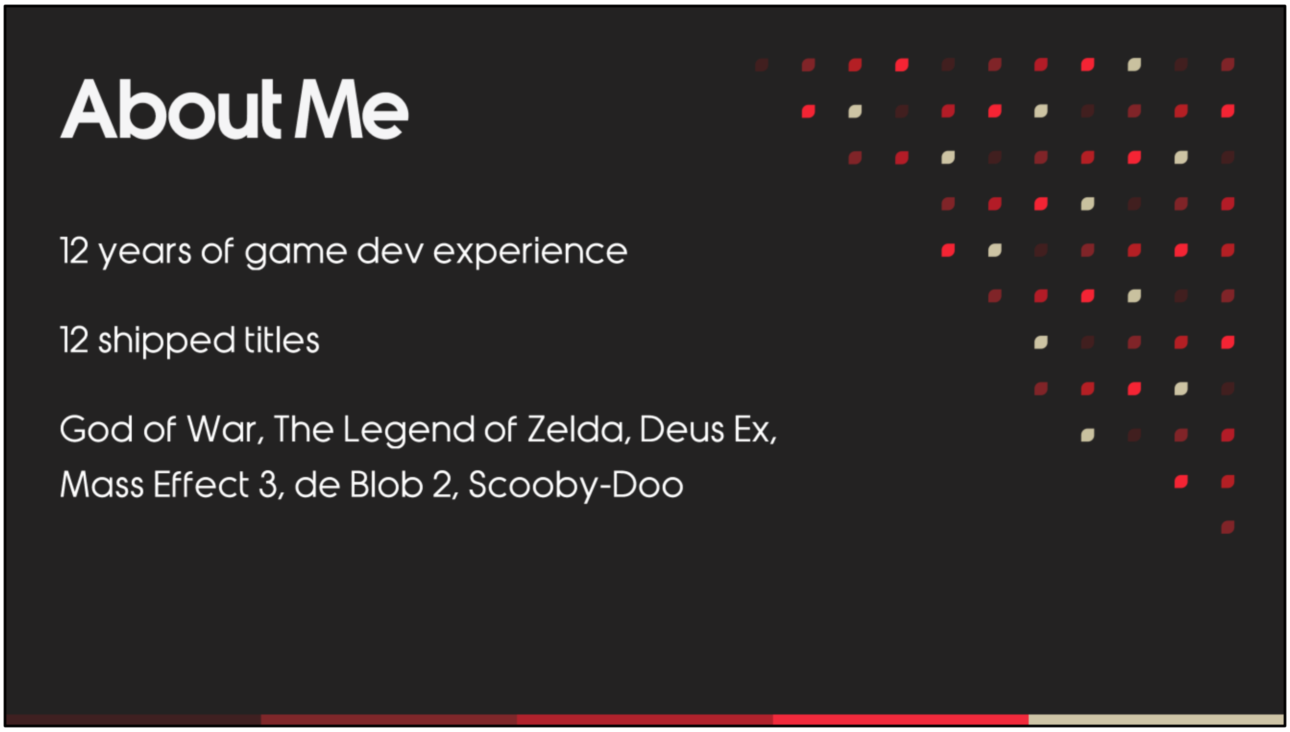
Santa Monica Studio™

Disintegrating Meshes

Rupert Renard



About Me

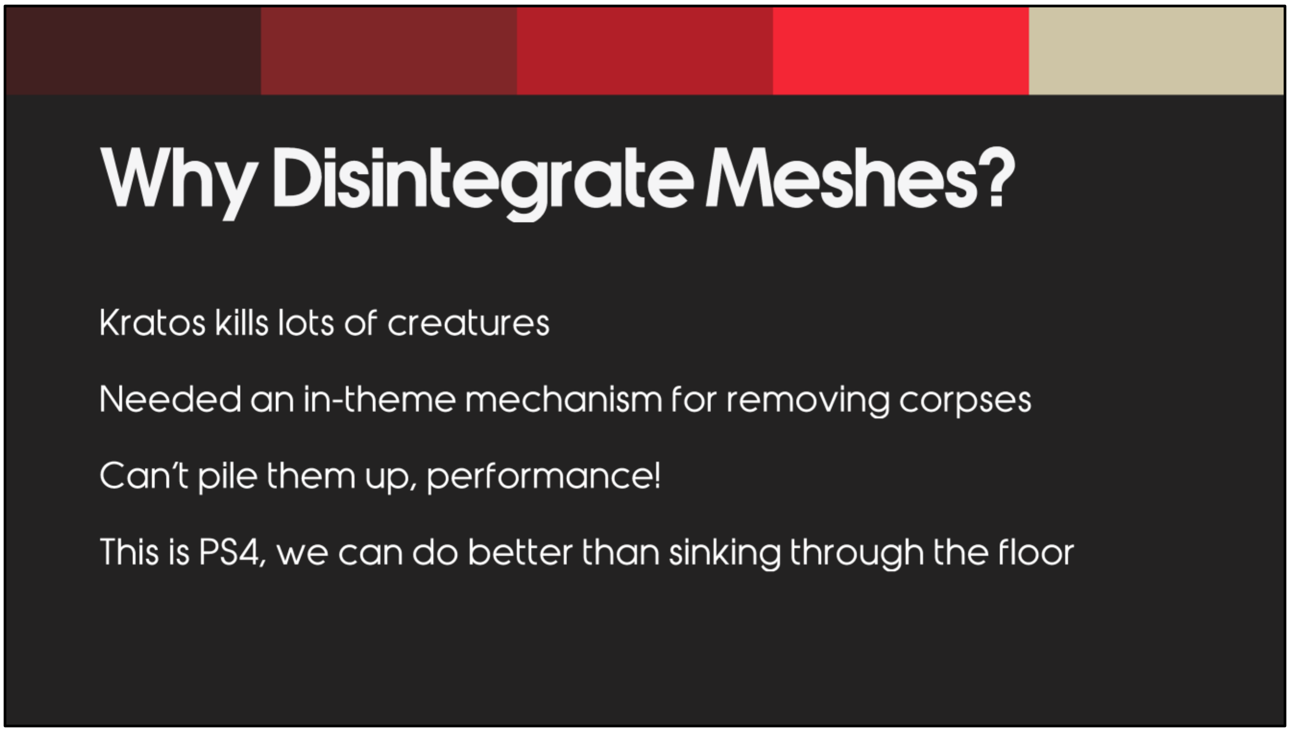
12 years of game dev experience

12 shipped titles

God of War, The Legend of Zelda, Deus Ex, Mass Effect 3, de Blob 2, Scooby-Doo

[00:00] [31 seconds]

Welcome to my talk.
My name is Rupert Renard.
I’m an Australian game developer.
I’ve been programming games for over 12 years now.
I’ve worked on 12 shipped titles, and half a dozen cancelled titles.
Some of the games I’ve worked on you may have heard about, such as: God of War, The Legend of Zelda, Deus Ex, Mass Effect 3, de Blob 2, and Scooby-Doo.
I’ve worked in a variety of programming positions.
I’m currently at Sony Santa Monica as a graphics and engine programmer, where we shipped God of War in April 2018, and it did pretty well.



Why Disintegrate Meshes?

Kratos kills lots of creatures

Needed an in-theme mechanism for removing corpses

Can't pile them up, performance!

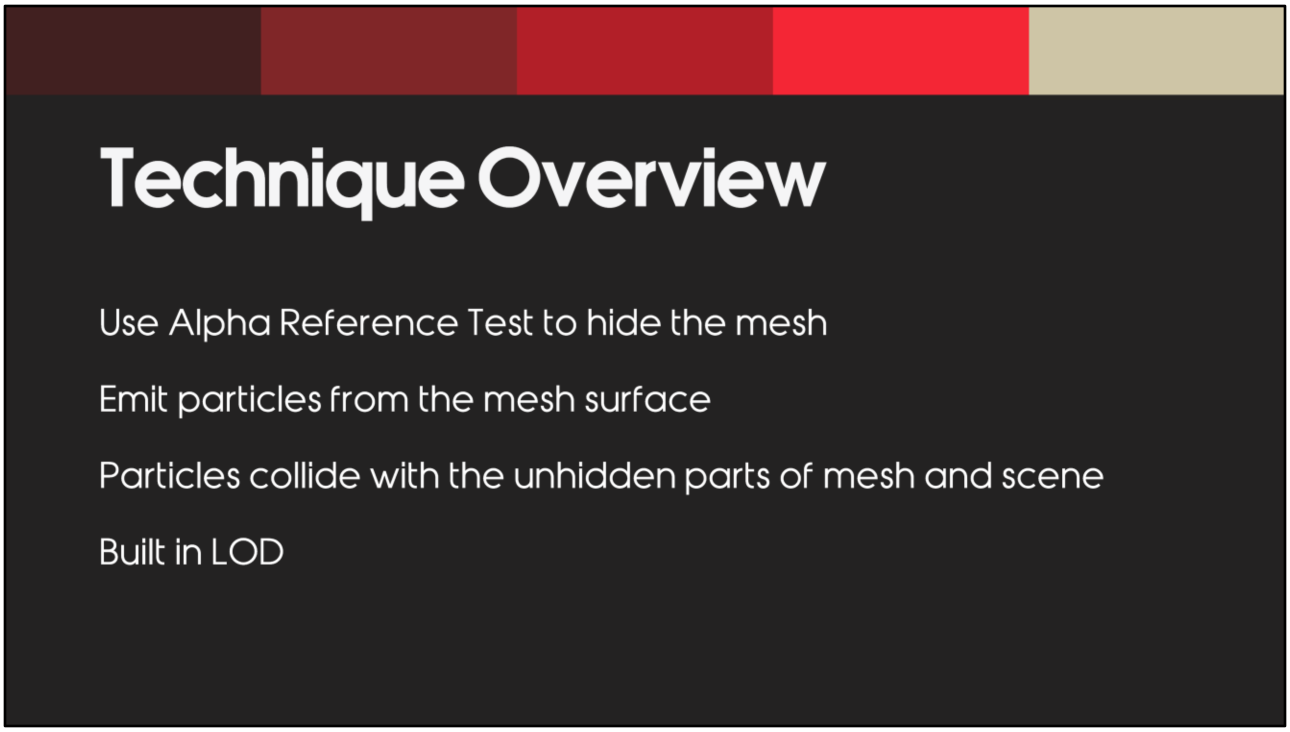
This is PS4, we can do better than sinking through the floor

[00:31] [55 seconds]

In previous God of War games, Kratos would plow through lots of enemies, and the new game wasn't going to differ too much in that aspect.
So back in around mid 2015 we held a meeting to come up with in-theme methods of removing defeated enemies from the screen.
The method needed to fit in to the God of War world.
We couldn't just simply let the pile of corpses stack up.
We needed to remove these bodies to ensure the frame rate wouldn't suffer.
Most games don't really take this in to account.
They usually just do something simple, like let the body sink through the floor.
Or fade the body with alpha blending.
So I proposed a method to disintegrate the body, pixel by pixel.
I quickly prototyped this, the results looked promising, and it got implemented in to the game.

[01:26] [97 seconds]

I've prepared a short demonstration video from the final game to show you what to expect from the technique.
Also note, that some other standard particle emission techniques are used in combination at times.
The disintegration technique can occur pretty quickly at times, so please refrain from blinking.



Technique Overview

Use Alpha Reference Test to hide the mesh

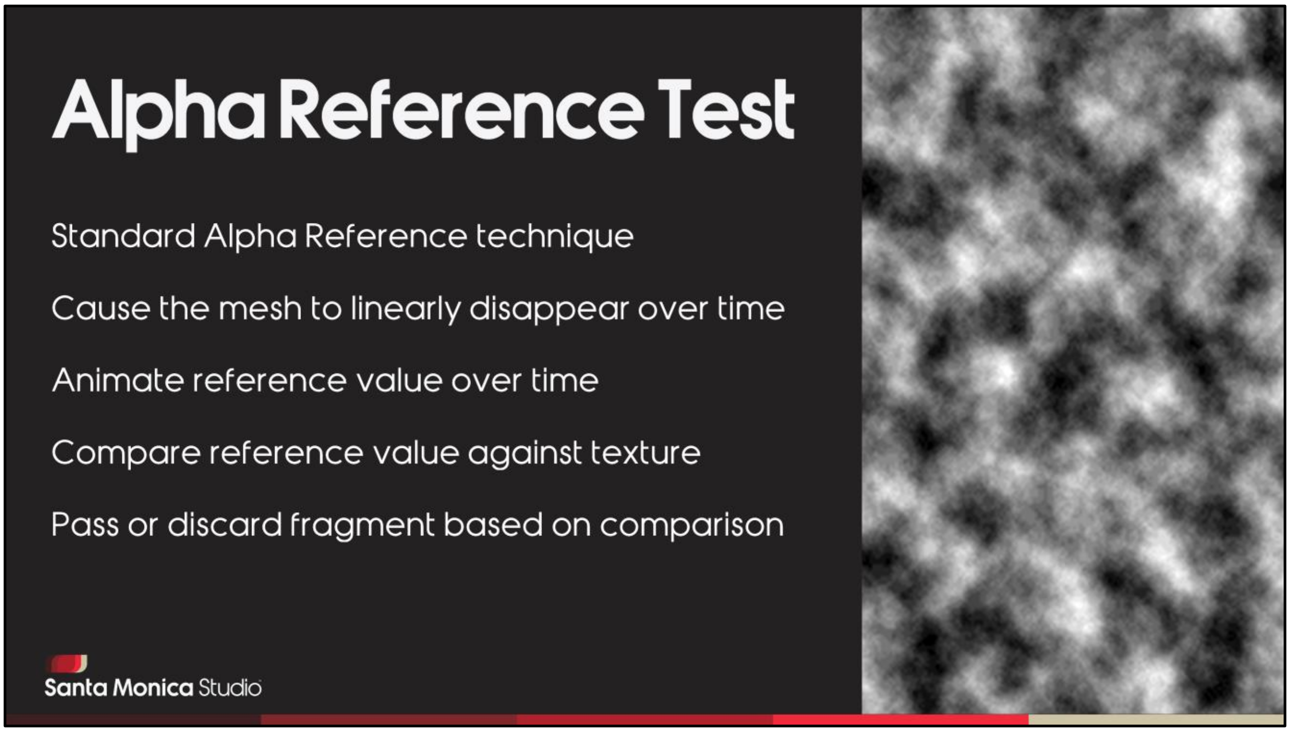
Emit particles from the mesh surface

Particles collide with the unhidden parts of mesh and scene

Built in LOD

[03:03] [45 seconds]

The technique is applied in two major parts.
The first part is to make the mesh disappear.
To do this we apply simple alpha reference testing, nothing really new there.
The second part is to emit particles from the mesh.
We do this by emitting particles the frame immediately before the pixel will be hidden from alpha reference testing.
This gives the illusion that the pixels that make up the mesh are disintegrating in to small particles.
The particles are then able to move and animate independently, as well as collide with the screens depth buffer.
The technique also conveniently has built in level of detail.



Alpha Reference Test

Standard Alpha Reference technique

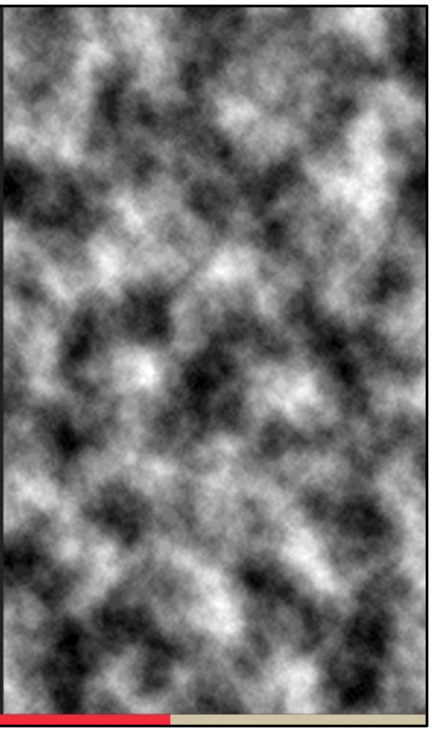
Cause the mesh to linearly disappear over time

Animate reference value over time

Compare reference value against texture

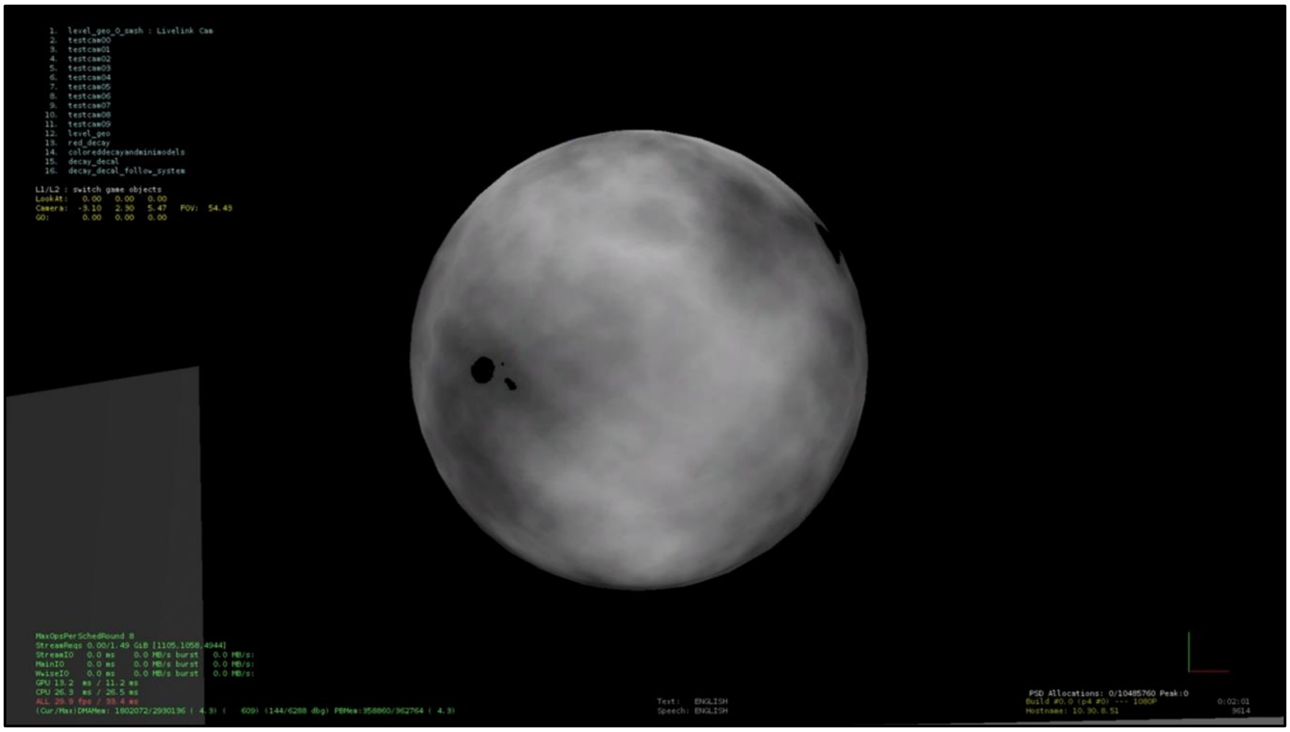
Pass or discard fragment based on comparison

Santa Monica Studio



[03:48] [60 seconds]

We start with a simple alpha reference comparison test.
For each fragment of the mesh being drawn, we sample a single channel texture.
We compare the sample against a reference value.
The reference value is shared globally for the mesh, and changes over time.
The comparison results are used to determine whether or not the fragment is visible.
If it's visible, we continue on in the shader, if it's not visible, we discard the fragment.
Here is a sample texture from one of the enemies in the game, it's obviously a noise texture that's used all over the character in wrap sampling mode.
This texture is used to demonstrate the character decaying away from random parts on the body.
That is to say, its effect is to not decay in a specific manner, let it just be random, and it works quite well.
Other cases may use more specific, tailored textures for certain models or scenarios.



```
 1.  level_geo_0_smsh : Livelink Cam
 2.  testcam00
 3.  testcam01
 4.  testcam02
 5.  testcam03
 6.  testcam04
 7.  testcam05
 8.  testcam06
 9.  testcam07
10.  testcam08
11.  testcam09
12.  level_geo
13.  red_decay
14.  coloreddecayandminimodels
15.  decay_decal
16.  decay_decal_follow_system

L1/L2 : switch game objects
LookAt:    0.00   0.00   0.00
Camera:   -3.10   2.30   5.47   FOV:  54.43
GO:        0.00   0.00   0.00
```

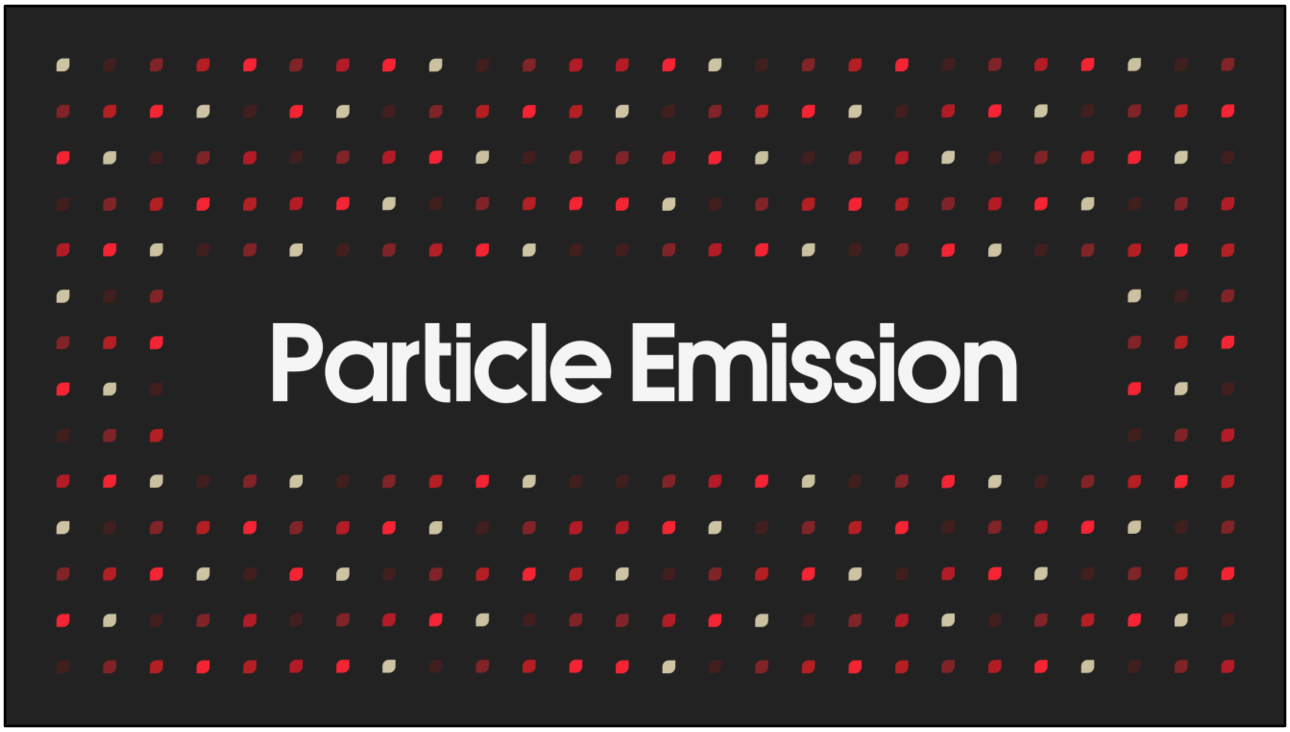
```
MaxOpsPerSchedRound 8
StreamReqs 0.00/1.49 GiB [1105,1058,4944]
StreamIO   0.0 ms    0.0 MB/s burst   0.0 MB/s:
MainIO     0.0 ms    0.0 MB/s burst   0.0 MB/s:
WwiseIO    0.0 ms    0.0 MB/s burst   0.0 MB/s:
GPU 13.2  ms / 11.2 ms
CPU 26.3  ms / 26.5 ms
ALL 29.9 fps / 33.4 ms
(Cur/Max)DMAMem: 1802072/2930136 ( 4.3) (   609) (144/6288 dbg) PBMem:358860/362764 ( 4.3)
```

```
Text:   ENGLISH
Speech: ENGLISH
```

```
PSD Allocations: 0/10485760 Peak:0
Build #0.0 (p4 #0) --- 1080P            0:02:01
Hostname: 10.30.8.51                       3614
```

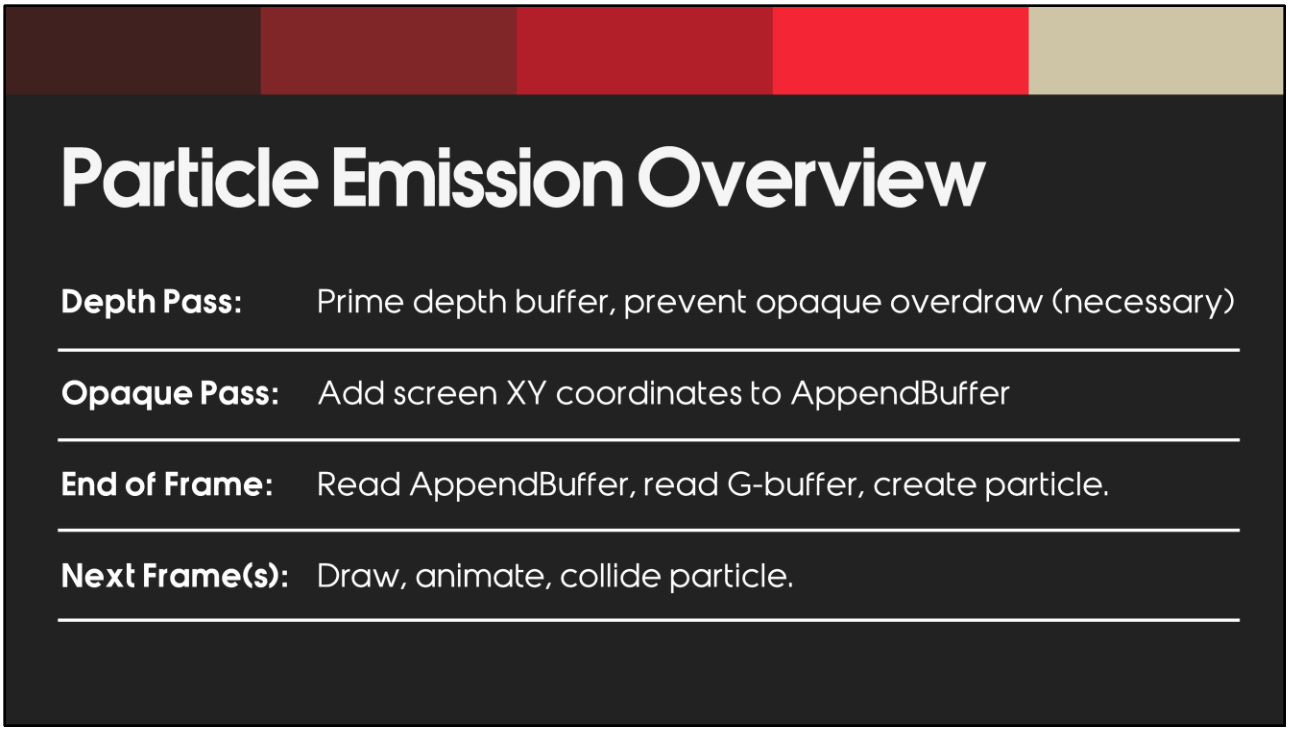
[04:48] [21 seconds]

Here is a demo of an animated alpha reference.
I'm using the same texture for alpha reference as I am for diffuse.
I've done this so you can visually see where the alpha recedes from and to.
You can see the mesh starts disappearing at the dark sections, and moves to the brighter sections as the reference level rises.



Particle Emission

[05:09] [5 seconds]



Particle Emission Overview

Depth Pass:	Prime depth buffer, prevent opaque overdraw (necessary)
Opaque Pass:	Add screen XY coordinates to AppendBuffer
End of Frame:	Read AppendBuffer, read G-buffer, create particle.
Next Frame(s):	Draw, animate, collide particle.

[05:14] [80 seconds]

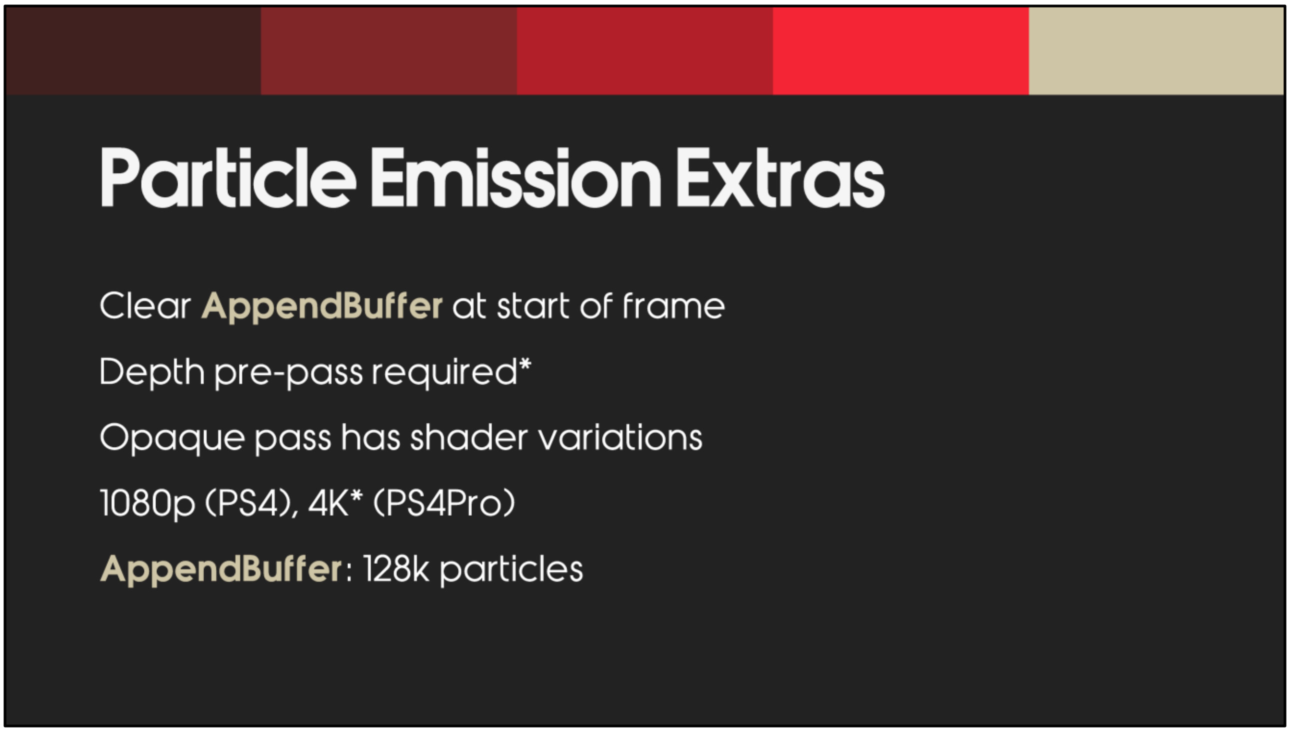
The particle emission is broken down in to generally three phases.

We need to leverage a depth pre-pass.
The depth pre-pass is needed to guarantee we only run one fragment shader per pixel in the opaque pass.
This prevents multiple particles from being emitted from the same pixel if triangles were to overlap.

Once we have populated the depth buffer, we run the opaque pass.
The opaque pass will potentially add to an AppendBuffer the screen coordinates of pixels that are emitting particles for this frame.

Once we have populated the AppendBuffer, later in the frame we're going to read the contents.
We initiate a DispatchIndirect to read the AppendBuffer, and convert pixel coordinates in to proper emitted particles.
This is done by using the screen co-ordinates to lookup information in the G-buffer such as depth, normal, and lighting or other surface information.
While also converting screen co-ordinates plus depth read from the depth buffer in to world space co-ordinates.

The next frame, we’re able to draw the newly emitted particles, and animate, move, and collide all the particles as usual.



Particle Emission Extras

Clear **AppendBuffer** at start of frame

Depth pre-pass required*

Opaque pass has shader variations

1080p (PS4), 4K* (PS4Pro)

AppendBuffer: 128k particles

[06:34] [50 seconds]

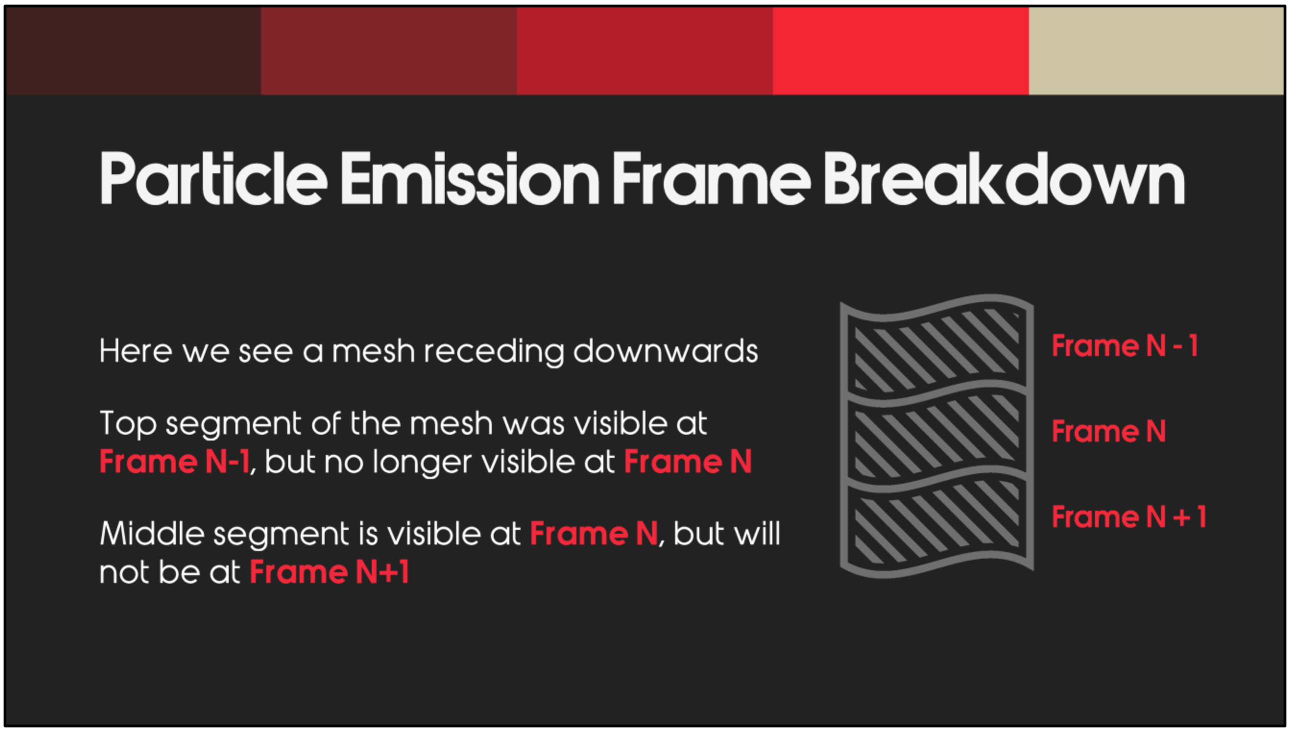
The AppendBuffer will obviously need to be emptied at the start of the frame.

While I say the depth pre-pass is “required”, there are obvious ways around it, you just need to be careful with your drawing and utilization of this technique.

In the opaque pass, shader variations were needed to emit the particles.
We didn’t want the particle emission shader code to be in shaders of materials that never emitted particles, for obvious performance reasons.

We do all this in the native 1080p resolution for the base PS4, or at the 4K checkerboard for PS4Pro.

The AppendBuffer can pack all the information needed, screen co-ordinates etc, in to a single 32-bit entry.
The AppendBuffer has enough storage to emit 128 thousand particles per frame, but it’s unlikely we’ll ever hit that.

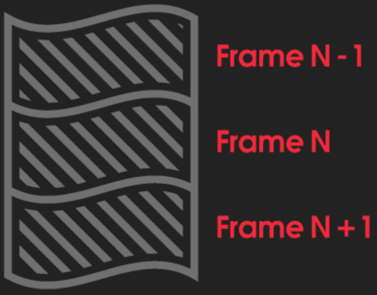


Particle Emission Frame Breakdown

Here we see a mesh receding downwards

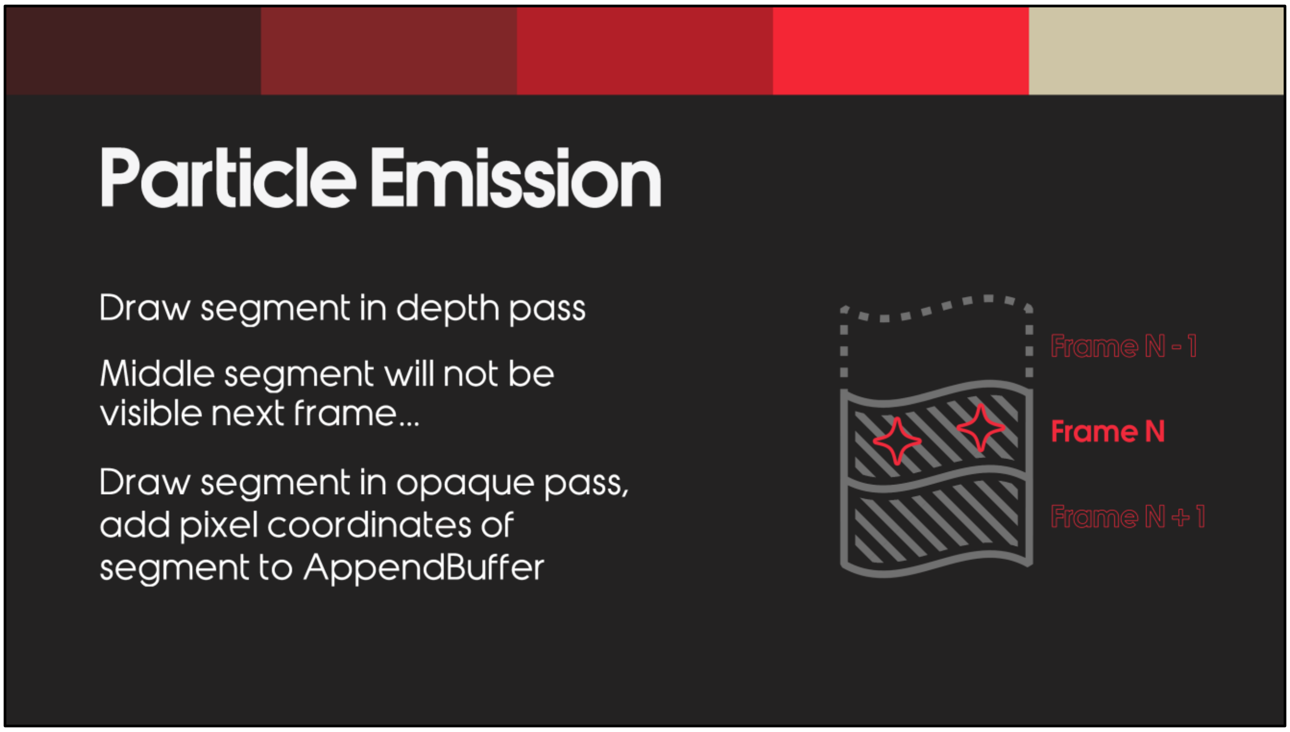
Top segment of the mesh was visible at **Frame N-1**, but no longer visible at **Frame N**

Middle segment is visible at **Frame N**, but will not be at **Frame N+1**



[07:24] [15 seconds]

Here we're going to demonstrate the emission of particles that are coupled with alpha reference test.
Here you can see three segments of a mesh.
Over 3 frames, these three segments will disappear in a cascade from top to bottom.

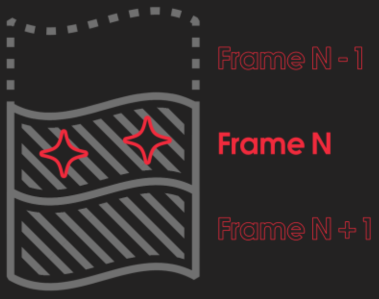


Particle Emission

Draw segment in depth pass

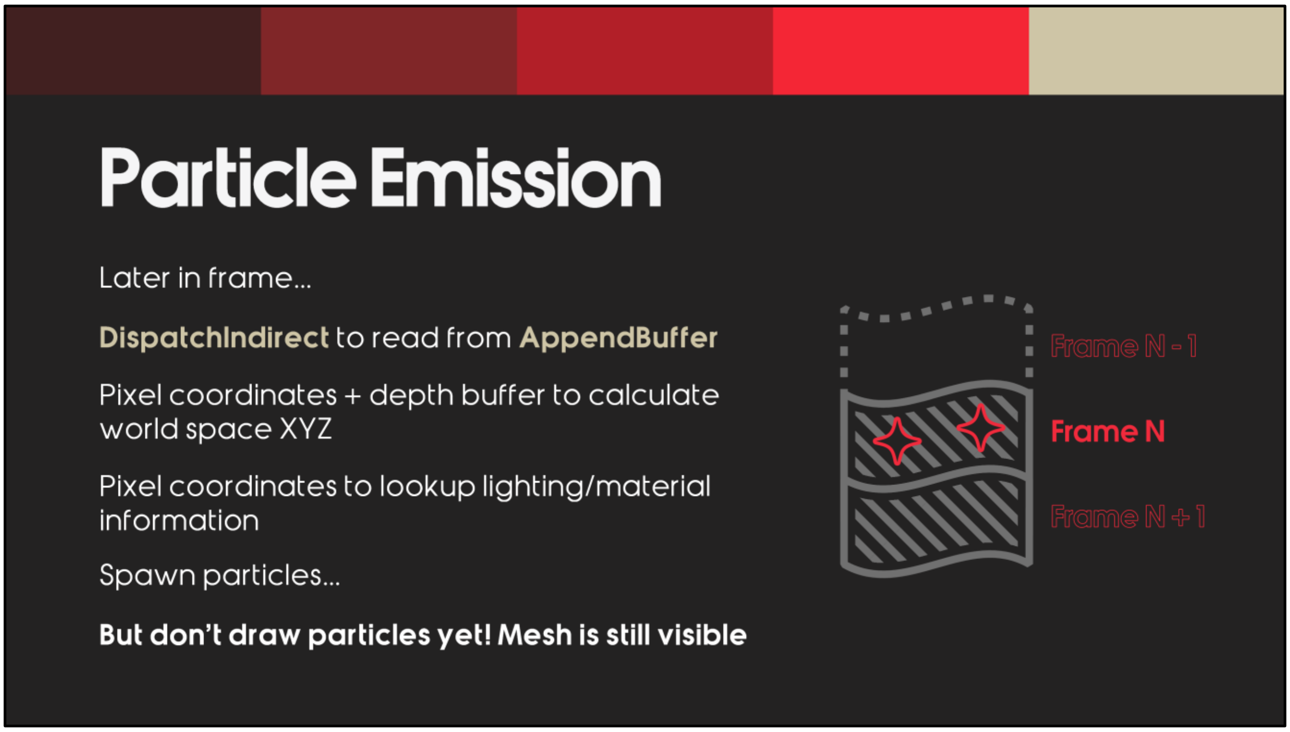
Middle segment will not be visible next frame...

Draw segment in opaque pass, add pixel coordinates of segment to AppendBuffer



[07:39] [45 seconds]

At the start of frame N, we draw the three segments of the mesh.
We start by drawing the segments in the depth pass.
The top segment doesn't pass the reference test, so it executes a discard in the pixel shader of the depth pass.
The other two segments pass the reference test, so they don't discard, and populate the depth buffer.
We have determined the middle segment will not pass the reference test in the next frame, so we need to emit particles this frame to represent the invisible segment in the next frame.
We start by taking the pixel coordinates of each fragment in the middle section, and add them to an AppendBuffer.



Particle Emission

Later in frame…

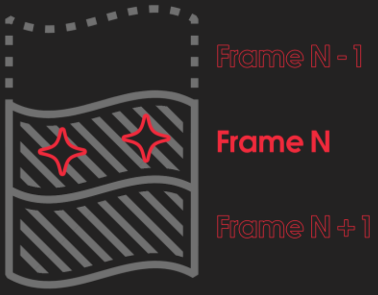
DispatchIndirect to read from **AppendBuffer**

Pixel coordinates + depth buffer to calculate world space XYZ

Pixel coordinates to lookup lighting/material information

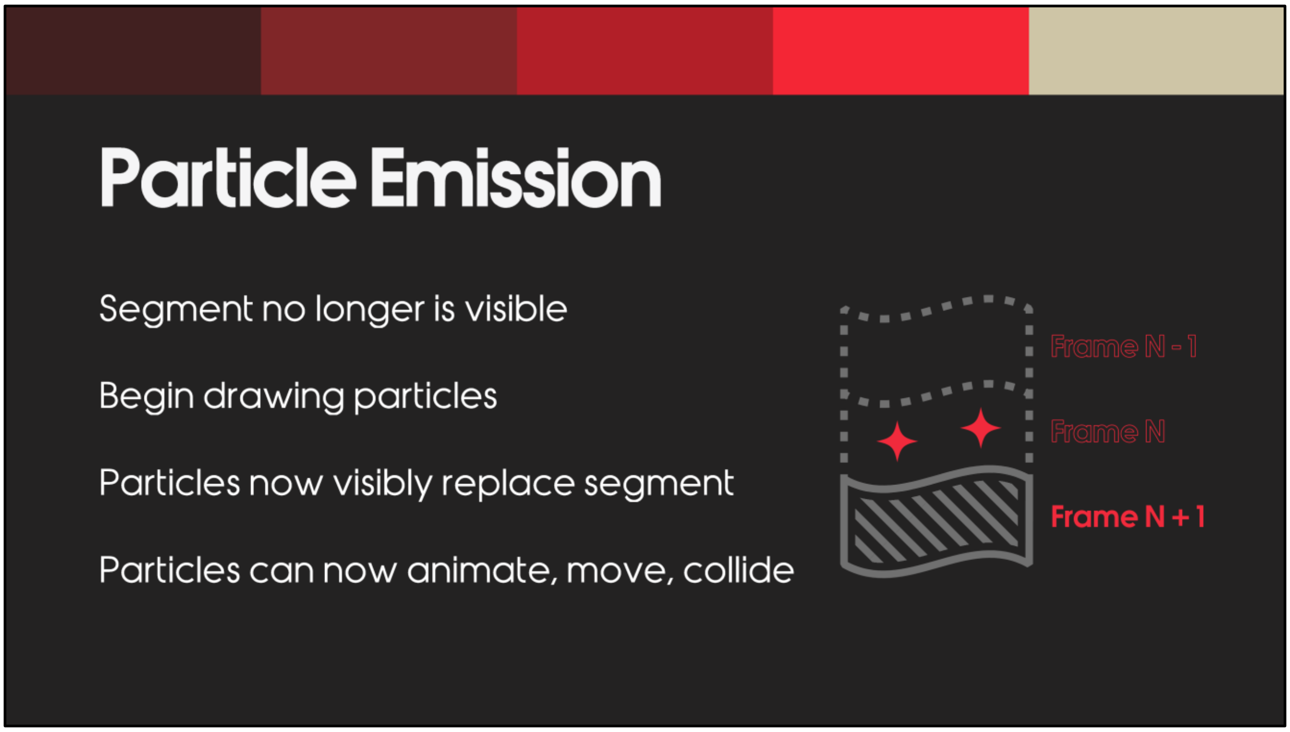
Spawn particles…

But don't draw particles yet! Mesh is still visible



[08:24] [65 seconds]

Later in the frame, we need to read the AppendBuffer full of pixel coordinates, and create particles from them.
We run a shader via DispatchIndirect, in order to process one particle per thread.
In each thread, we read the pixel coordinates linearly from the AppendBuffer, note we don't need to use a ConsumeBuffer here.
Now that we have pixel coordinates, we can index in to the g-buffer, and also the depth buffer.
We read the depth buffer, and can now combine the pixel coordinates with the depth value in to world space coordinates.
We use the world space coordinates as the particles spawning position.
This is also a great opportunity to read other attributes that may be used by the particle, such as normal from the normal buffer, or material properties, or final lighting values of the pixel.
But make note, the particles we're spawning in this frame are definitely NOT to be drawn this frame.
The reason being, the mesh is still visible!
No point drawing particles on top of the mesh, since the mesh fragments are supposed to turn INTO the particles.



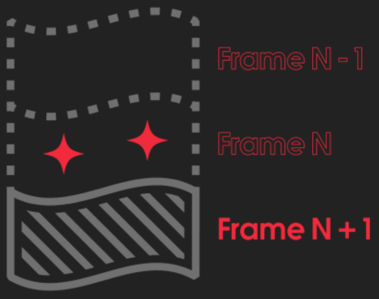
Particle Emission

Segment no longer is visible

Begin drawing particles

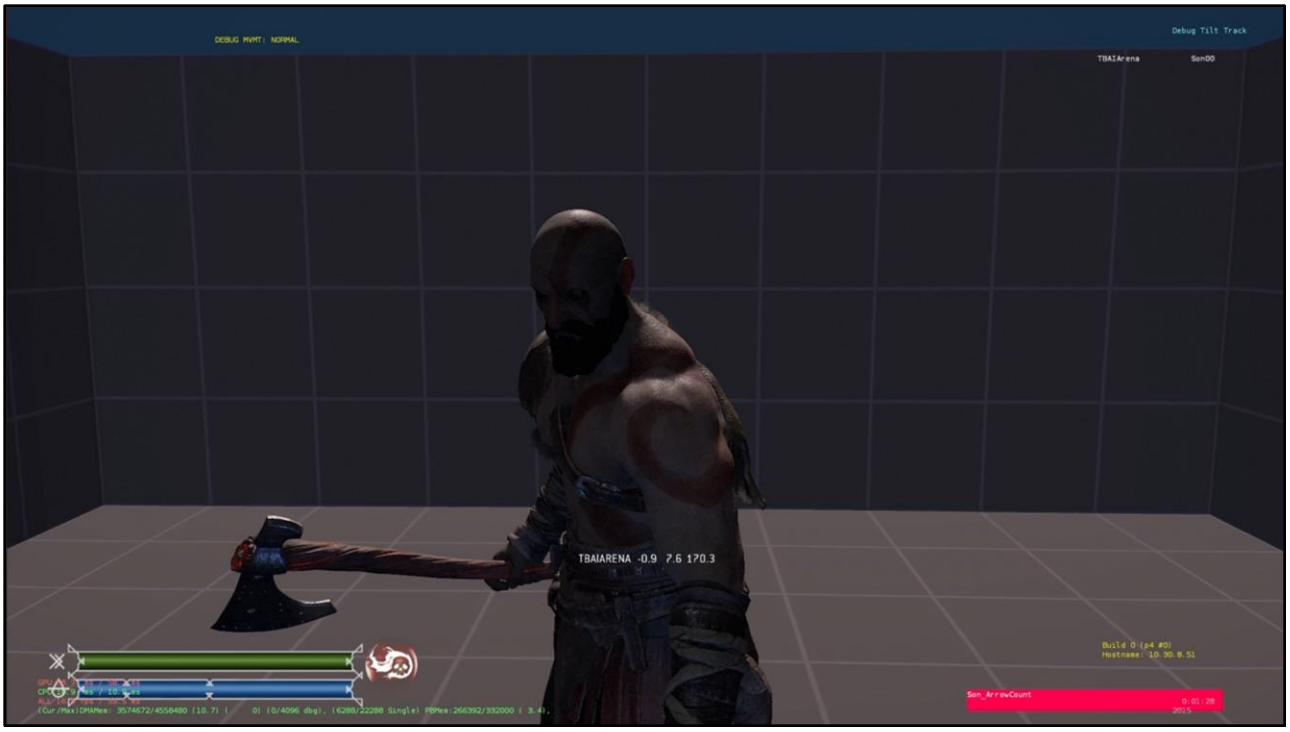
Particles now visibly replace segment

Particles can now animate, move, collide



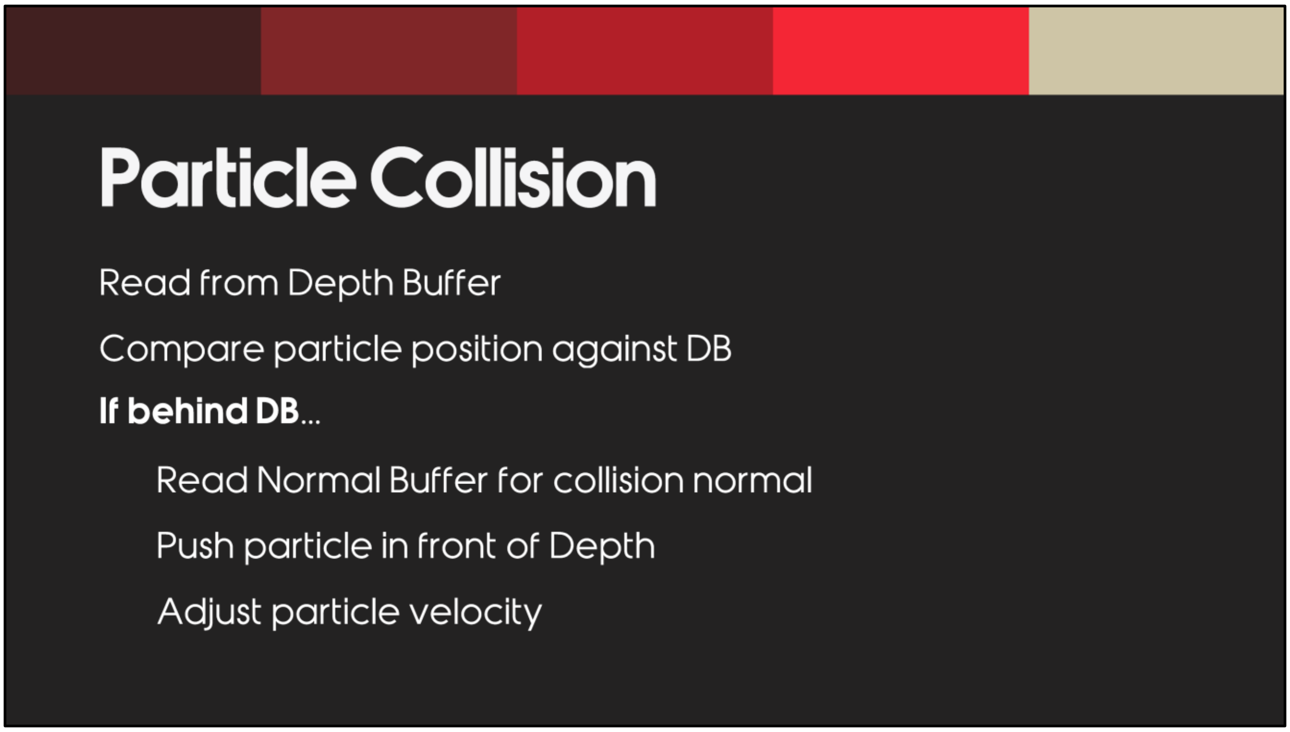
[09:29] [40 seconds]

We have now advanced forward a frame.
Now is the time to start drawing the particles we spawned from the previous frame.
The segment that was visible in the previous frame, but not in this frame, has visually been replaced with the particles we emitted but didn't draw in the previous frame.
We also draw all the other particles that have spawned previously.
We're able to draw all the particles together and treat them uniformly.
We can also animate the particle, and move the particle, and collide the particle with other primitives or the depth buffer.



[10:09] [45-52 seconds]

I wanted to test that we could convincingly recreate a mesh entirely with particles.
What you are seeing here in this video demonstrates exactly that.
Each of the Kratos clones are fully created from little particles.
Every frame, we emit particles from the mesh of Kratos.
As Kratos flies around the test level, the particles remain where they were spawned.
I had a fixed particle ring buffer for this, so the trails of Kratos end where the particle buffer size is full.
You can also see that segments of the clones are missing pieces, this is because they were obscured on emission.



Particle Collision

Read from Depth Buffer

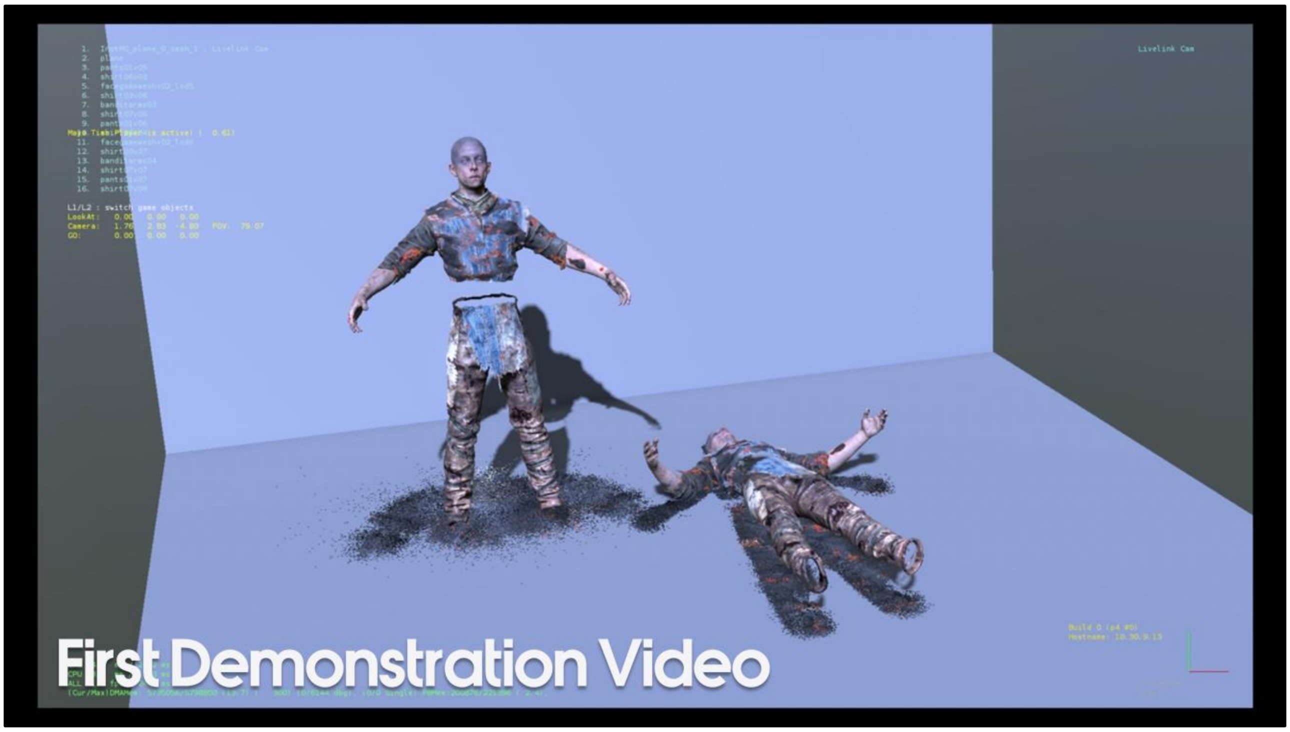
Compare particle position against DB

If behind DB…

- Read Normal Buffer for collision normal
- Push particle in front of Depth
- Adjust particle velocity

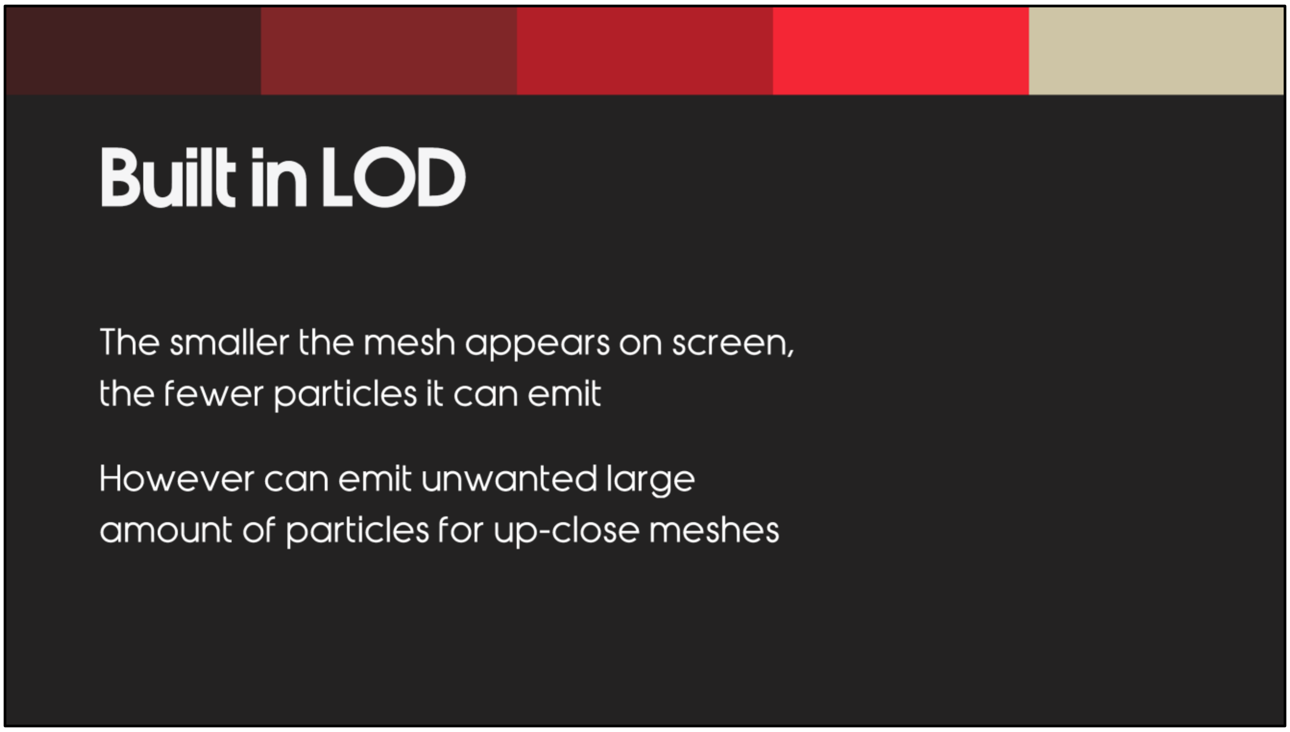
[11:00] [40 seconds]

Particles are easily able to collide with the depth buffer.
Start by taking the particle's world space co-ordinates, and project them to screen space co-ordinates.
You can now sample the depth buffer with these screen space co-ordinates.
Compare the particle's projected z value with the depth buffer value.
If the particle is behind the depth buffer, you can use the same screen space co-ordinates to read the normal buffer and use that as your collision normal.
Then nudge the particle in front of the depth buffer, and update the particle velocity.



[11:40] [60 seconds]

Now here we have the first video of putting this technique in to the hands of our lead character artist, Raf Grassetti.
Raf is playing the decay animation on a loop.
Raf is also moving the model around manually while the loop plays, to inspect the effect from multiple angles.
The animation changes the alpha reference value over time, causing the particles to emit and the mesh to disappear.
You can also see Raf added some material animation to help sell the effect.
Raf has made the mesh burn to ash before emitting the particle, causing a pile of ashes to fall to the ground.
You can also see that the mesh has burned in to a pile of ash in the shape of the silhouette of the mesh, very neat!
Just as a note, the character has its feet and waist missing, this model was a work in progress at the time of testing.



Built in LOD

The smaller the mesh appears on screen, the fewer particles it can emit

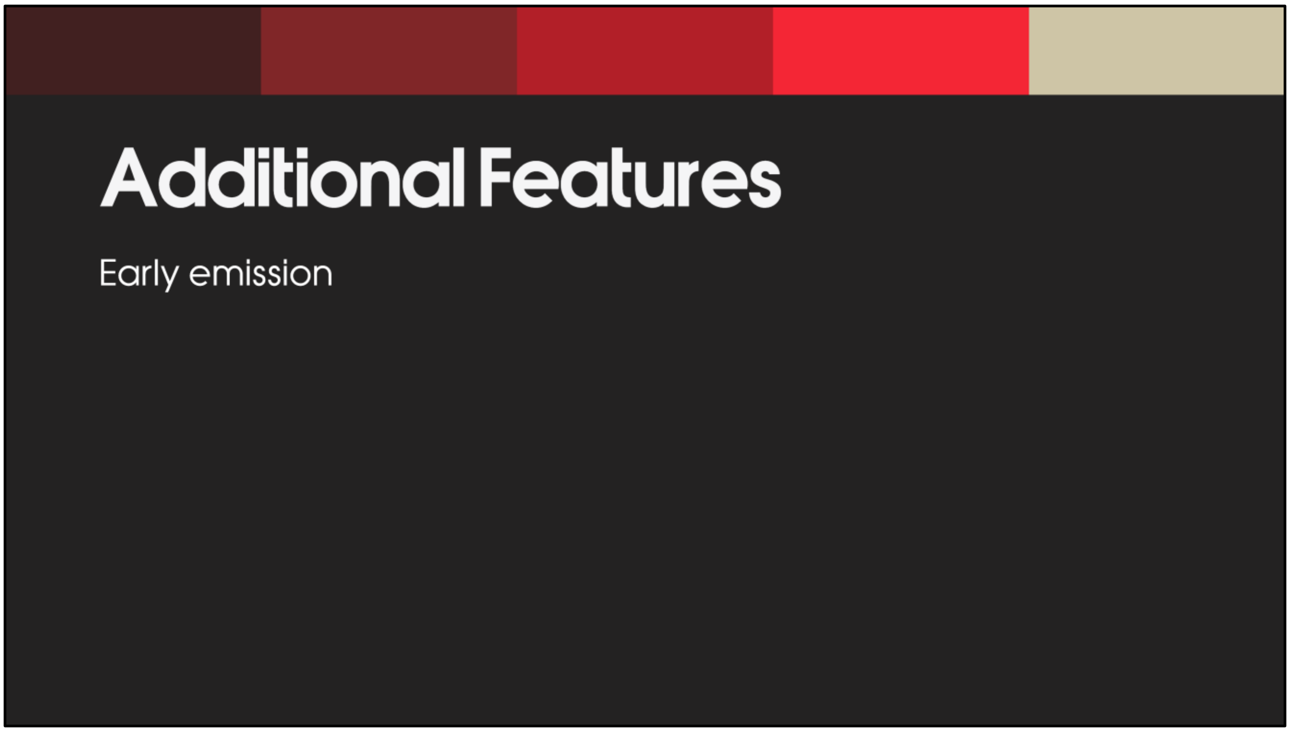
However can emit unwanted large amount of particles for up-close meshes

[12:40] [65 seconds]

One of the key benefits to this technique is it has built in level of detail.
The smaller the mesh appears on screen, the fewer particles it is able to emit.
However this relationship is obviously not linear.
If a mesh is unfortunate enough to be close to the viewing camera, it may occupy large portions of the screen.
This can create the opportunity to emit a large quantity of particles per frame.
However you can counter act this with the authoring parameters.
Such as ensuring you do not animate your reference value too quickly.
With a smaller reference value speed, you create smaller segments of particle emission.
Our particle system programmer, Paolo, also introduced a method of emission randomization to help reduce particle emission count in certain scenarios.

Additional Features

[13:45] [5 seconds]

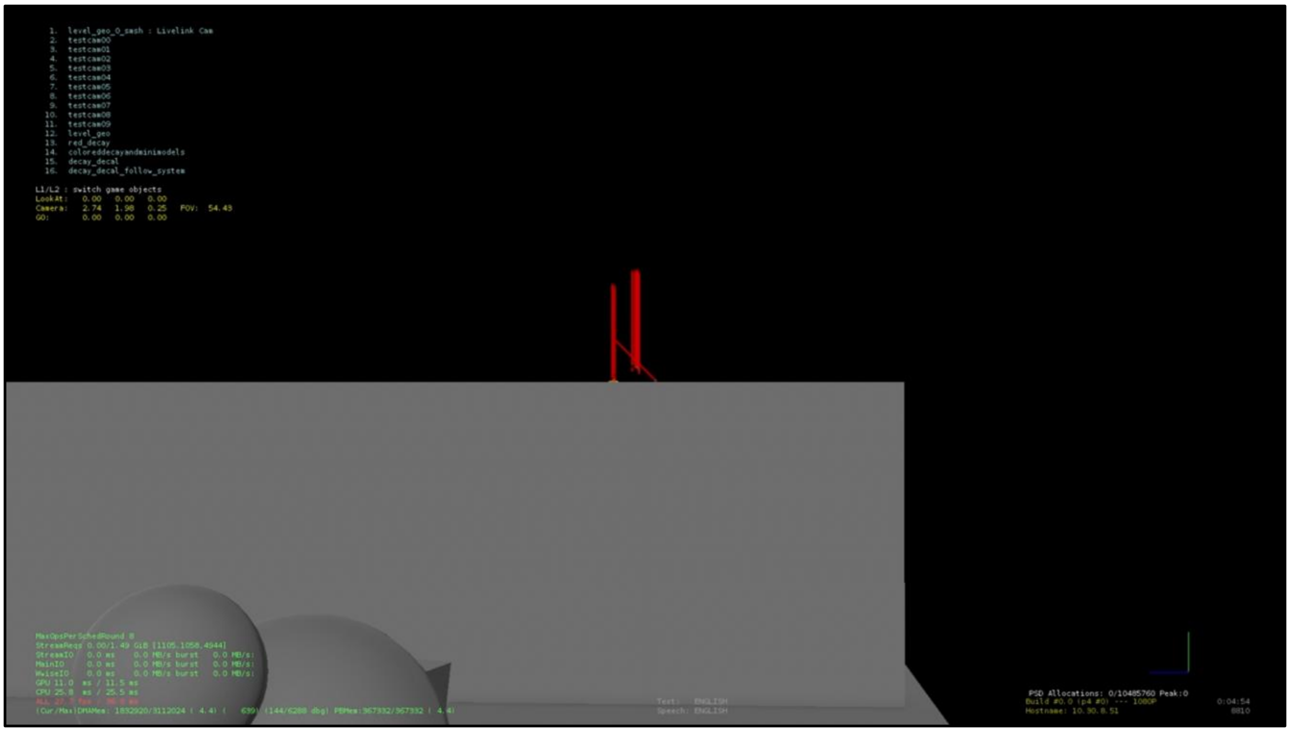


Additional Features

Early emission

[13:50] [65 seconds]

By decoupling the reference value between emission and alpha reference testing, you can actually start emitting particles early, before the mesh will disappear.
That is to say, instead of emitting a single particle the frame immediately before the pixel of the mesh disappears, you emit one particle per frame over multiple frames right before the pixel mesh disappears.
This gives you the ability to produce a more substantial decay effect, for example denser objects, but at a cost of more particles.
You can achieve this with a single ALU add instruction in the emission shader, you add a uniform value to the result of the texture sample before you do the alpha reference comparison.
This detaches the emission of the particle from the disappearing mesh by a variable amount supplied.



```
 1.  level_geo_0_smsh : Livelink Cam
 2.  testcam00
 3.  testcam01
 4.  testcam02
 5.  testcam03
 6.  testcam04
 7.  testcam05
 8.  testcam06
 9.  testcam07
10.  testcam08
11.  testcam09
12.  level_geo
13.  red_decay
14.  coloreddecayandminimodels
15.  decay_decal
16.  decay_decal_follow_system

L1/L2 : switch game objects
LookAt:   0.00   0.00   0.00
Camera:   2.74   1.98   0.25   FOV:  54.43
GO:       0.00   0.00   0.00
```

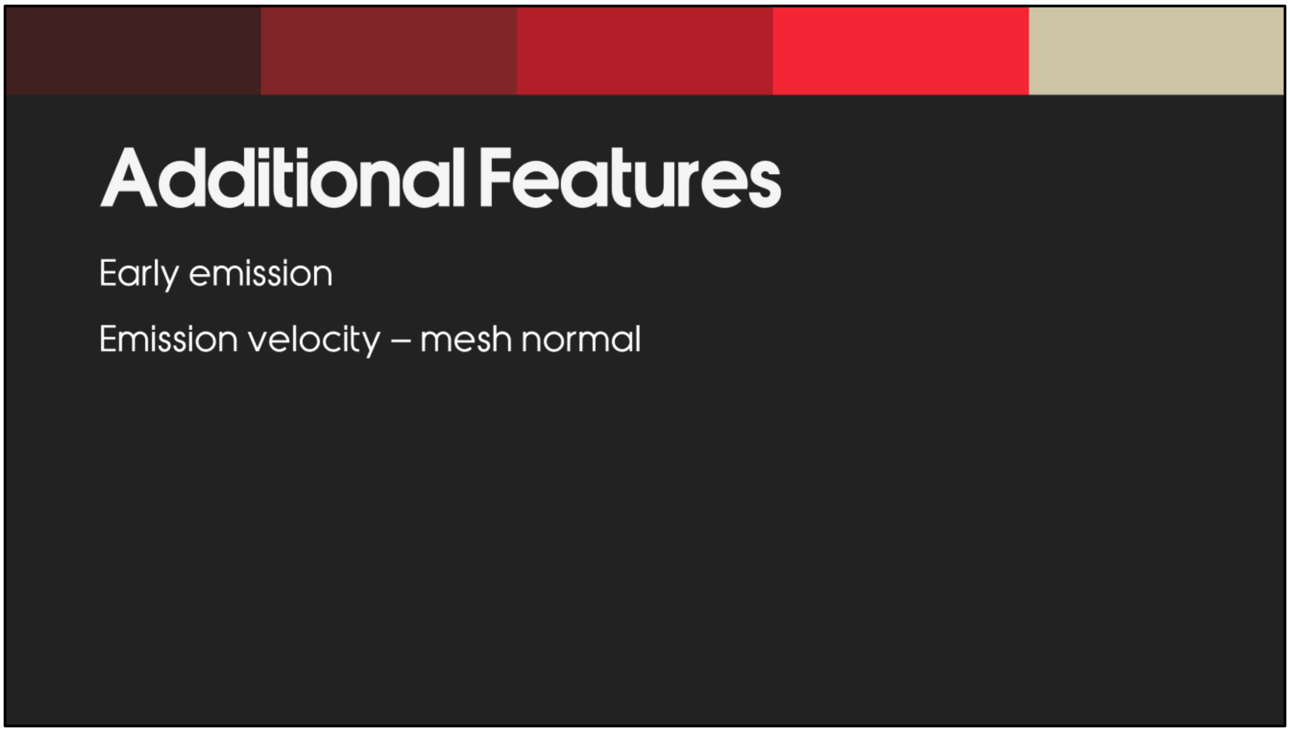
```
MaxOpsPerSchedRound 8
StreamReqs 0.00/1.49 GiB [1105.1058.4944]
StreamIO   0.0 ms    0.0 MB/s burst   0.0 MB/s:
MainIO     0.0 ms    0.0 MB/s burst   0.0 MB/s:
WwiseIO    0.0 ms    0.0 MB/s burst   0.0 MB/s:
GPU 11.0  ms / 11.5 ms
CPU 25.8  ms / 25.5 ms
[illegible]
(Cur/Max)DMAMem: 1832920/3112024 ( 4.4) (   639) (144/6288 dbg) PBMem:367332/367332 ( 4.4)
```

```
Text:   ENGLISH
Speech: ENGLISH
```

```
PSD Allocations: 0/10485760 Peak:0
Build #0.0 (p4 #0) --- 1080P                 0:04:54
Hostname: 10.30.8.51                            8810
```

[14:55] [21 seconds]

Here you can see a video of the particles emitting at pixels right as the mesh is disappearing.
When the little explosion occurs, it will change over to emitting particles earlier than when the mesh pixel disappears.
It gives a nice effect of giving the emitting area a visually larger size.



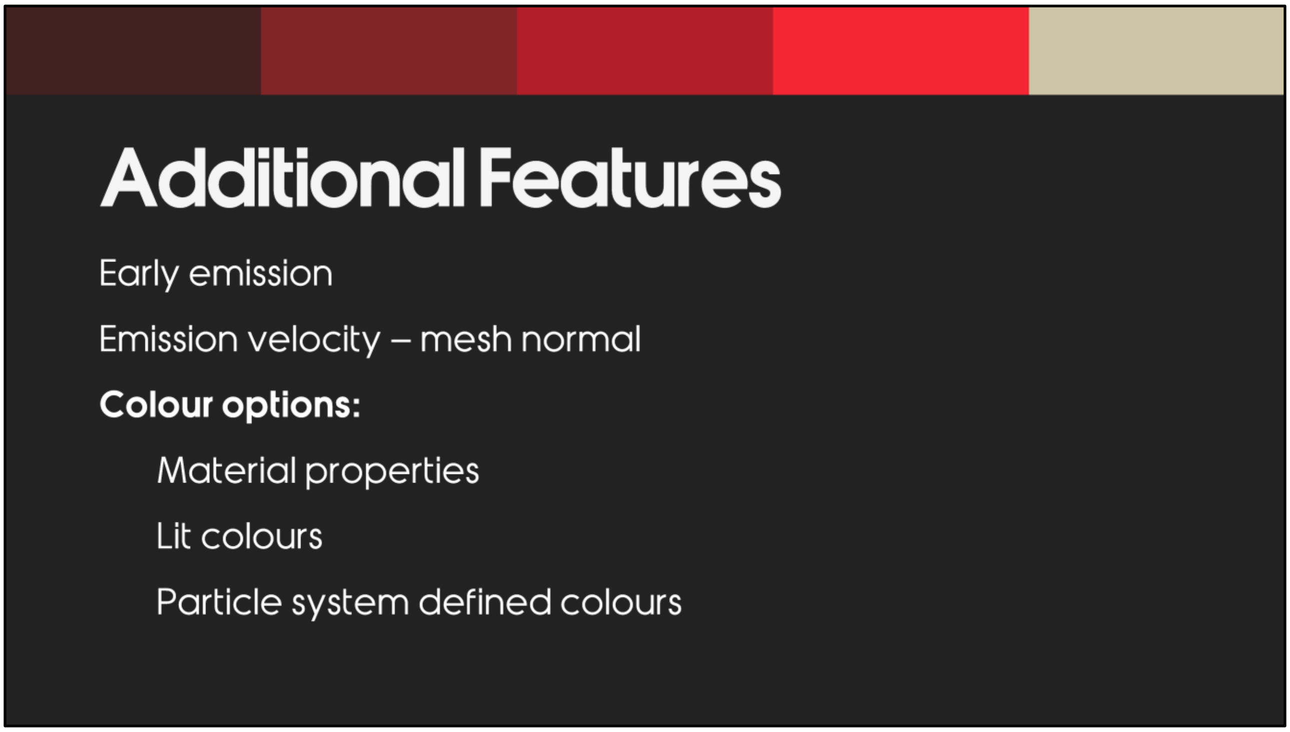
Additional Features

Early emission

Emission velocity – mesh normal

[15:16] [20 seconds]

Emission velocity is a useful feature to have available to designers.
Particles may just want to fall from the mesh, or perhaps they may want to sample the normal buffer from their emission screen coordinates, and explode off the mesh instead.



Additional Features

Early emission

Emission velocity – mesh normal

Colour options:

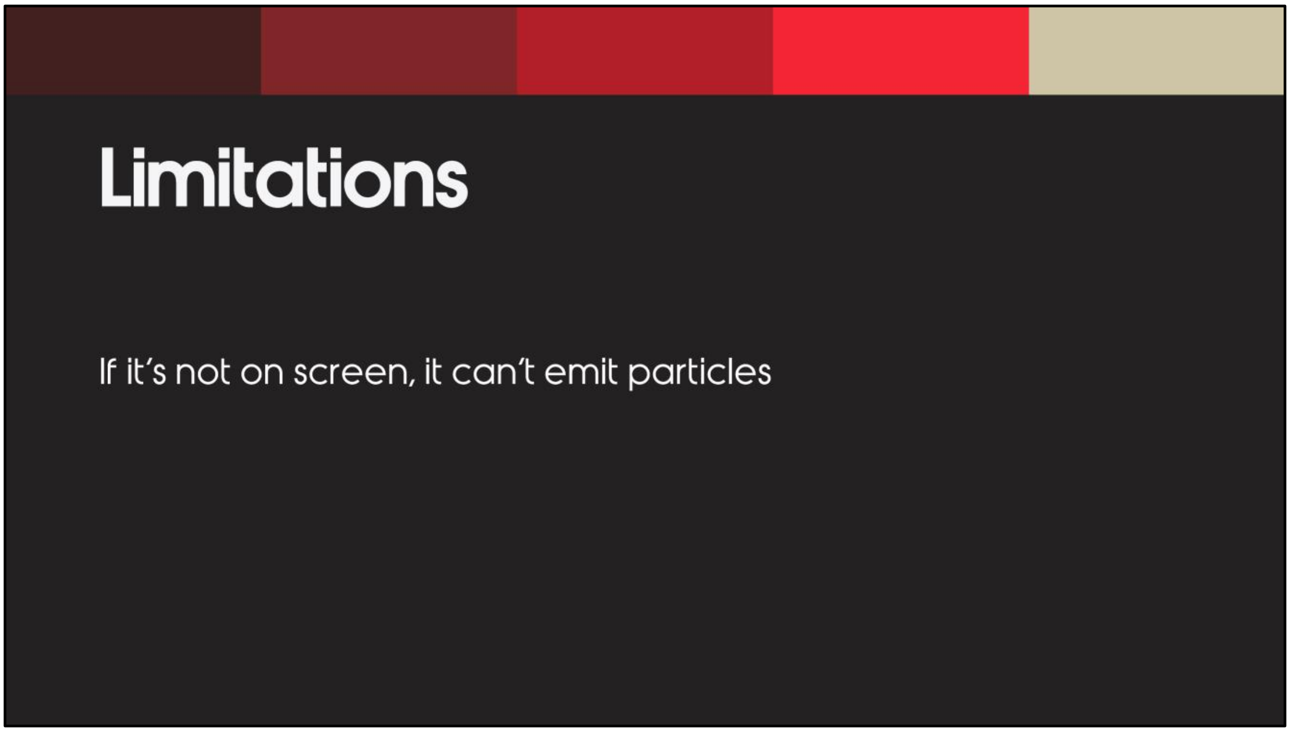
- Material properties
- Lit colours
- Particle system defined colours

[15:36] [40 seconds]

Particles have a range of options to choose from when picking their colour.
The emitted colour could be the final lit colour of the pixel.
Or it could derive a colour from the material in the rendered g-buffer.
Or the colour could be a part of the particle system attributes.
There's no reason it can't be a combination of these, such as start off as the same colour as the final lit pixel, then blend towards the external colours supplied with the particle system over time.

Limitations

[16:16] [5 seconds]

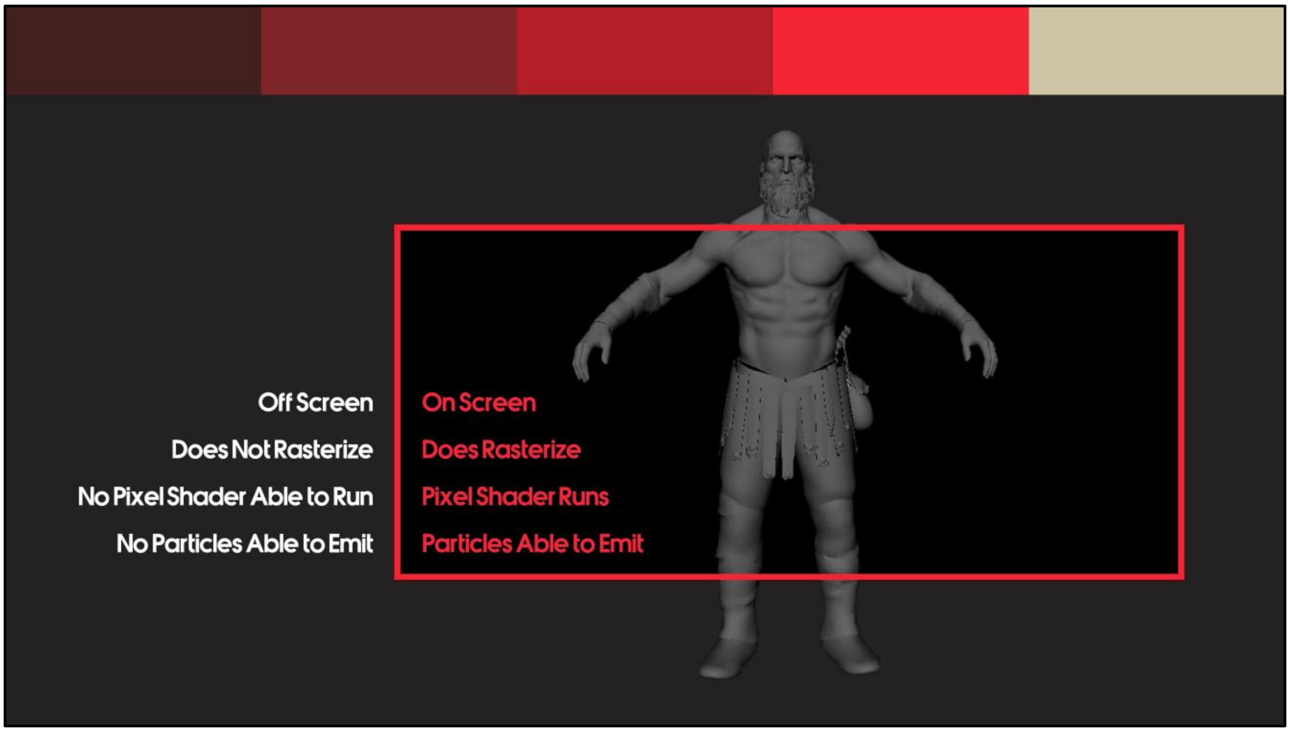


Limitations

If it's not on screen, it can't emit particles

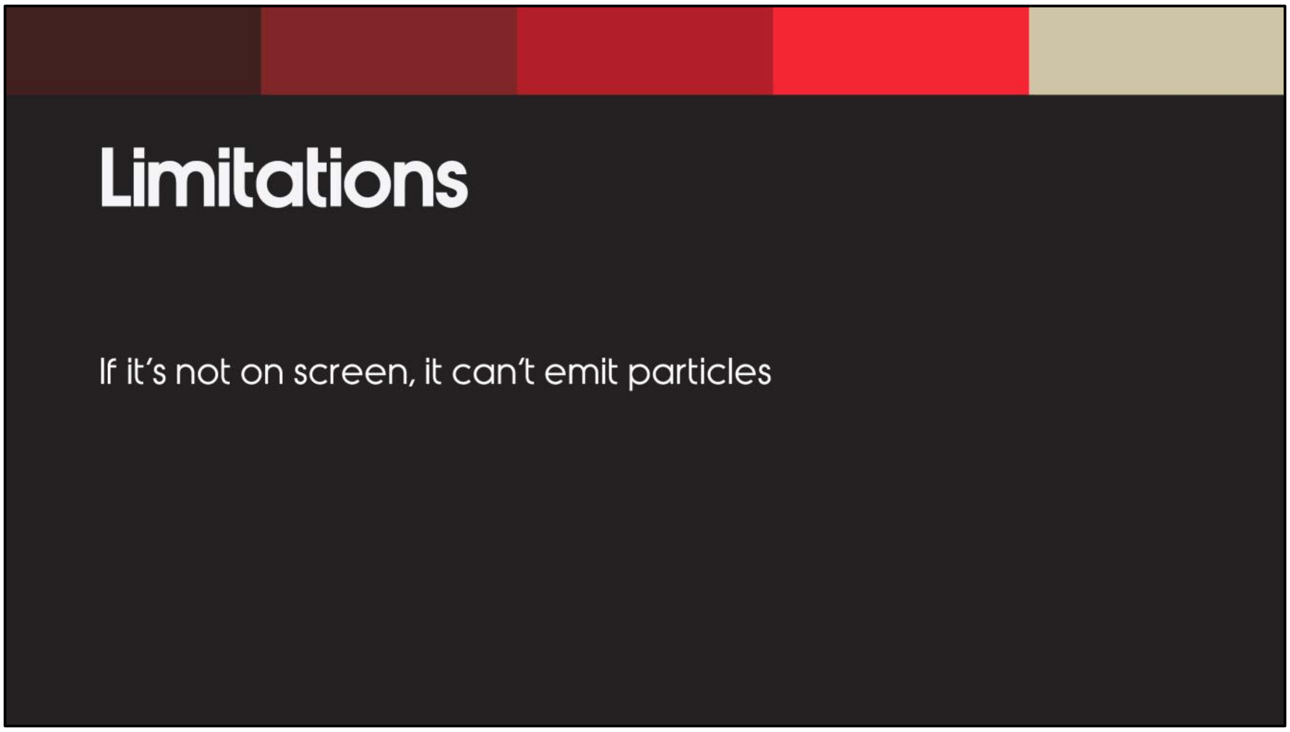
[16:21] [10 seconds]

This technique also has some drawbacks.
If the mesh is not on screen, it can't emit particles.



[16:31] [38 seconds]
So Kratos' head and feet are off screen.
The rest of the body is on screen.
Pixel shaders are only ever run on pixels that get rasterized.
It can't be rasterized if it's outside the viewport.
So this means Kratos' head and feet will not be able to emit particles.

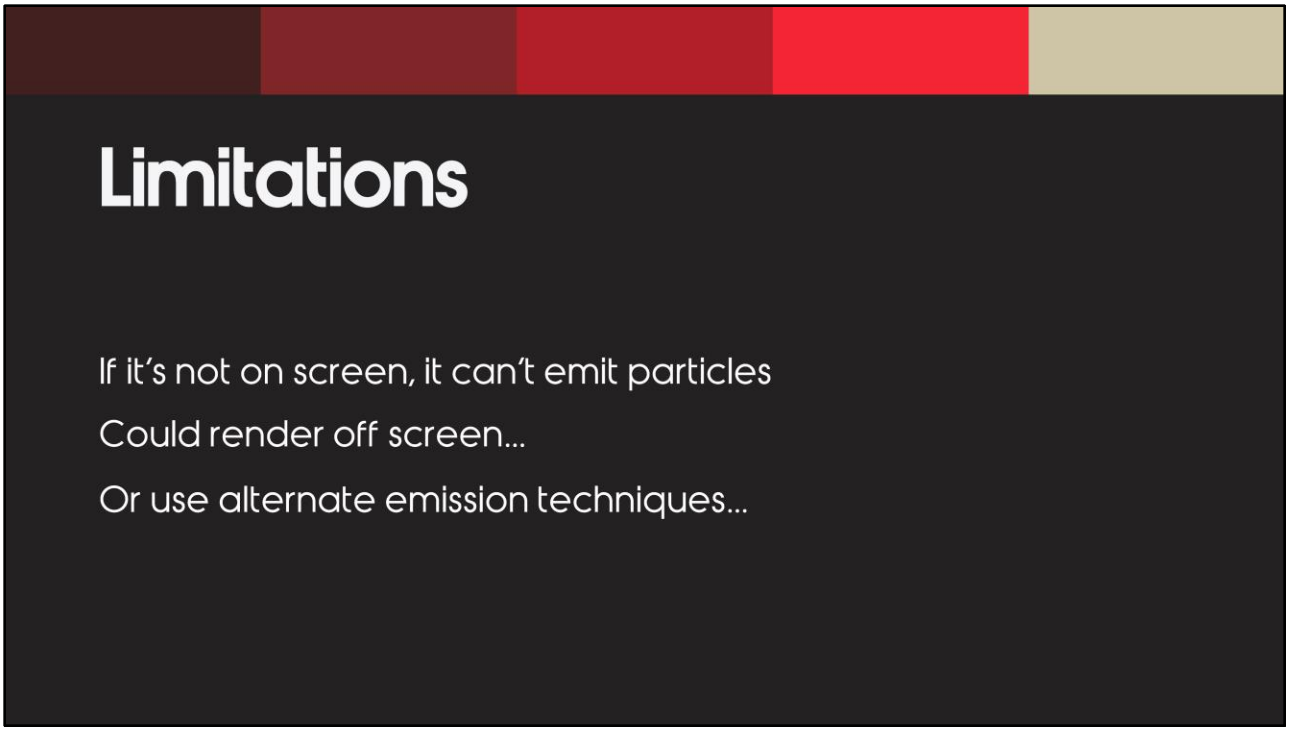
If we were to have an alpha reference emit particles from Kratos' head to feet, you would expect Kratos' particles to fall from his head and in to the viewport.
But they won't, the head will gradually disappear downwards even though you won't be able to tell.
Then the top of his shoulders and chest will start disappearing and emitting particles.


Limitations

If it's not on screen, it can't emit particles

[17:09] [45 seconds]

Some of our designers unfortunately fell in to this trap a couple of times.
One in particular was very impressed with the technique, he wanted to use it for revealing a hidden passageway from a fake wall.
He set the fake wall material up with the decay option, and was pleased with the visual result, it was decaying like sand from top to bottom right in front of your eyes.
However he noticed that when triggering the effect from up close and looking around, the wall kind of just cut off near the top of the screen as the particles were falling down.
This was caused by the mesh not being on screen in order to emit particles.



Limitations

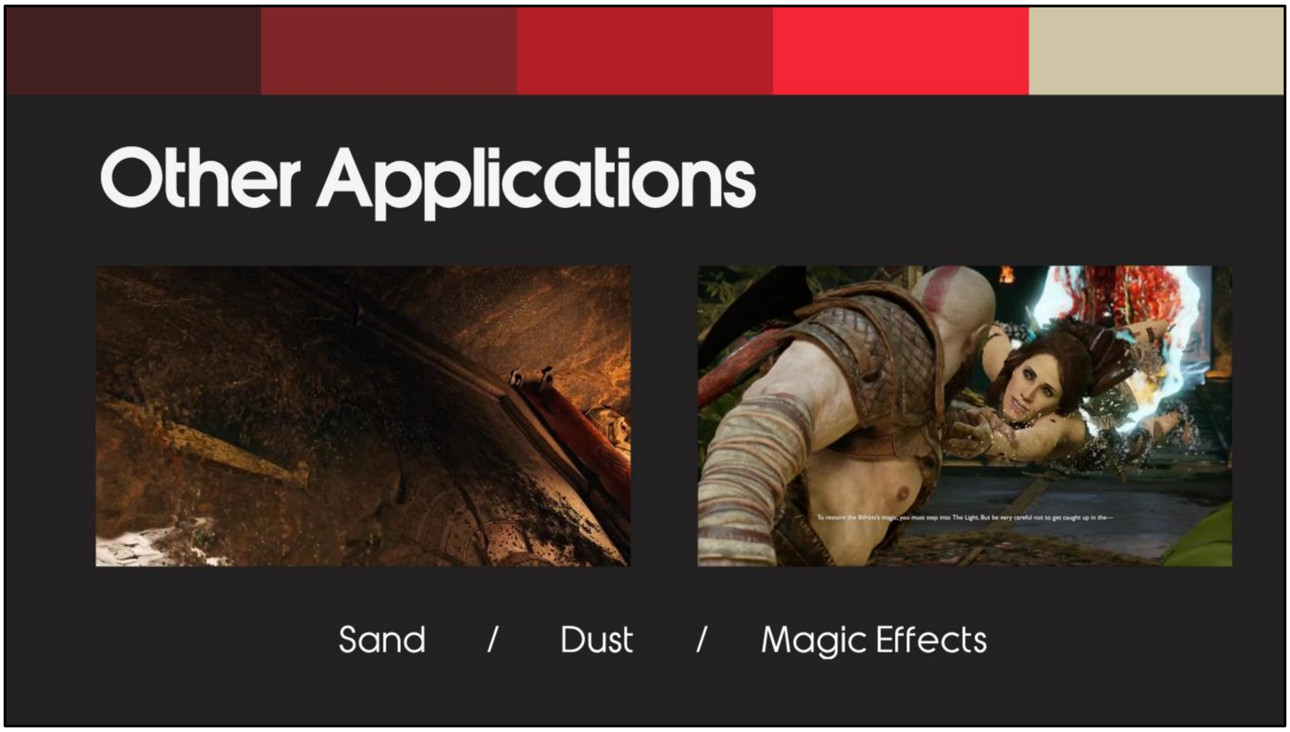
If it's not on screen, it can't emit particles

Could render off screen...

Or use alternate emission techniques...

[17:54] [20 seconds]

There are methods available to you to help counteract this, such as rendering the mesh off screen at the same resolution, or simply just resort to your simple triangle emission techniques.
But these are outside the scope of this technique.

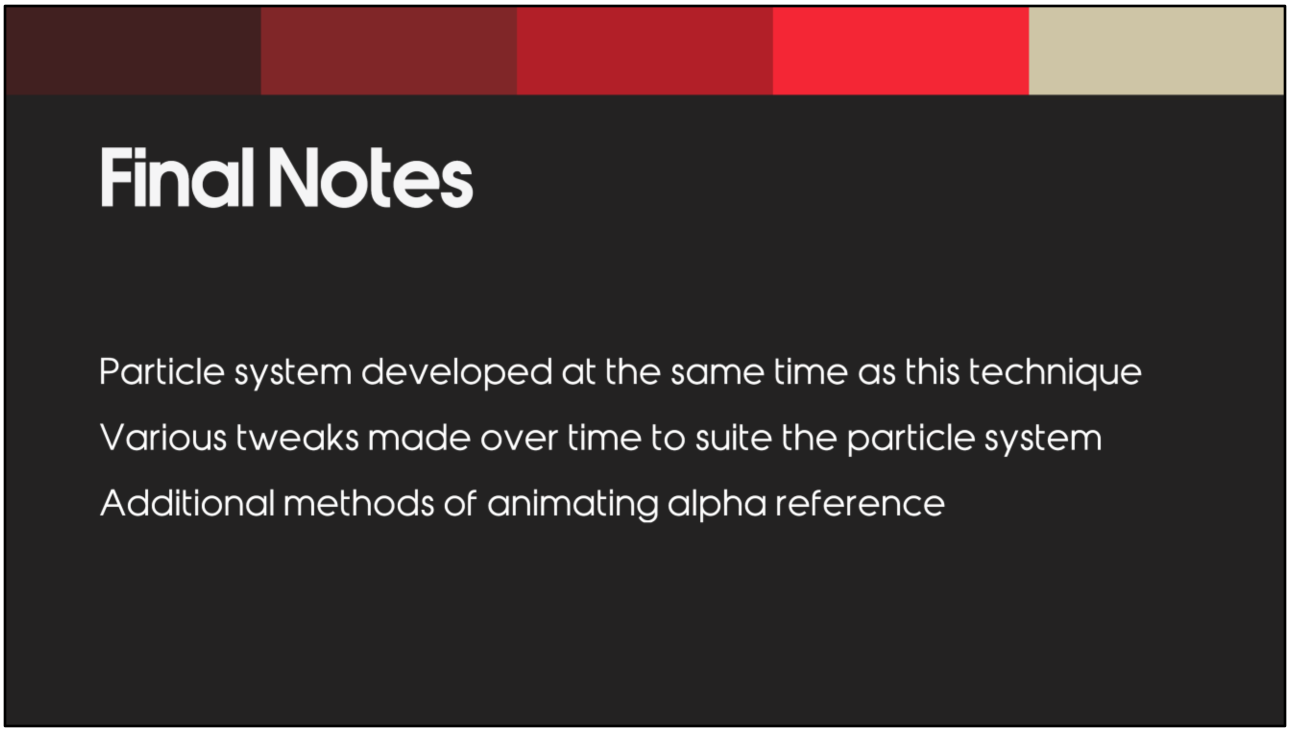


Other Applications

Sand / Dust / Magic Effects

[18:14] [60 seconds]

This technique proved to be very valuable to the effects artists, they've wanted to use it in several other scenarios as well.
As described in the limitations section, it was used for revealing hidden fake walls.
Various magic effects were also applicable to this technique, and was used extensively in one particular cut scene in the game.
The cut scene shows a character being sucked in to a portal.
The character is attempting to resist, and little pieces of the character end up falling away in to the portal.
One setup of this technique was repeatedly asked for; the reverse.
Where particles were scattered around, and are pulled together to form the mesh.
This is very doable, but not with the technique presented here.



Final Notes

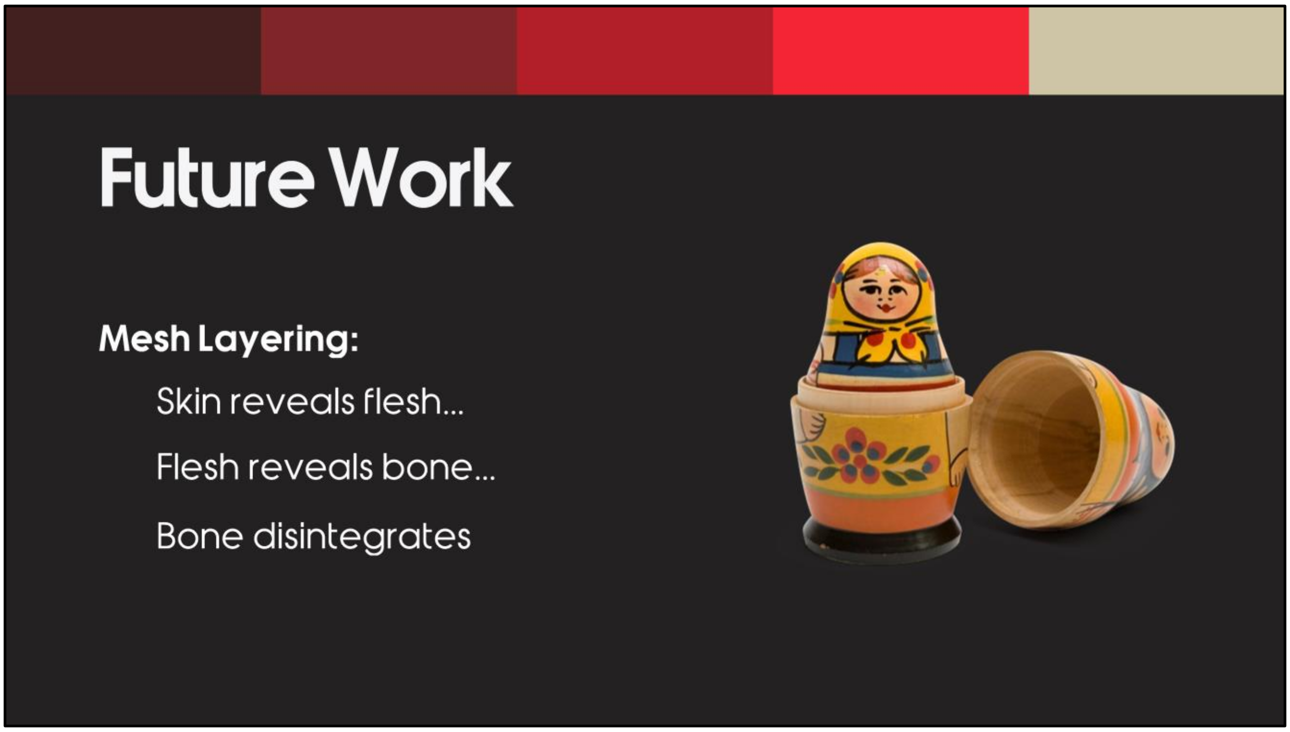
Particle system developed at the same time as this technique

Various tweaks made over time to suite the particle system

Additional methods of animating alpha reference

[19:14] [70 seconds]

So as I said earlier, this was all initially developed in 2015, so this GDC presentation has been 4 years in the making, but we were occupied making the game itself.
Everything I've presented to you was the core, initial version of the technique, and is everything you need to get this technique up and running yourself.
At the same time as this was being developed, our GPU particle system was being developed by Simone Kulczycki.
Over the next few years, some changes were made to the technique, in order to fit in nicely with our new particle system.
Our particle system programmer Paolo (“Sooreekkeo”) Surricchio also added some new mechanisms on how to animate the alpha reference value.
Originally you could key-frame the alpha reference value through our animation sequences.
He expanded on that workflow and added mechanisms to animate it through script, and also hooked it up to our "death system".
He also added a feature to ration out the particle spawning between different multiple disintegrating meshes dependent on their screen size.



Future Work

Mesh Layering:

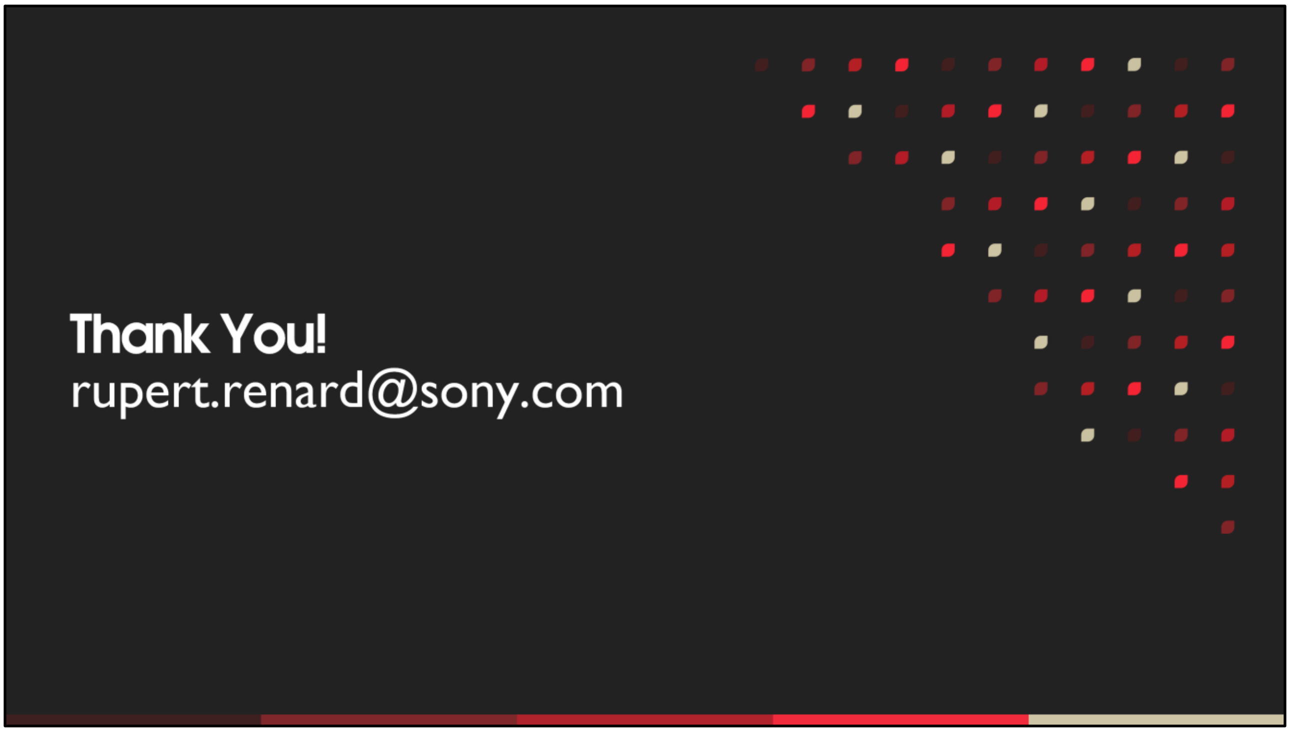
Skin reveals flesh…

Flesh reveals bone…

Bone disintegrates

[20:24] [45 seconds]

One particular feature I was hoping some of our artists would explore was mesh layering.
The character would have multiple layers of meshes, like a Matryoshka doll aka the Russian nesting doll.
Before the disintegration begins, the inside layers would never be rendered for the sake of performance.
But once disintegration starts, the inner layers are rendered.
As the outer mesh layer of skin disintegrates, it would reveal the flesh mesh layer beneath it.
When the flesh mesh layer would disintegrate, it would reveal bone beneath it.
Then the bone mesh would disintegrate.



Thank You!

rupert.renard@sony.com

[21:09] [20 seconds]

Thanks everyone for attending, I'd just like to take a few moments to thank others who helped out in various ways.
Paolo ("Sooreekkeo") Surricchio and the rest of the Rendering Team.
Max Ancar, Kevin Huynh and the rest of the FX Team.
Jack Mulholland, and Christina Coffin.

Santa Monica Studio
10|10 IGN
10|10 DUALSHOCKERS
"A TRIUMPHANT REINVENTION" WAYPOINT
"THE BEST GOD OF WAR GAME" VG247
"A MIGHTY SUCCESS" WASHINGTON POST
10|10 DARKSTATION
10|10 GAMEBLOG
95 OPENCRITIC

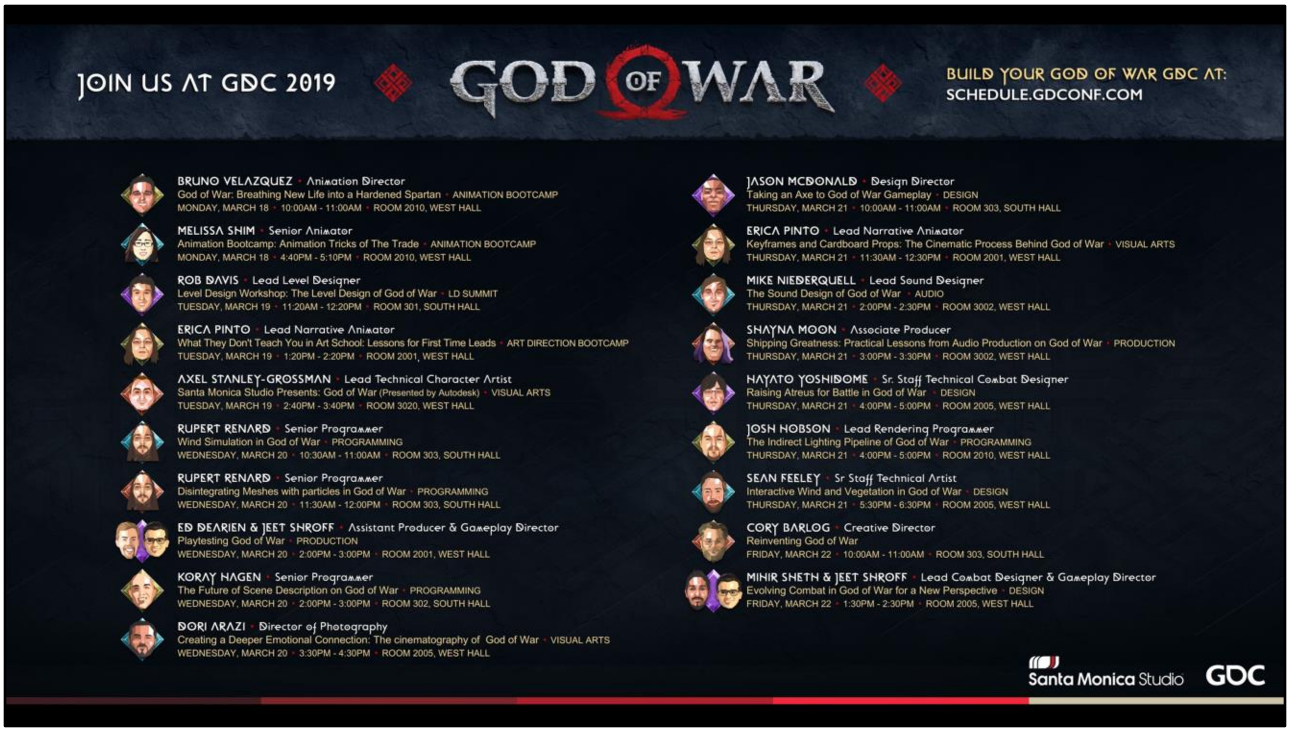
Our journey
Your story

We're hiring for what's next!

We're expanding our family across disciplines and would love to meet you. Please visit sms.playstation.com/careers for all openings or drop us a line at sms@sony.com

@santamonicastudio @SonySantaMonica @santamonicastudio

[21:29]



JOIN US AT GDC 2019

GOD OF WAR

BUILD YOUR GOD OF WAR GDC AT:
SCHEDULE.GDCONF.COM

BRUNO VELAZQUEZ • Animation Director
God of War: Breathing New Life into a Hardened Spartan • ANIMATION BOOTCAMP
MONDAY, MARCH 18 • 10:00AM - 11:00AM • ROOM 2010, WEST HALL

MELISSA SHIM • Senior Animator
Animation Bootcamp: Animation Tricks of The Trade • ANIMATION BOOTCAMP
MONDAY, MARCH 18 • 4:40PM - 5:10PM • ROOM 2010, WEST HALL

ROB DAVIS • Lead Level Designer
Level Design Workshop: The Level Design of God of War • LD SUMMIT
TUESDAY, MARCH 19 • 11:20AM - 12:20PM • ROOM 301, SOUTH HALL

ERICA PINTO • Lead Narrative Animator
What They Don't Teach You in Art School: Lessons for First Time Leads • ART DIRECTION BOOTCAMP
TUESDAY, MARCH 19 • 1:20PM - 2:20PM • ROOM 2001, WEST HALL

AXEL STANLEY-GROSSMAN • Lead Technical Character Artist
Santa Monica Studio Presents: God of War (Presented by Autodesk) • VISUAL ARTS
TUESDAY, MARCH 19 • 2:40PM - 3:40PM • ROOM 3020, WEST HALL

RUPERT RENARD • Senior Programmer
Wind Simulation in God of War • PROGRAMMING
WEDNESDAY, MARCH 20 • 10:30AM - 11:00AM • ROOM 303, SOUTH HALL

RUPERT RENARD • Senior Programmer
Disintegrating Meshes with particles in God of War • PROGRAMMING
WEDNESDAY, MARCH 20 • 11:30AM - 12:00PM • ROOM 303, SOUTH HALL

ED DEARIEN & JEET SHROFF • Assistant Producer & Gameplay Director
Playtesting God of War • PRODUCTION
WEDNESDAY, MARCH 20 • 2:00PM - 3:00PM • ROOM 2001, WEST HALL

KORAY HAGEN • Senior Programmer
The Future of Scene Description on God of War • PROGRAMMING
WEDNESDAY, MARCH 20 • 2:00PM - 3:00PM • ROOM 302, SOUTH HALL

DORI ARAZI • Director of Photography
Creating a Deeper Emotional Connection: The cinematography of God of War • VISUAL ARTS
WEDNESDAY, MARCH 20 • 3:30PM - 4:30PM • ROOM 2005, WEST HALL

JASON MCDONALD • Design Director
Taking an Axe to God of War Gameplay • DESIGN
THURSDAY, MARCH 21 • 10:00AM - 11:00AM • ROOM 303, SOUTH HALL

ERICA PINTO • Lead Narrative Animator
Keyframes and Cardboard Props: The Cinematic Process Behind God of War • VISUAL ARTS
THURSDAY, MARCH 21 • 11:30AM - 12:30PM • ROOM 2001, WEST HALL

MIKE NIEDERQUELL • Lead Sound Designer
The Sound Design of God of War • AUDIO
THURSDAY, MARCH 21 • 2:00PM - 2:30PM • ROOM 3002, WEST HALL

SHAYNA MOON • Associate Producer
Shipping Greatness: Practical Lessons from Audio Production on God of War • PRODUCTION
THURSDAY, MARCH 21 • 3:00PM - 3:30PM • ROOM 3002, WEST HALL

HAYATO YOSHIDOME • Sr. Staff Technical Combat Designer
Raising Atreus for Battle in God of War • DESIGN
THURSDAY, MARCH 21 • 4:00PM - 5:00PM • ROOM 2005, WEST HALL

JOSH HOBSON • Lead Rendering Programmer
The Indirect Lighting Pipeline of God of War • PROGRAMMING
THURSDAY, MARCH 21 • 4:00PM - 5:00PM • ROOM 2010, WEST HALL

SEAN FEELEY • Sr Staff Technical Artist
Interactive Wind and Vegetation in God of War • DESIGN
THURSDAY, MARCH 21 • 5:30PM - 6:30PM • ROOM 2005, WEST HALL

CORY BARLOG • Creative Director
Reinventing God of War
FRIDAY, MARCH 22 • 10:00AM - 11:00AM • ROOM 303, SOUTH HALL

MIHIR SHETH & JEET SHROFF • Lead Combat Designer & Gameplay Director
Evolving Combat in God of War for a New Perspective • DESIGN
FRIDAY, MARCH 22 • 1:30PM - 2:30PM • ROOM 2005, WEST HALL

Santa Monica Studio GDC