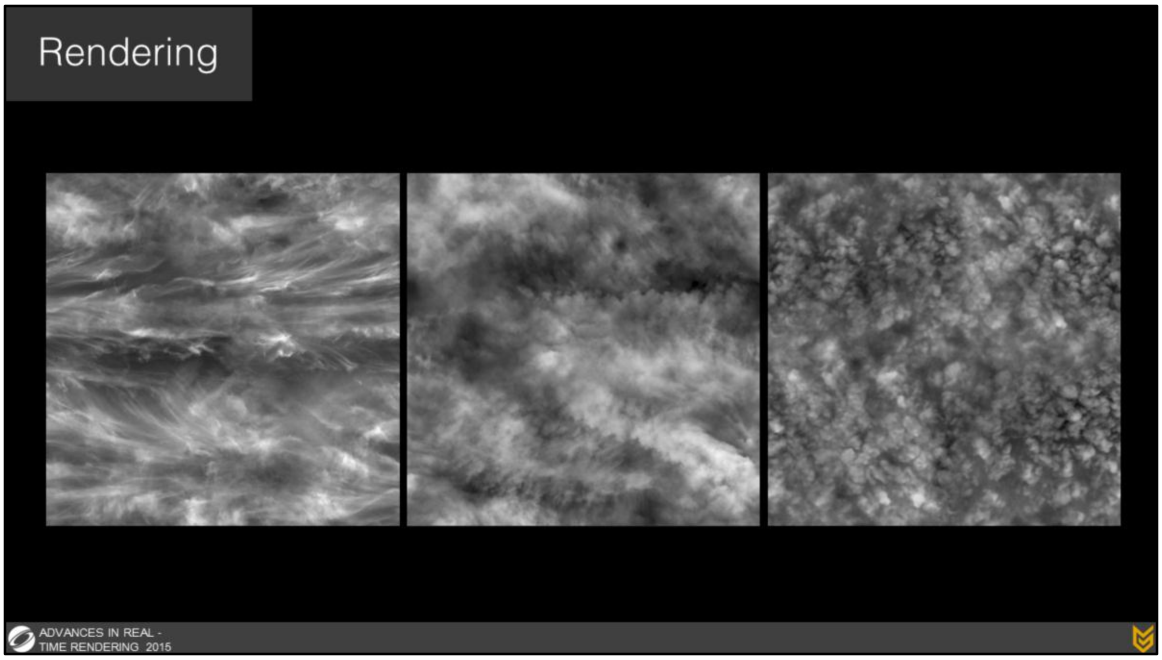
$$\text{Energy} = e^{-d * r} * \text{HG} * \text{P}$$

The lighting samples replace the lower case d, or depth in the beers law portion of our lighting model. This energy value is then attenuated by the depth of the sample in the cloud to produce the image as per the standard volumetric ray-marching approach.

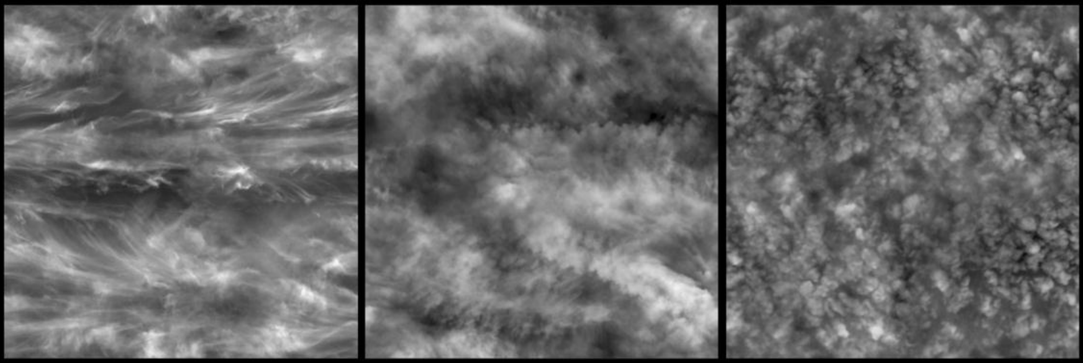


The last step of our ray march was to…

- sample the 2d cloud textures for the high altitude clouds

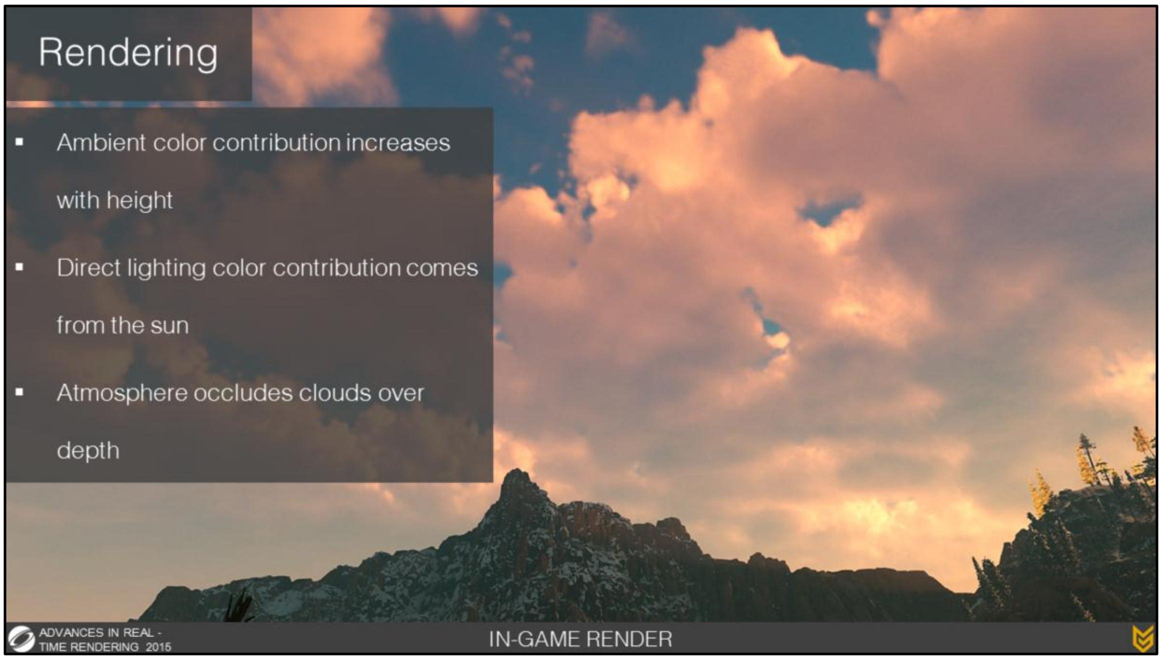


Rendering



ADVANCES IN REAL - TIME RENDERING 2015

These were a collection of the various types of cirrus and alto clouds that were tiling and scrolling at different speeds and directions above the volumetric clouds.



Rendering

- Ambient color contribution increases with height
- Direct lighting color contribution comes from the sun
- Atmosphere occludes clouds over depth

ADVANCES IN REAL-TIME RENDERING 2015

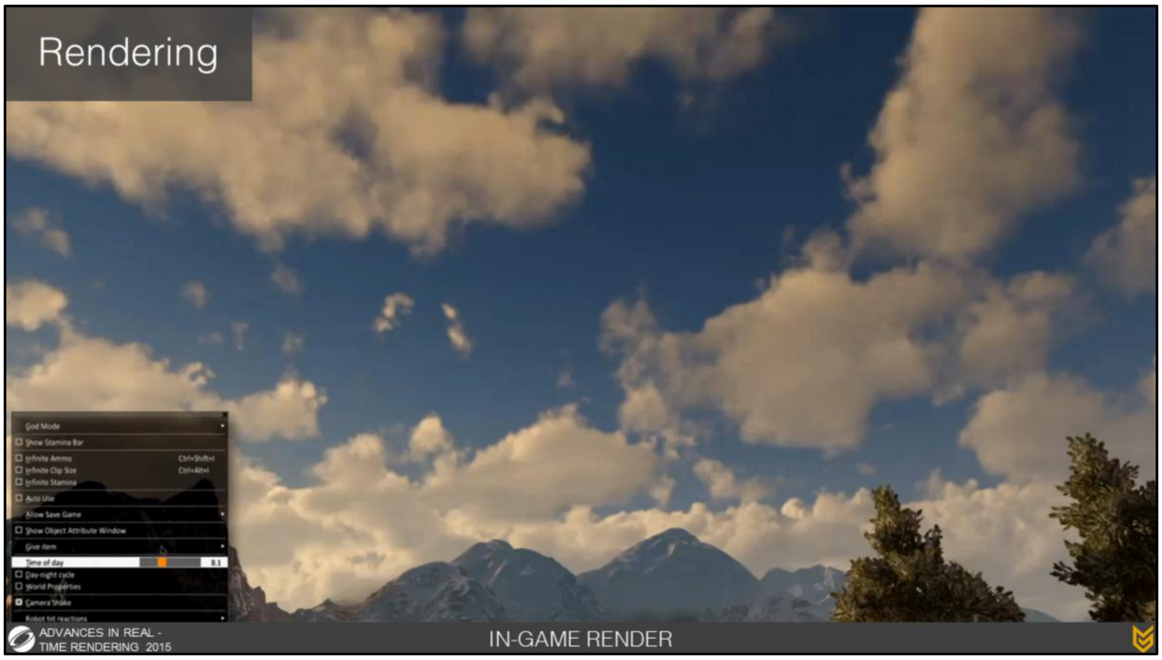
IN-GAME RENDER

In reality light rays of different frequencies are mixing in a cloud producing very beautiful color effects. Since we live in a world of approximations, we had to base cloud colors on some logical assumptions.

We color our clouds based on the following model:

- Ambient sky contribution increases over height
- Direct lighting would be dominated by the sun color
- Atmosphere would occlude clouds over depth.

We add up our ambient and direct components and attenuate to the atmosphere color based on the depth channel.

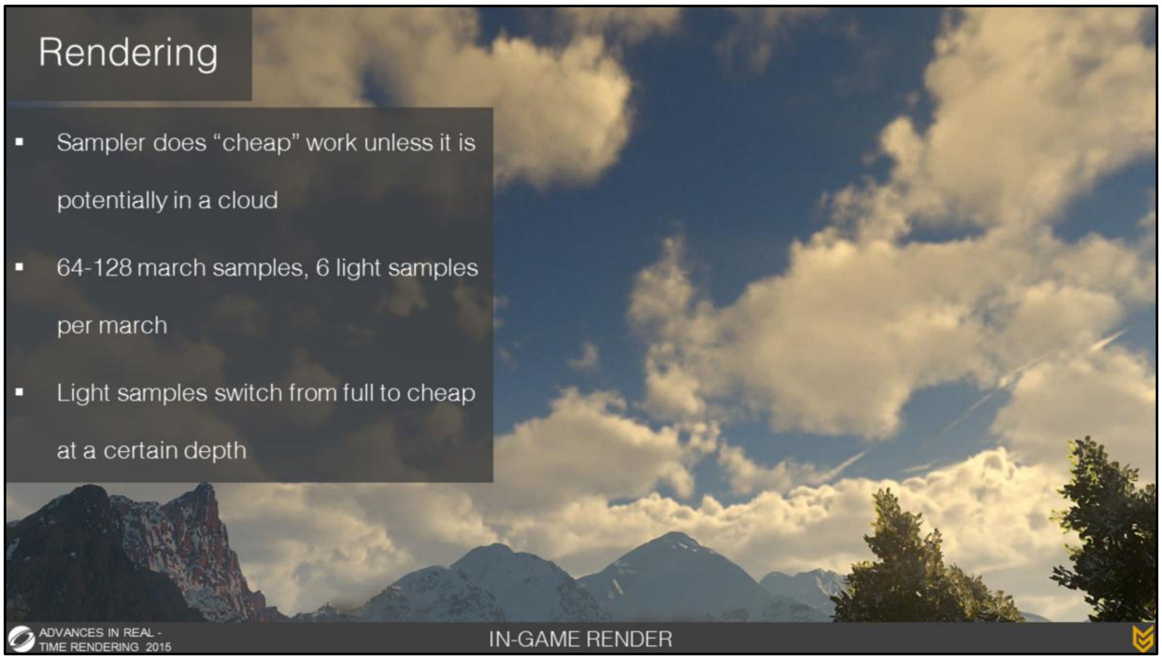


Rendering

ADVANCES IN REAL - TIME RENDERING 2015

IN-GAME RENDER

Now, you can change the time of day in the game and the lighting and colors update automatically. This means no pre-baking and our unique memory usage for the entire sky is limited to the cost of 2 3d textures and 1 2d texture instead of dozens of billboards or sky domes.



Rendering

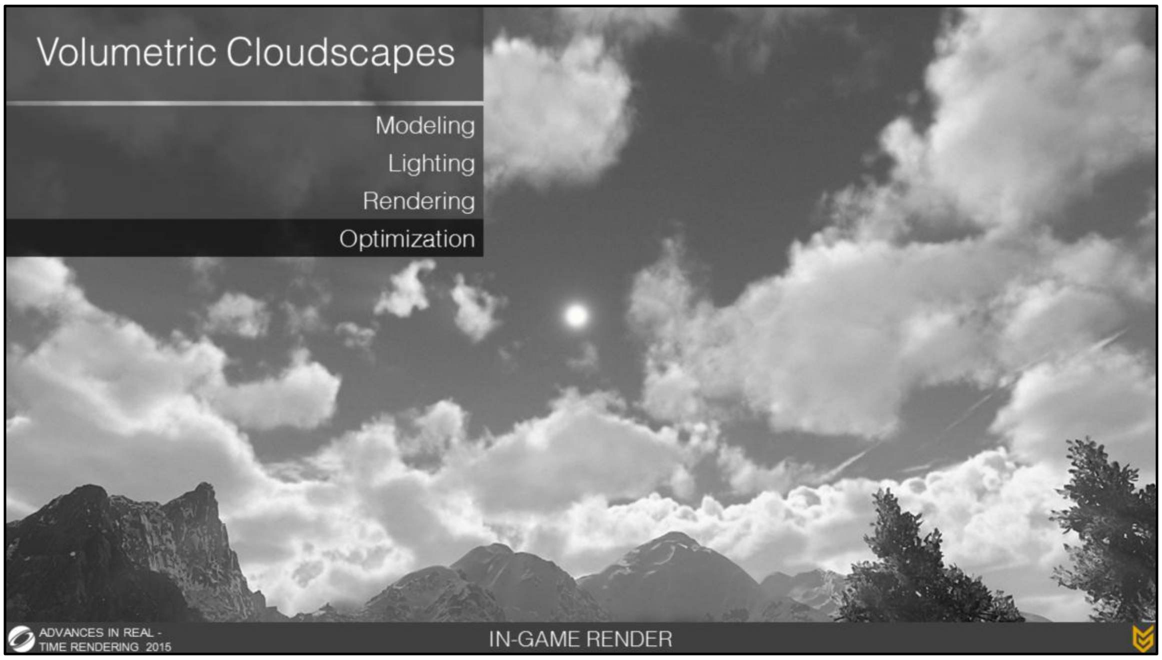
- Sampler does "cheap" work unless it is potentially in a cloud
- 64-128 march samples, 6 light samples per march
- Light samples switch from full to cheap at a certain depth

ADVANCES IN REAL - TIME RENDERING 2015

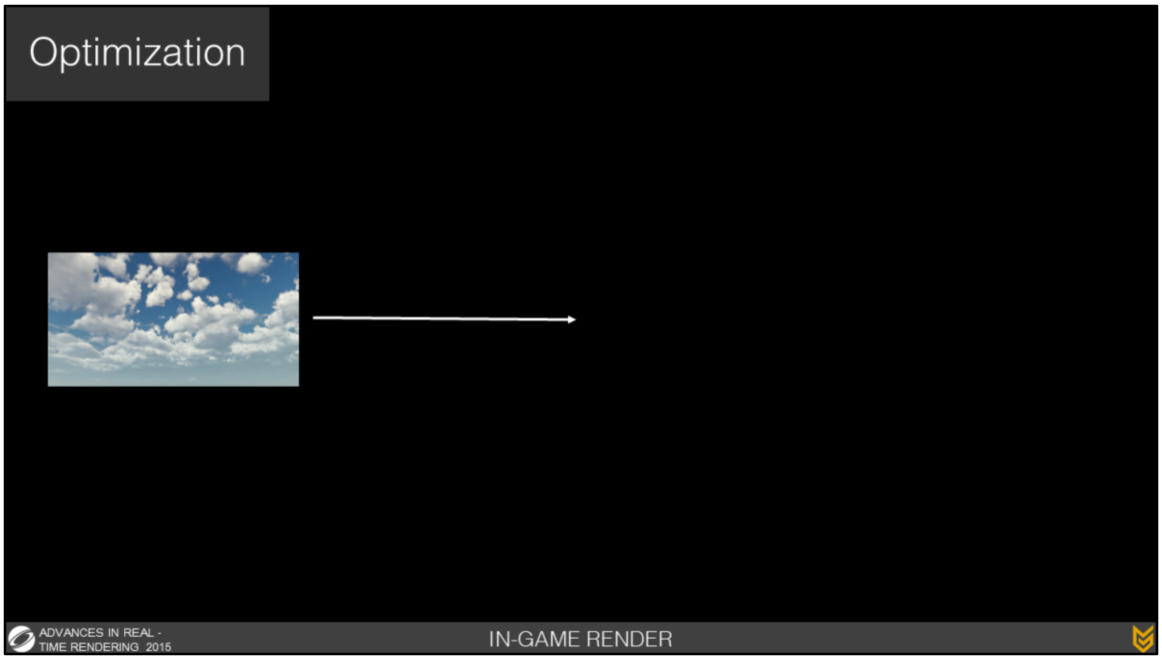
IN-GAME RENDER

To sum up what makes our rendering approach unique:

- Sampler does "cheap" work unless it is potentially in a cloud
- 64-128 potential march samples, 6 light samples per march in a cone, when we are potentially in a cloud.
- Light samples switch from full to cheap at a certain depth



The approach that I have described so far costs around 20 milliseconds.
(pause for laughter)
Which means it is pretty but, it is not fast enough to be included in our game. My co-developer and mentor on this, Nathan Vos, Had the idea that…



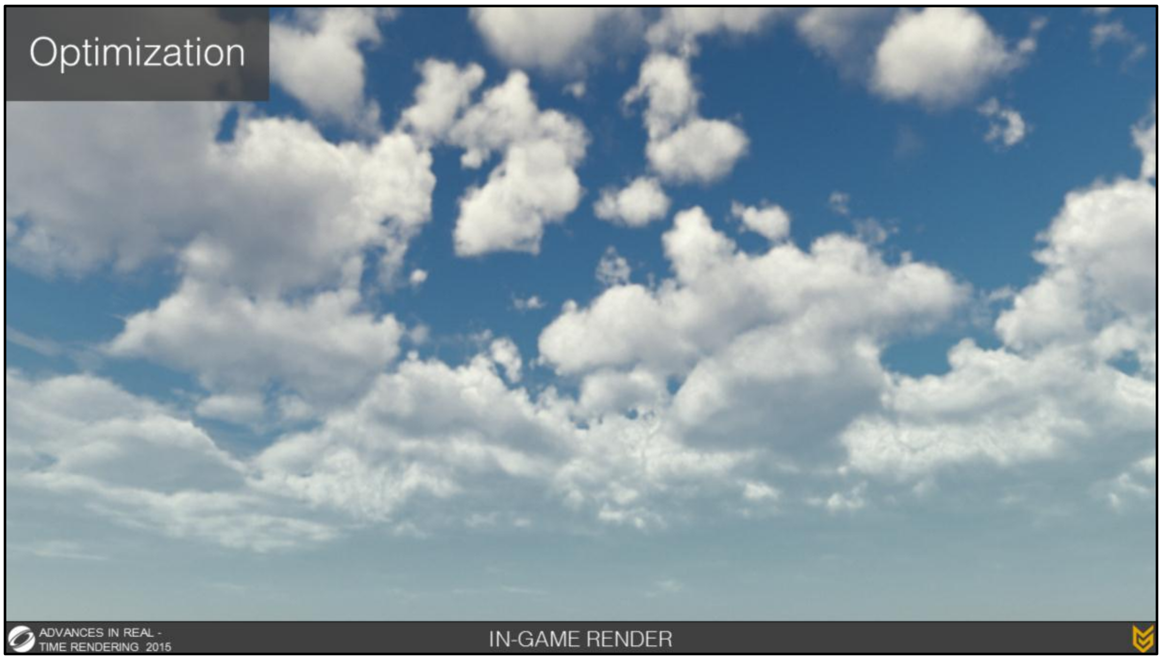
Optimization

ADVANCES IN REAL - TIME RENDERING 2015

IN-GAME RENDER

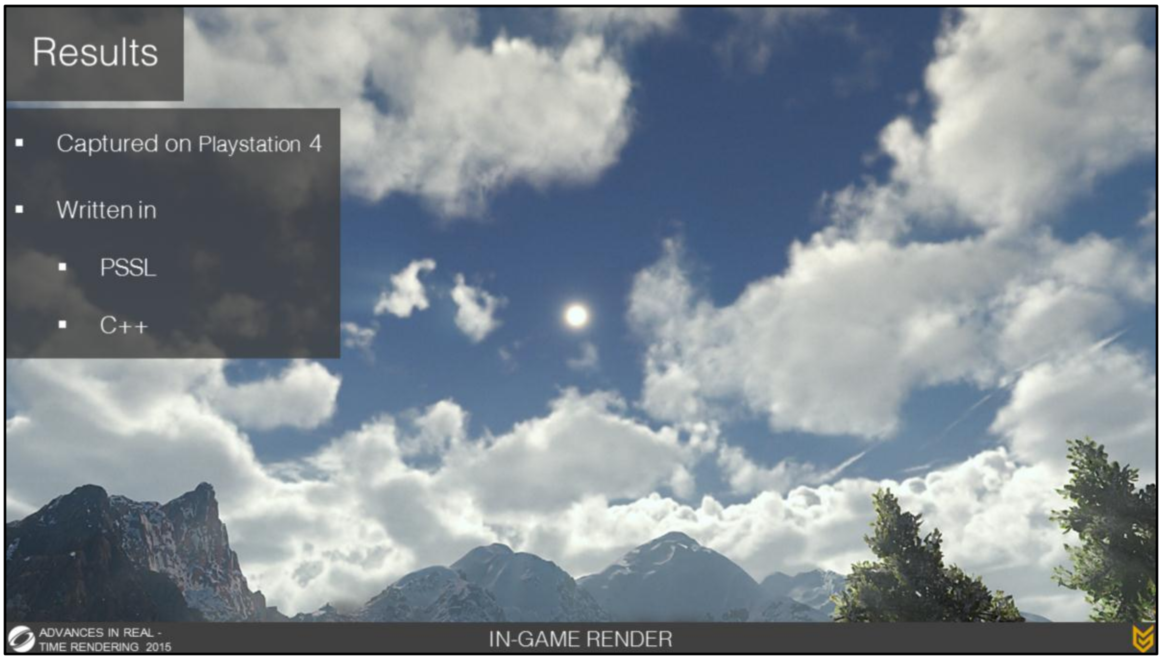
- Every frame we could use a quarter res buffer…
- to update 1 out of 16 pixels for each 4x4 pixel bl.ock within our final image.

We reproject the previous frame to ensure we have something persistent.



…and where we could not reproject, like the edge of the screen, We substitute the result from one of the low res buffers.

Nathan's idea made the shader 10x faster or more when we render this at half res and use filters to upscale it.
It is pretty much the whole reason we are able to put this in our game. Because of this our target performance is around 2 milliseconds, most of that coming from the number of instructions.

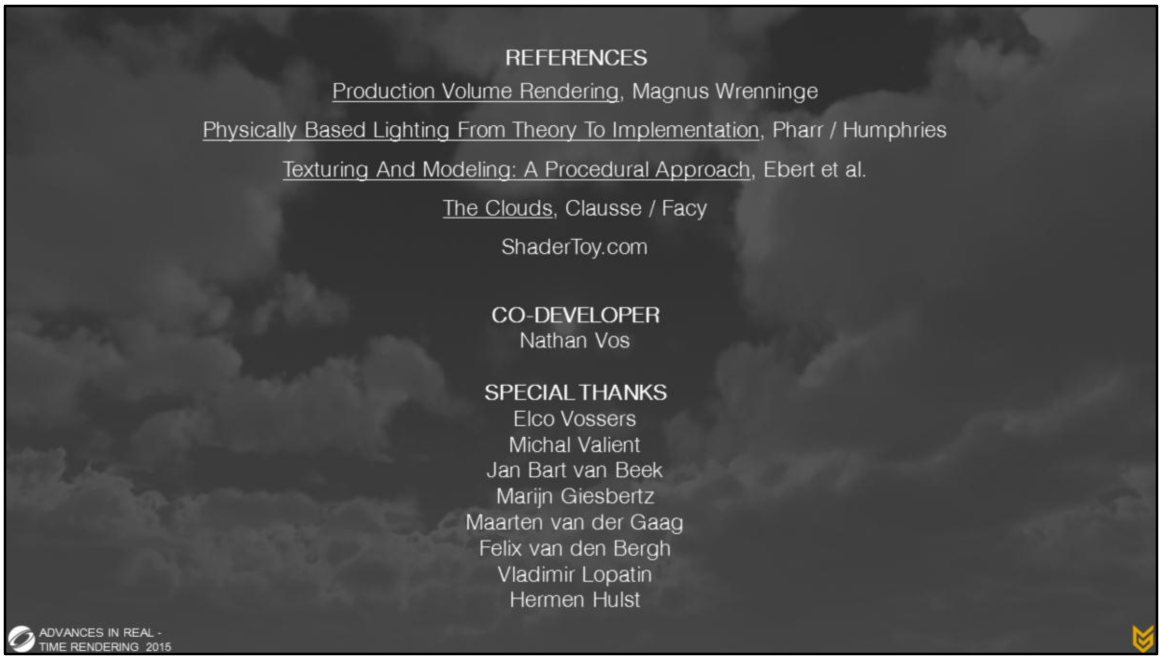


Results

- Captured on Playstation 4
- Written in
  - PSSL
  - C++

In review we feel that
We have largely achieved our initial goals. This is still a work in progress as there is still time left in the production cycle so we hope to improve performance and direct-ability a bit more. We're also still working on our atmospheric model and weather system and we will be sharing more about this work in the future on our website and at future conferences.

- All of this was captured on a playstation 4
- And this solution was written in PSSL and C++



REFERENCES

Production Volume Rendering, Magnus Wrenninge

Physically Based Lighting From Theory To Implementation, Pharr / Humphries

Texturing And Modeling: A Procedural Approach, Ebert et al.

The Clouds, Clausse / Facy

ShaderToy.com

CO-DEVELOPER

Nathan Vos

SPECIAL THANKS

Elco Vossers
Michal Valient
Jan Bart van Beek
Marijn Giesbertz
Maarten van der Gaag
Felix van den Bergh
Vladimir Lopatin
Hermen Hulst

ADVANCES IN REAL - TIME RENDERING 2015

- A number of sources were utilized in the development of this system. I have listed them here.
- I would like to thank My co-developer, Nathan vos most of all

Also some other Guerrillas..
Elco – weather system and general help with transition to games
Michal – supervising the shader development with me and Nathan
Jan Bart, - for keeping us on target with our look goals
Marijn – for allowing me the time in the fx budget to work on this and for his guidance
Maarten van der Gaag for some optimization ideas
Felix van den Bergh for slaving away at making polygon clouds and voxel clouds in the early days
Vlad Lapotin, for his work testing out spherical harmonics
And to Hermen Hulst, manager of Guerrilla for hiring me and for allowing us the resources and time to properly solve this problem for real-time.

Are there any questions?

Peace out.