Game Developers Conference® | **March 9-13, 2010** | Moscone Center | San Francisco, CA

THINK SERVICES
A DIVISION OF UNITED BUSINESS MEDIA LLC

GDC 10

Learn. Network. Inspire.

www.GDConf.com

Rock Show VFX: Bringing Brütal Legend to Life

We are:

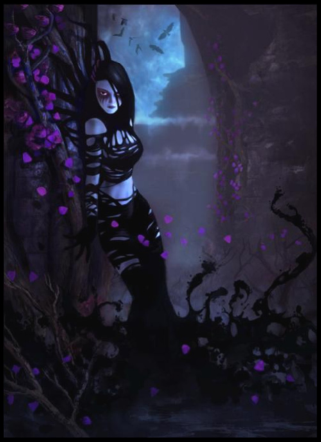
Drew Skillman – Visual Effects / Tech Artist

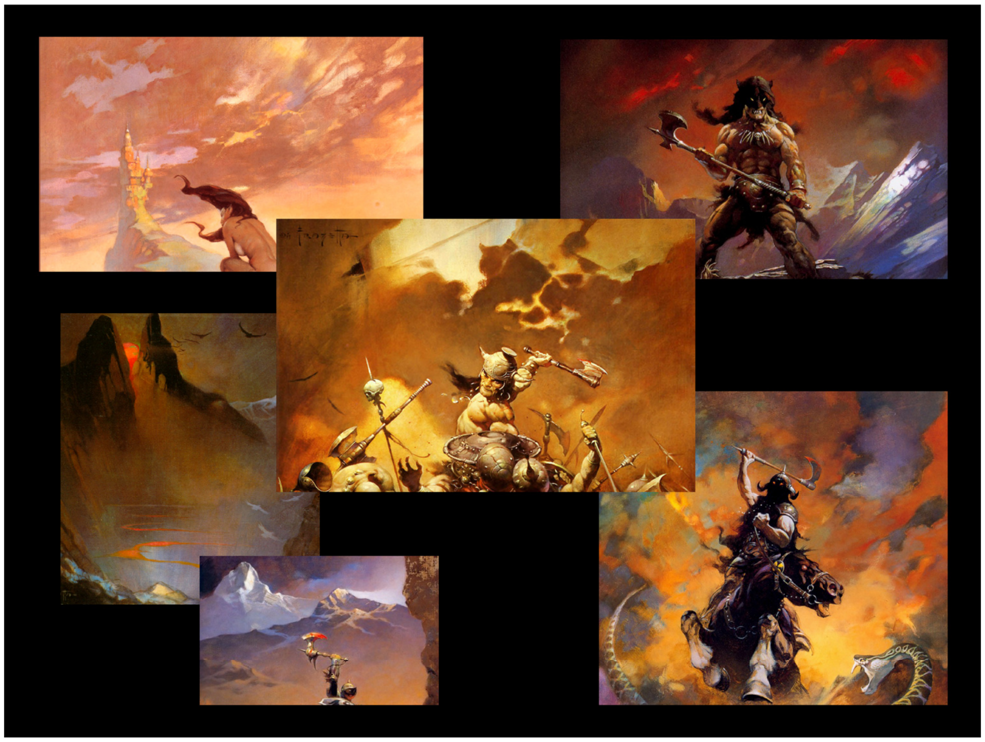
Pete Demoreuille – Programmer

Brütal Legend

Open World, Dynamic Time of Day

Heavy Metal Visual Style





Frazetta’s skies – amazing shapes and colors that really help make the image. We want some of this..

Immortal
At The Heart Of Winter
DIO
HOLY DIVER
IRON MAIDEN
Kill
Rhapsody
Legendary Tales

Heavy metal album cover art. Even more brutal, exaggerated shapes and colors in the skies set the scene. Of course we want some of that.

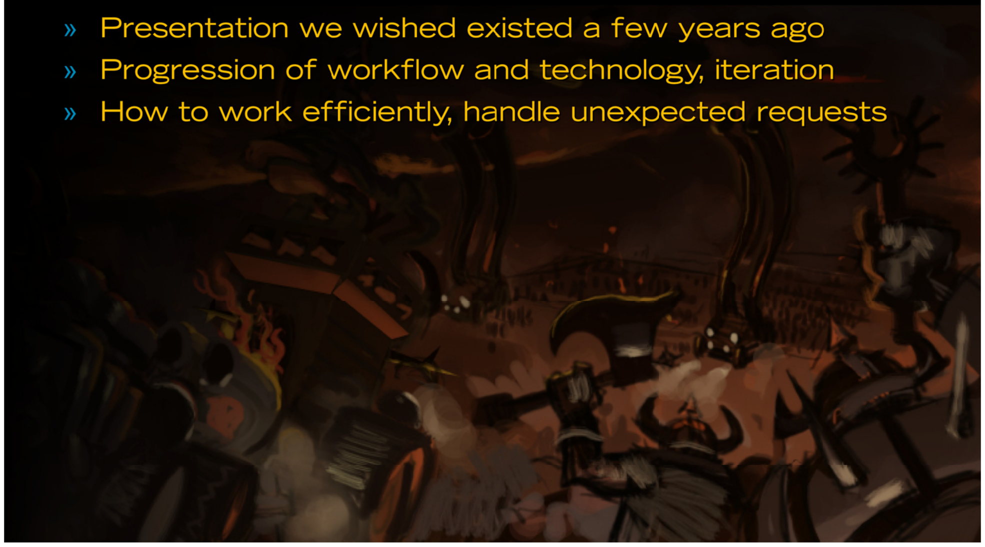
CLAW ATTACK!
THE BRIDE

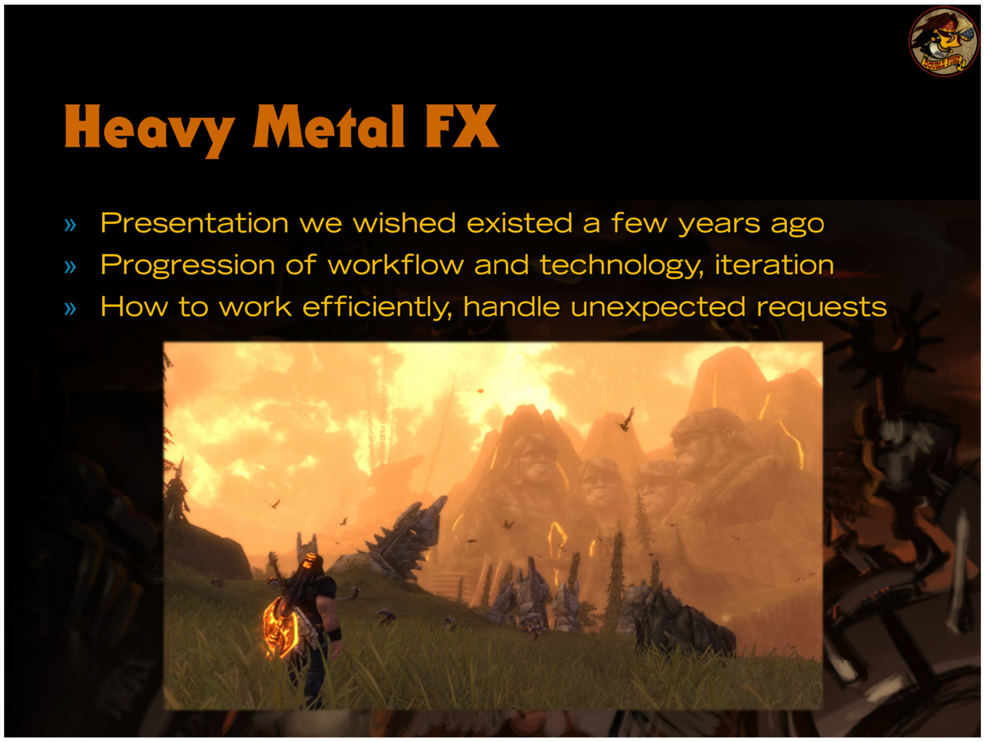
Endless stream of absurdly awesome concepts… all of them featuring some sort of custom fx, skies, or whatnot. We're starting to get a little nervous..

And working at Double Fine, we knew we'd get a lot of unexpected requests that we were not quite prepared to deliver on.. so we knew we had to set ourselves up to be flexible and able to experiment.

Heavy Metal FX

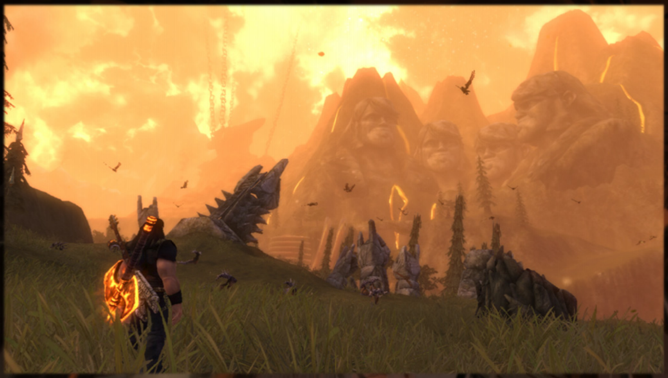
- Presentation we wished existed a few years ago
- Progression of workflow and technology, iteration
- How to work efficiently, handle unexpected requests



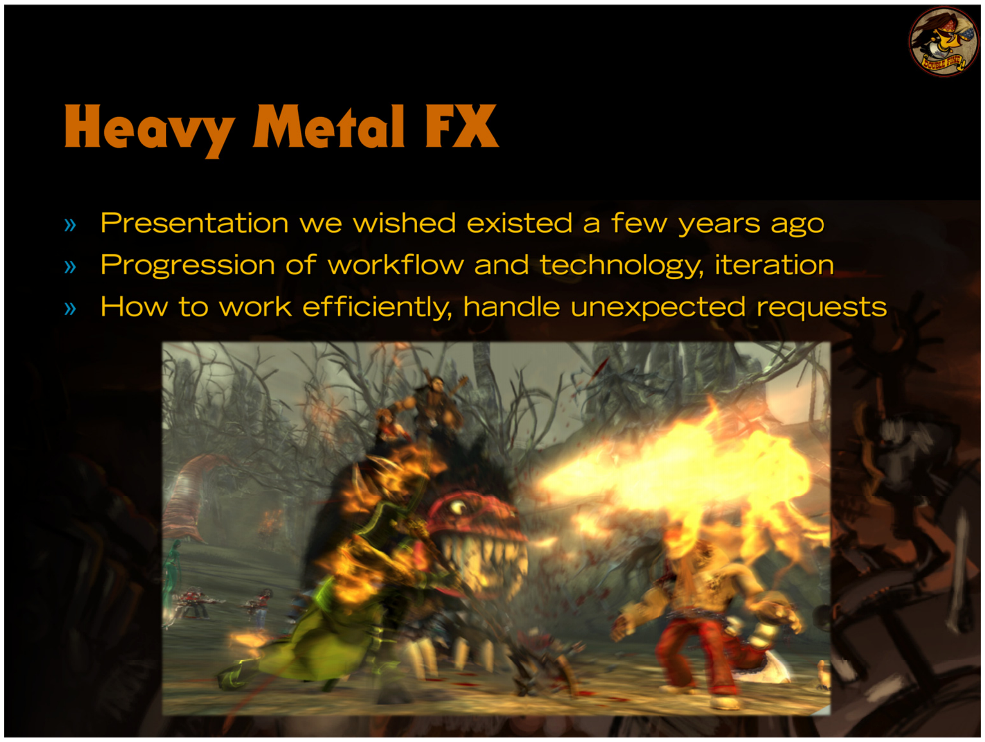


Heavy Metal FX

- Presentation we wished existed a few years ago
- Progression of workflow and technology, iteration
- How to work efficiently, handle unexpected requests



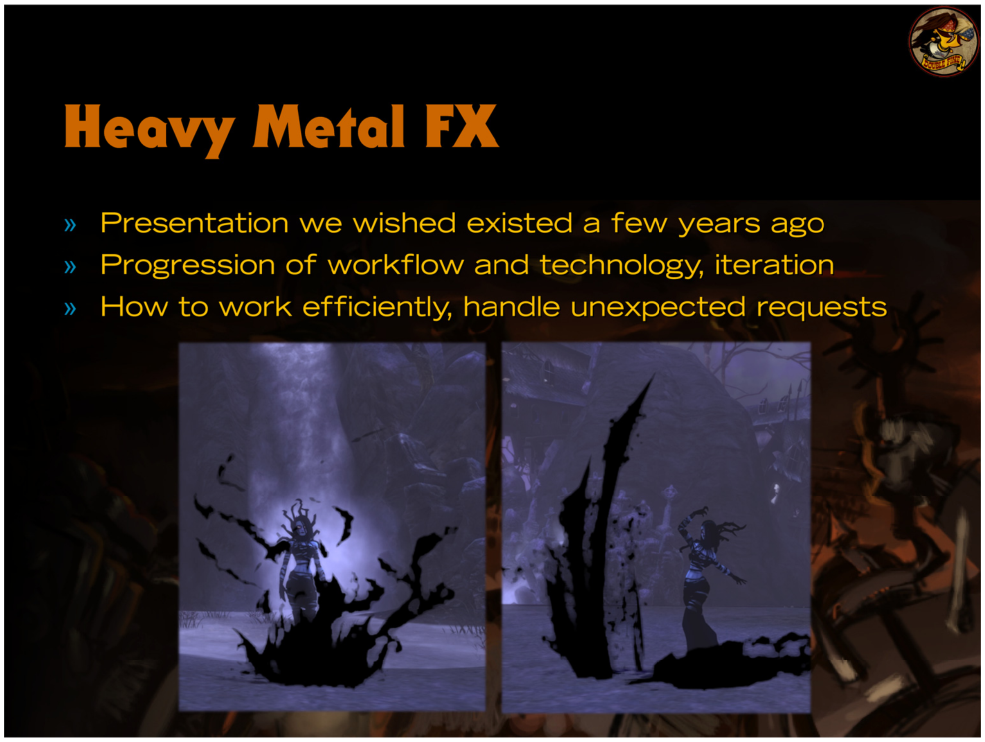
Skies and climates.



Heavy Metal FX

- Presentation we wished existed a few years ago
- Progression of workflow and technology, iteration
- How to work efficiently, handle unexpected requests

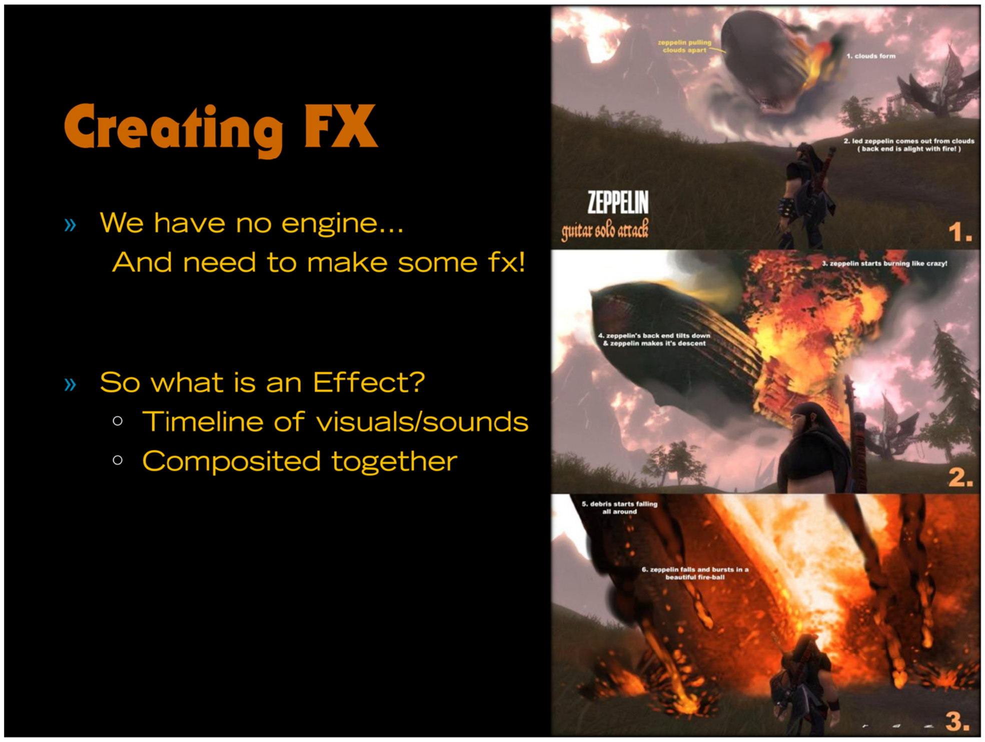
Particle lighting, rendering and simulation.



Heavy Metal FX

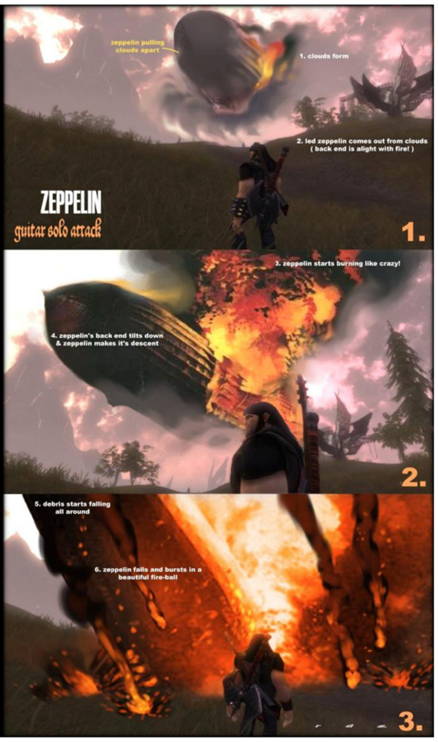
- Presentation we wished existed a few years ago
- Progression of workflow and technology, iteration
- How to work efficiently, handle unexpected requests

Ink.

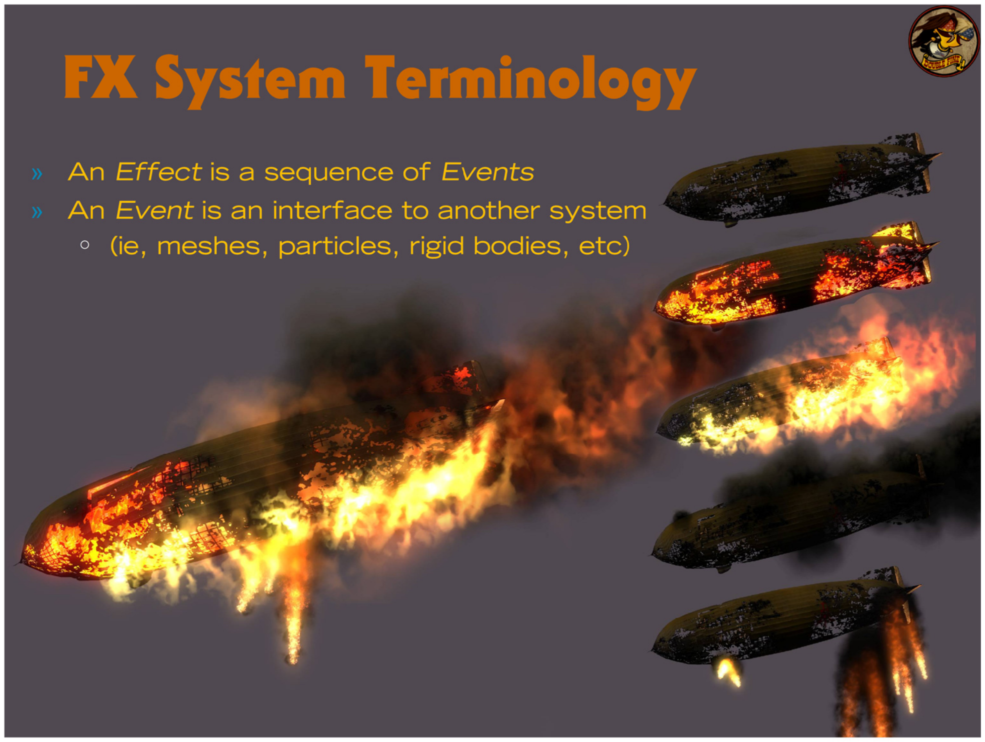


Creating FX

- We have no engine…
  And need to make some fx!
- So what is an Effect?
  - Timeline of visuals/sounds
  - Composited together


zeppelin pulling clouds apart
1. clouds form
2. led zeppelin comes out from clouds (back end is alight with fire!)
ZEPPELIN
guitar solo attack
1.
3. zeppelin starts burning like crazy!
4. zeppelin's back end tilts down & zeppelin makes it's descent
2.
5. debris starts falling all around
6. zeppelin falls and bursts in a beautiful fire-ball
3.

But first, quick overview of FX creation process. Say we need to create zeppelin explosion crazyness. Where do we start?



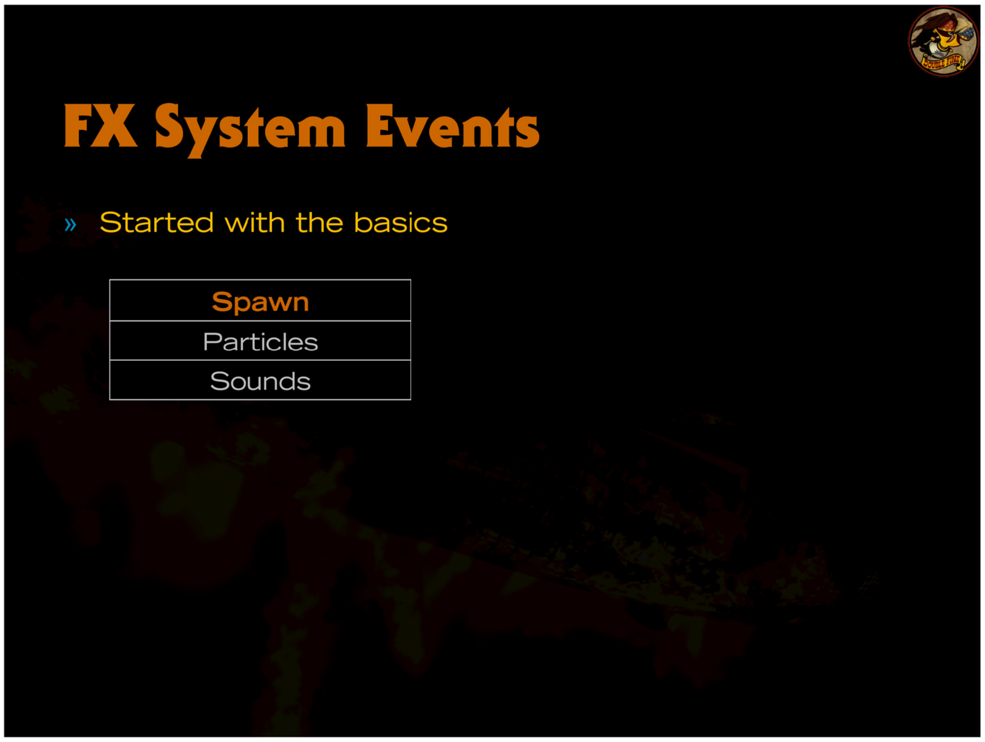
FX System Terminology

- An *Effect* is a sequence of *Events*
- An *Event* is an interface to another system
  - (ie, meshes, particles, rigid bodies, etc)

Effect (FX) built from Events. Events make up the individual elements (visual or otherwise). Each event can be timed, sorted, etc respective to other events in the overall effect.

Events are interesting as they acts as liaison between client and underlying code, and are the interface between engine and vfx artists. This is your chance to make a really good interface that makes sense...

FWIW, often times custom behavior (special attachments, moving effects in a custom way/timing with animations, emitter scaling, setting effect targets) are often implemented client-side. This required much less overall FX system work (and overall work) and was more understandable than a massively complex FX system that was able to handle everything out of the box with no programmer intervention. And given a fairly easy to use API, not at all a burden on the those writing the client code.



FX System Events

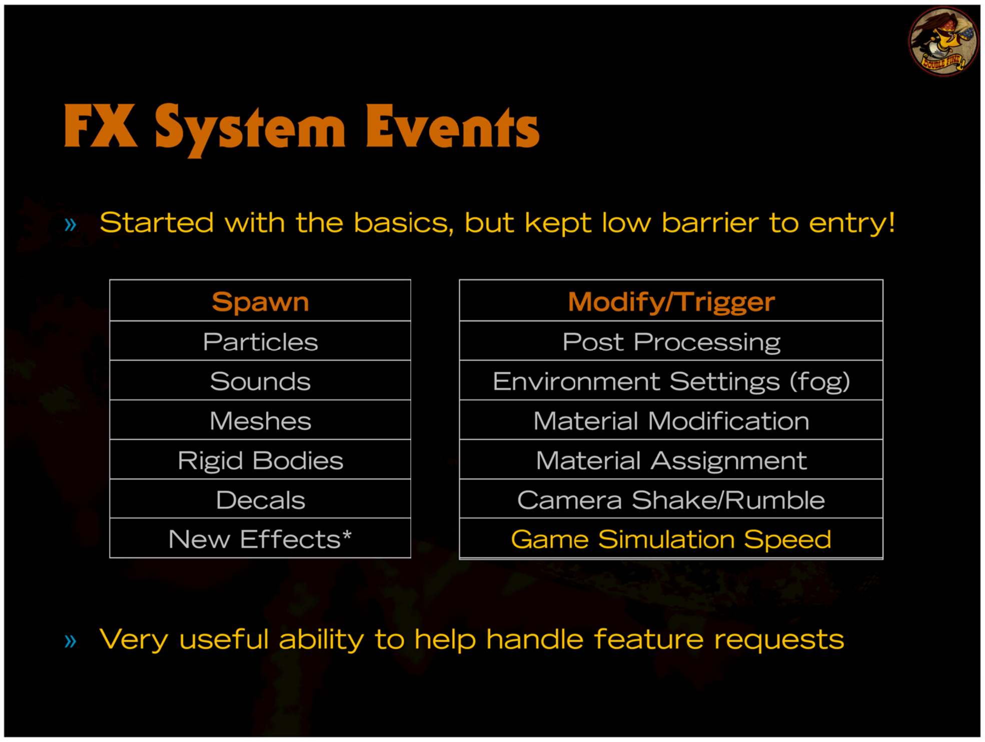
» Started with the basics

Spawn
Particles
Sounds

Key to this system is straightforward way to add new effects, low overhead. If it is easy to add, people will add new ones often, and this leads to more toys, which of course is the overall goal. We started out with just particle events and added the rest over time as the need arose.

Also given the placement of the FX system, it is your opportunity to create a really clean interface for artists and other programmers to use. Simplicity and predictability are paramount.

We often have a large number (many hundreds) of Effects active, and thus it is absolutely essential for the FX system to be as fast as possible. Proper selection and construction of datastructures, particularly with cache usage during the average system update in mind, can really help.



FX System Events

- Started with the basics, but kept low barrier to entry!

Spawn
Particles
Sounds
Meshes
Rigid Bodies
Decals
New Effects*

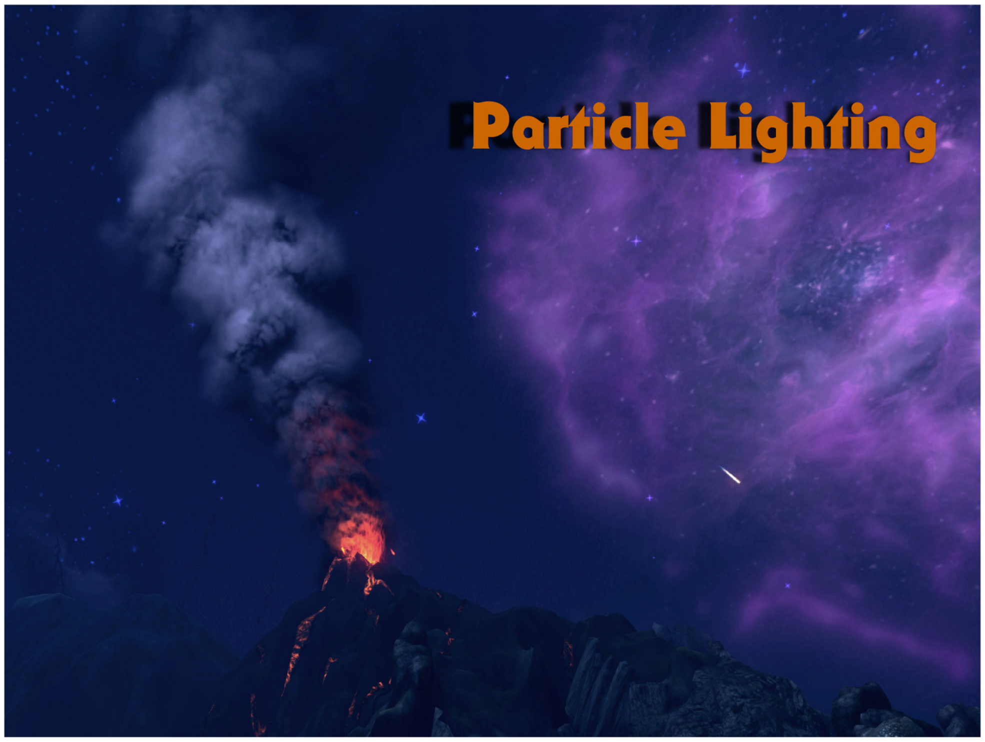
Modify/Trigger
Post Processing
Environment Settings (fog)
Material Modification
Material Assignment
Camera Shake/Rumble
Game Simulation Speed

- Very useful ability to help handle feature requests

Key to this system is straightforward way to add new effects, low overhead. If it is easy to add, people will add new ones often, and this leads to more toys, which of course is the overall goal. We started out with just particle events and added the rest over time as the need arose.

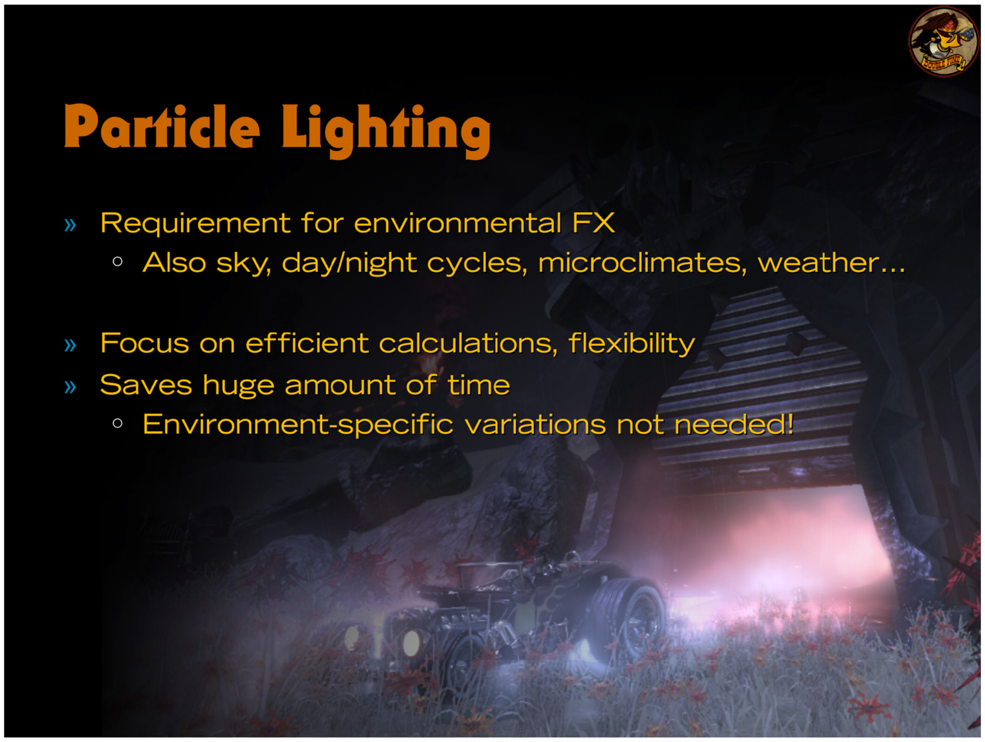
Also given the placement of the FX system, it is your opportunity to create a really clean interface for artists and other programmers to use. Simplicity and predictability are paramount.

We often have a large number (many hundreds) of Effects active, and thus it is absolutely essential for the FX system to be as fast as possible. Proper selection and construction of datastructures, particularly with cache usage during the average system update in mind, can really help.



Smoke is EVERYWHERE in BL. And making it look truly interesting, volumetric, and epic often boils down to how it is lit.

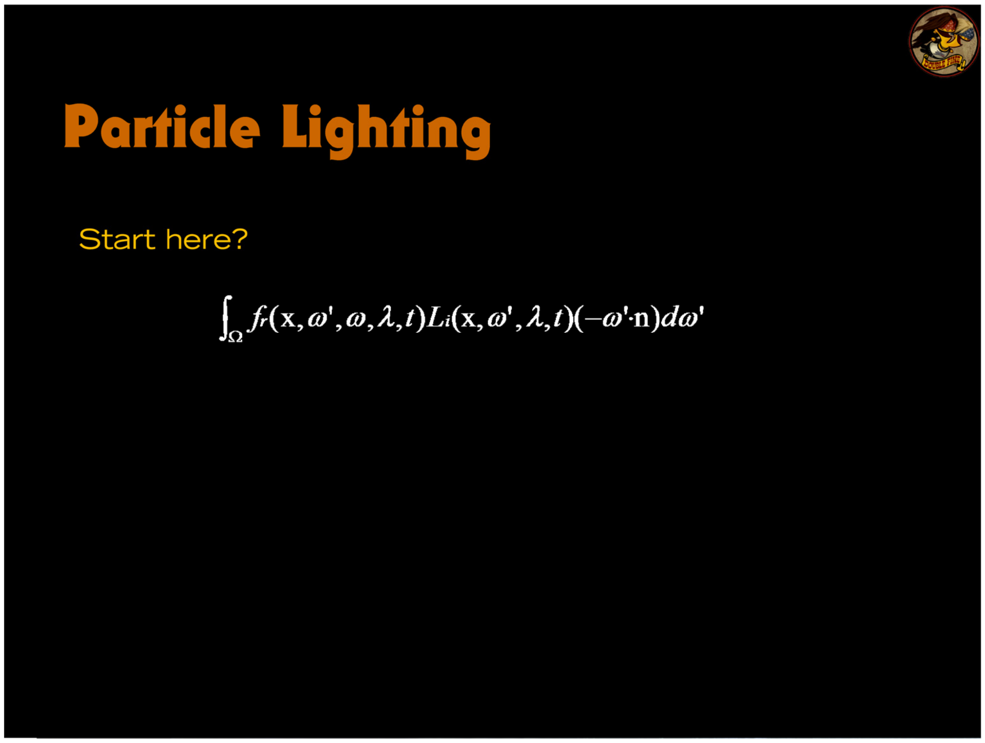
And since several of the massive landmarks in BL feature giant smoke plumes visible from most of the continent, we were hounded by the art team to make it look amazing.



Particle Lighting

- Requirement for environmental FX
  - Also sky, day/night cycles, microclimates, weather…
- Focus on efficient calculations, flexibility
- Saves huge amount of time
  - Environment-specific variations not needed!

We did know we had a long list of elements that would require some sort of lighting, but we weren't sure if all could be solved with the same solution… much of this did evolve over time. Likewise, it wasn't evident (until we started experimenting) that lighting things would be such a time saver.

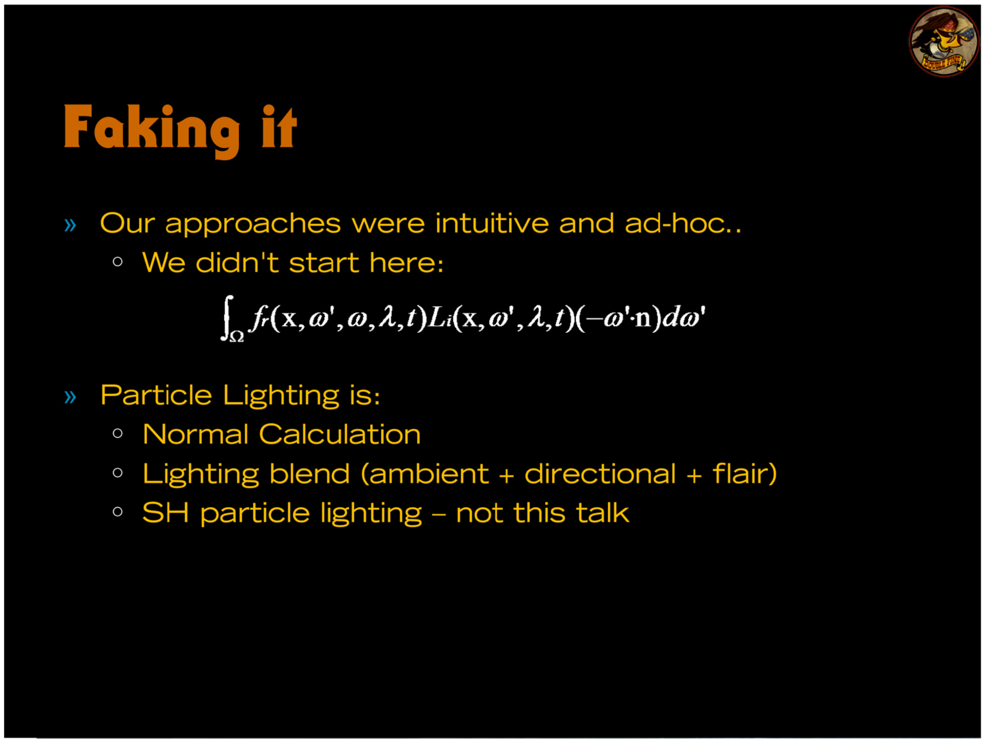


Particle Lighting

Start here?

$$\int_{\Omega} f_r(\mathrm{x},\omega',\omega,\lambda,t)L_i(\mathrm{x},\omega',\lambda,t)(-\omega'\cdot\mathrm{n})d\omega'$$

No.



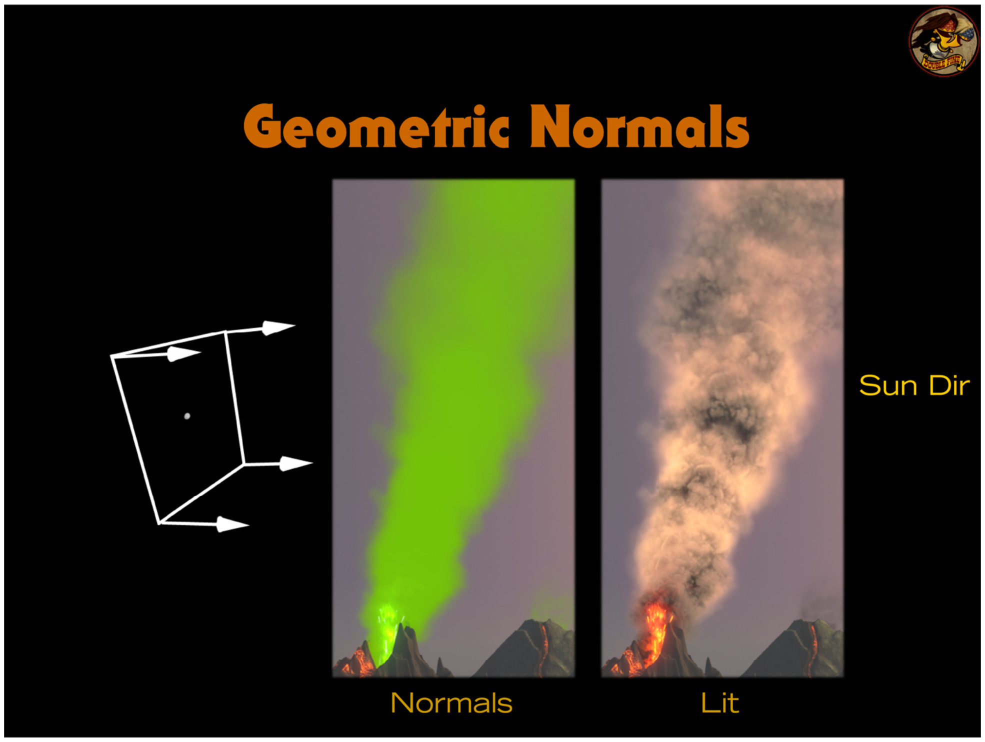
Faking it

» Our approaches were intuitive and ad-hoc..
  - We didn't start here:

$$\int_{\Omega} f_r(\mathrm{x},\omega',\omega,\lambda,t)L_i(\mathrm{x},\omega',\lambda,t)(-\omega'\cdot\mathrm{n})d\omega'$$

» Particle Lighting is:
  - Normal Calculation
  - Lighting blend (ambient + directional + flair)
  - SH particle lighting – not this talk

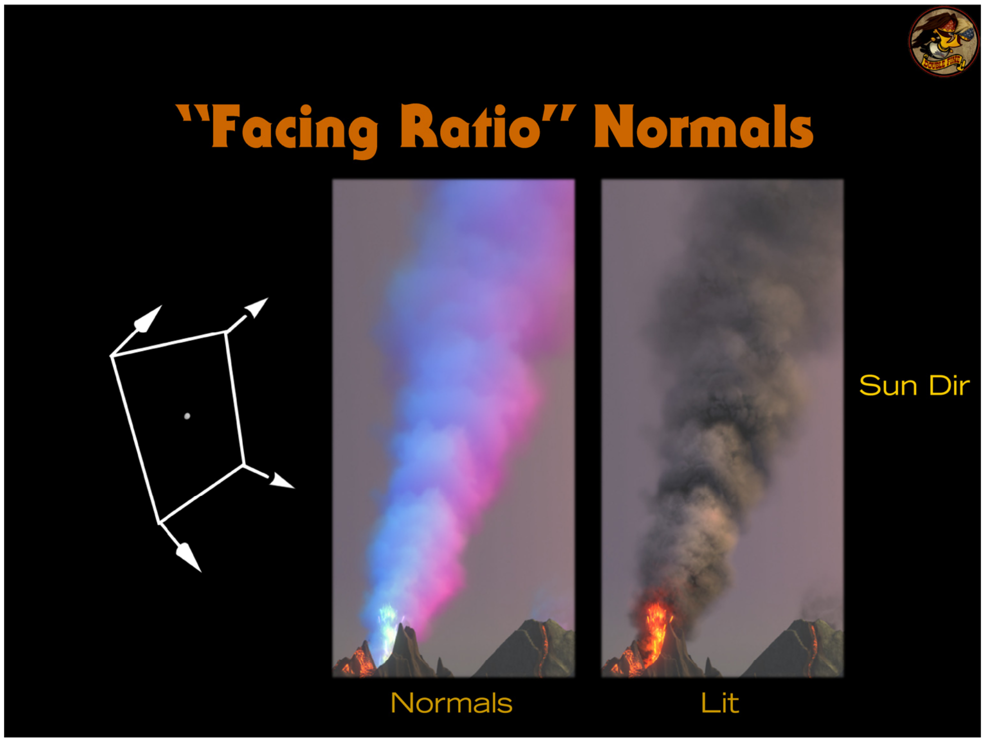
Lighting = Lighting Math (Normal) + “Artistic terms”



Geometric Normals

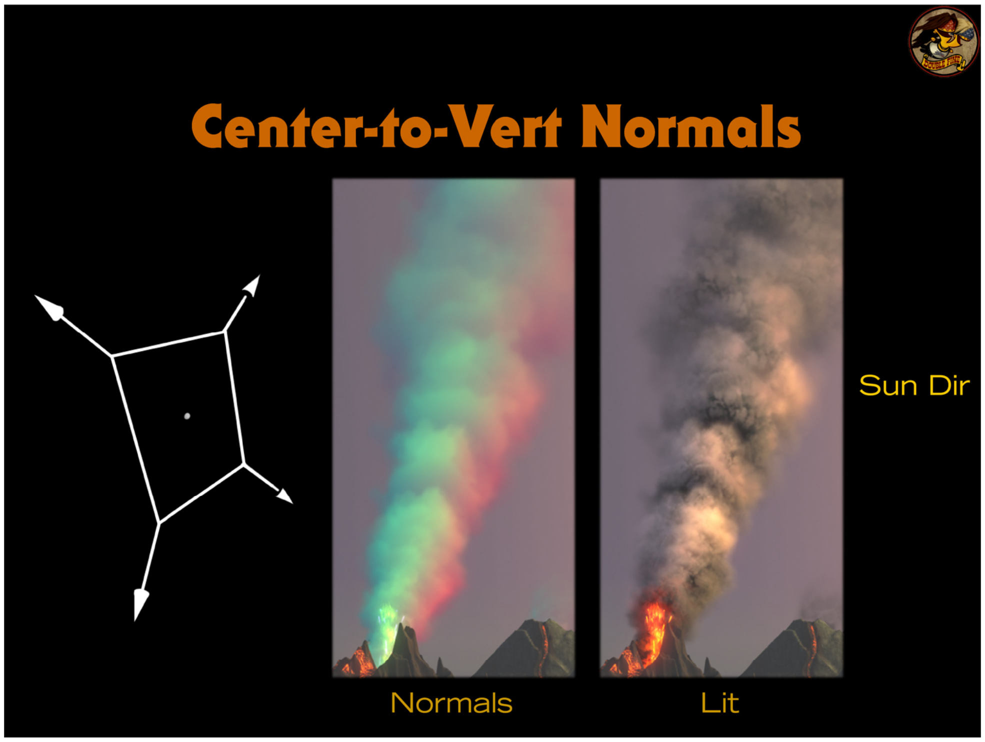
The naive implementation is to simply slap some normals on your billboard and call your standard, full on mesh lighting function.

This has got the right color (sort of), but it doesn't look like it has volume.. and it's slow to render.



"Facing Ratio" Normals

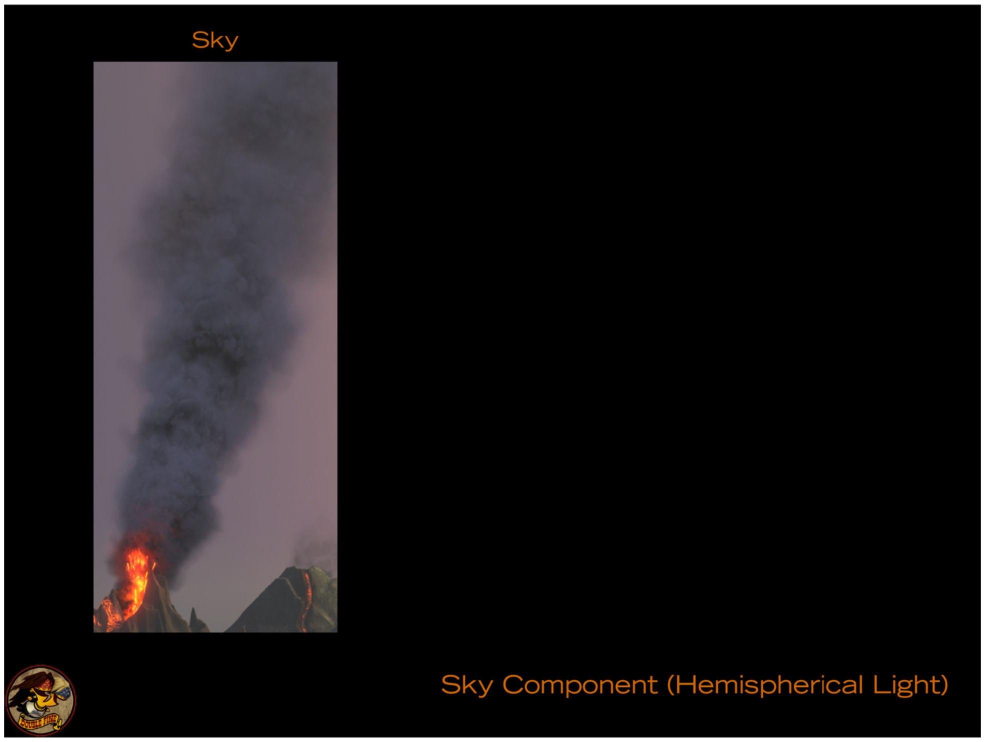
Digging around through various papers and experimenting turned up some more interesting normal generation methods. We stuck with this one for a long time (facing ratio 45 deg offset). It made a big difference, but still doesn't give us the dramatic lighting that we want. It also showed too much variation when the camera moved.



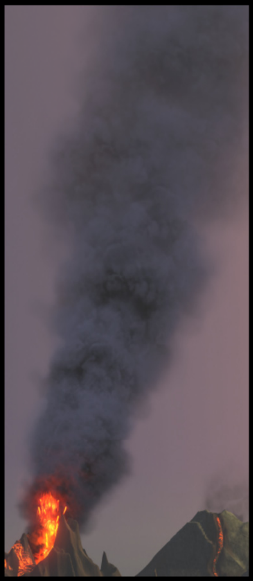
Center-to-Vert Normals

So, let's hack things up. We want a LOT of lighting variation. In fact, we want the MOST lighting variation we can have. So push those normals all the way out - to each vert of the quad. You're basically approximating a sphere. It's also incredibly cheap, and changes less than facing ratio normals when the camera moves.

It's worth noting we had a few other normal generation methods, including generating the normals based off the velocity of the particle. Most of these are specific to certain classes of visuals, but are not used nearl as much as the general techniques.

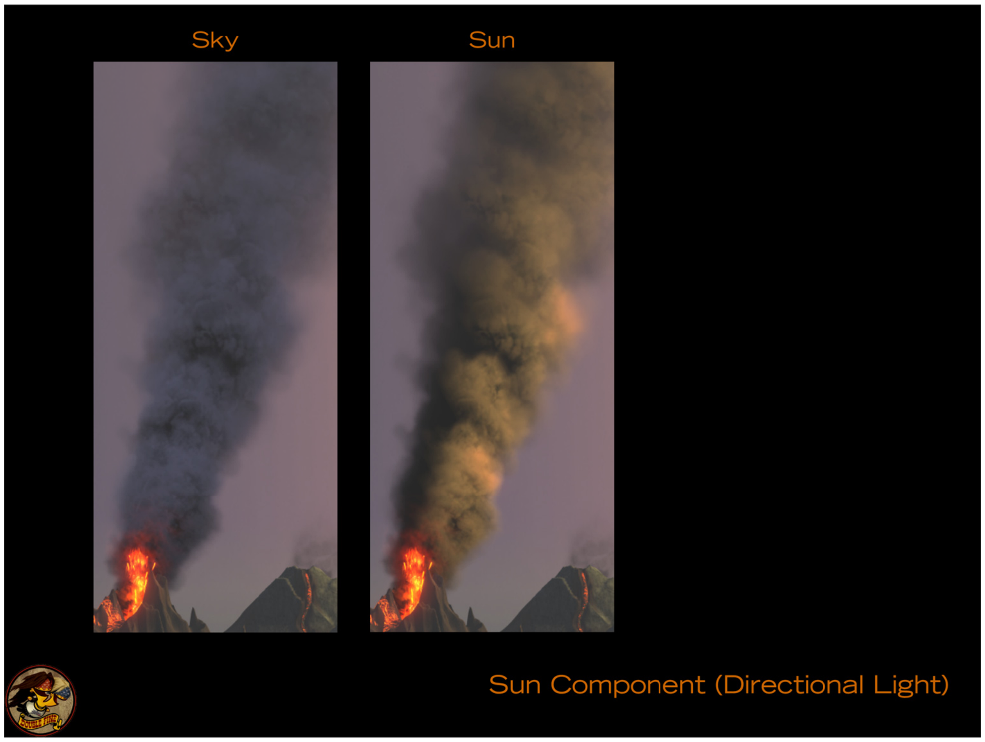


Sky



Sky Component (Hemispherical Light)

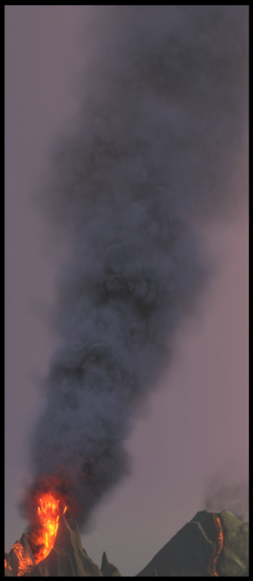
Ambient sky component, which uses a "RSA" light – essentially an artist authored color ramp. See "Radially-symmetric reflection maps" by J. Stone from the Sketches section of SIGGRAPH 2009.

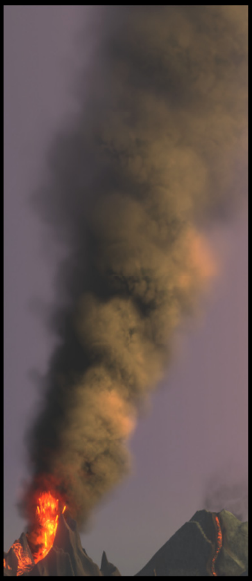


Sun Component (Directional Light)

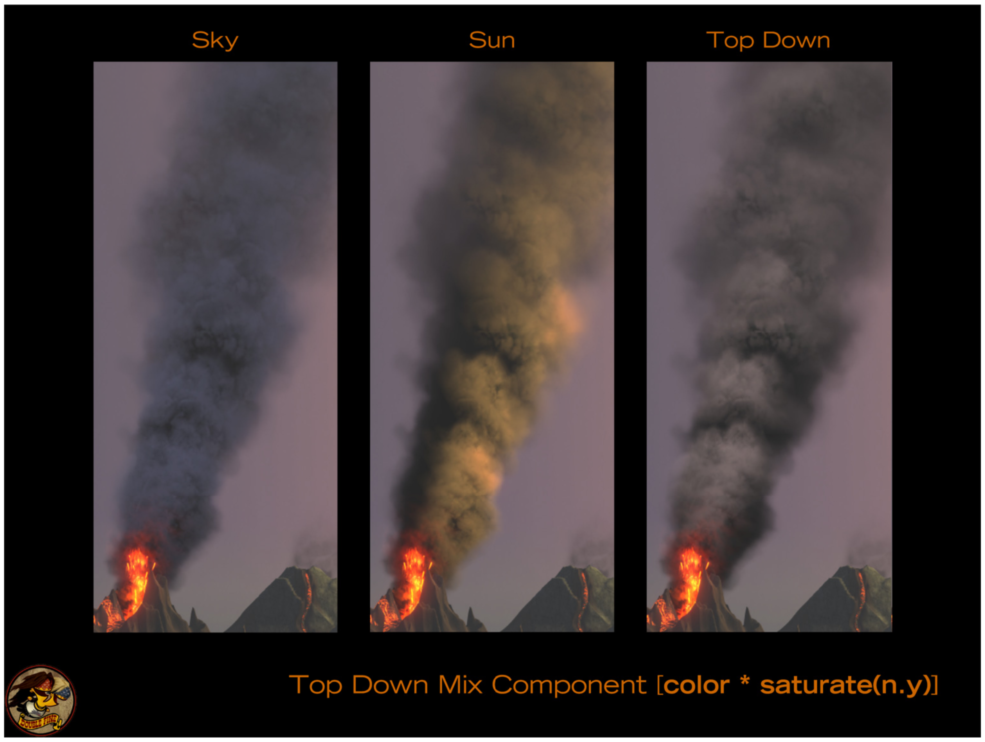
Sky

Sun



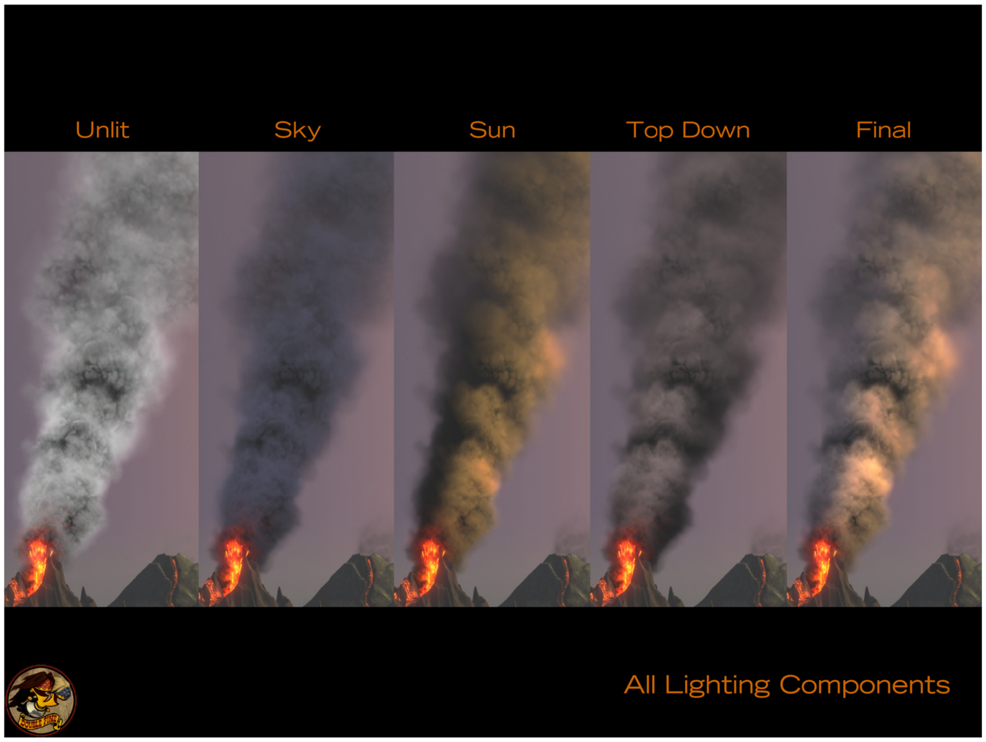


Directional component from the sun. May also use a RSA light, or some simple approximation of such.



Top Down Mix Component [**color * saturate(n.y)**]

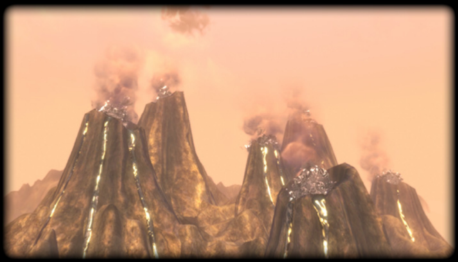
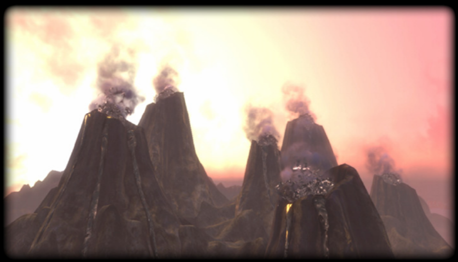
“Top down mix” component – color authored by the artists per lighting environment.



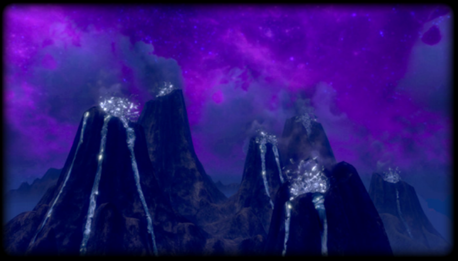
Lighting component breakdown, using center-to-vert normals.

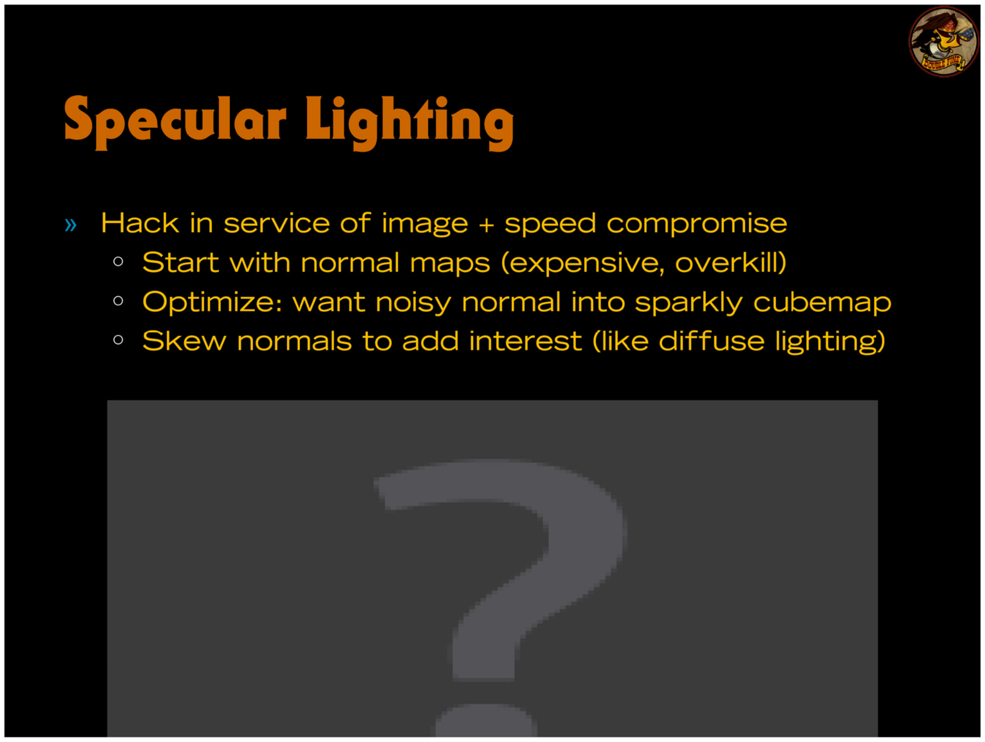
We allow the particle system to provide independent weights to combine each of these components. We also allow custom control of how much each particle gets fogged, as because our particles are so far away, we're often fighting against our fogging values to get the drama in the lighting that we want. From these control, we can get lighting that feels dramatic, responds to time of day changes well, and total gpu cost is within our limits.

We worked on many revisions of the lighting code – cleaning up after ourselves was a huge benefit. As much of the time the new lighting code was a generalization or improvement on a prior method, we could often automate the conversion from the old code’s parameters to the new. A python script that can parse your particle systems, perform changes, and write them back out has come in handy time and time again. The ability to batch update systems is one of the major reasons we are not stuck with a backlog of little-used, semi-broken features that would make developing new ones even more difficult.

Tim wants ***Chrome*** Volcanoes…





Specular Lighting

» Hack in service of image + speed compromise
  - Start with normal maps (expensive, overkill)
  - Optimize: want noisy normal into sparkly cubemap
  - Skew normals to add interest (like diffuse lighting)

And for chrome, you need specular lighting. Normal approaches to specular are generally a bit more expensive. And you really want some sort of normal-map type approach, because adding a big phong highlight to a particle is basically the last thing you would ever want.

So dropping in normal mapping and a basic, directional specular highlight into this does work, but it's slow. And this is such an effective way of describing wet surfaces, that we really want to be able to use it everwhere. So let's sacrifice accuracy for performance – and who really needs accurate details for fast moving fx? We just want a fast shimmery sparkle in most cases to make it sell a wet look.

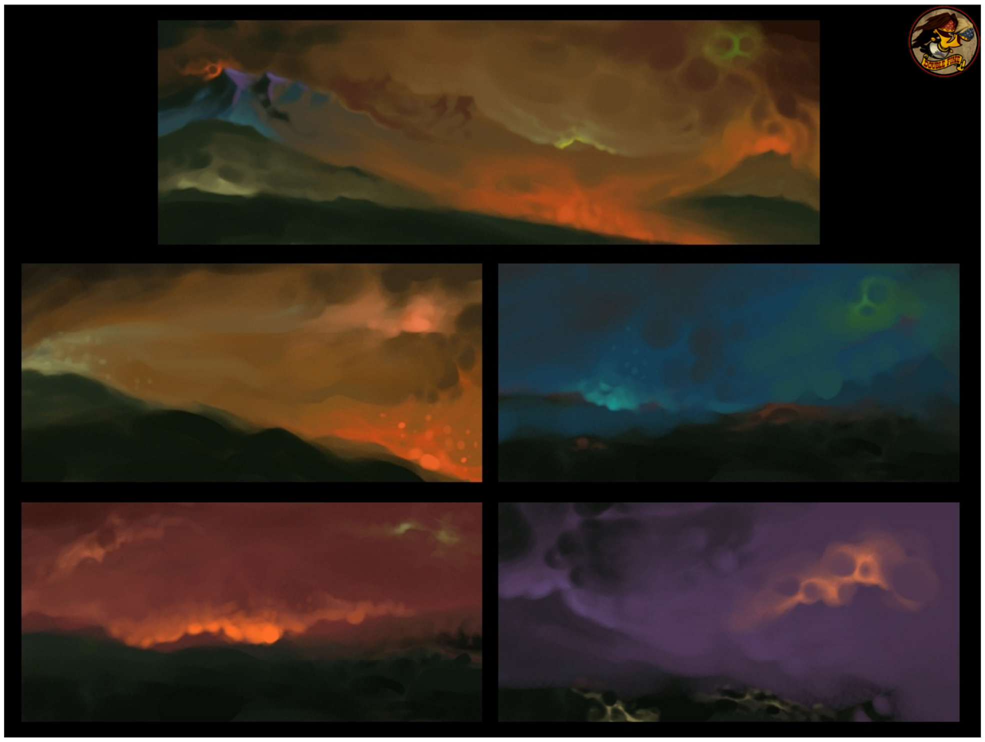
So we just came up with simplified logic to perturb a normal into a high frequency "sparkle" cubemap/texture, and this gave the basic look we desired. You can essentially start with normal mapping, and simply remove everything that would make it traditionally "correct", and you'll have something that is much cheaper and still sort-of works. Adding in the same normal warping we perform for the diffuse lighting also helps make it more interesting, and made it less camera dependent (it's quite distracting when the sparkles are very clearly moving with the camera instead of the particles).

NIGHT (FOG)
FOGGY DAY
DUSK (FOGGY)
DUSK
Sky,
Weather,
Environment
MID DAY
DUSK

Like so many aspects of this game, the Brütal Legend climate is one of extremes:

* Dynamic time of day
* Dynamic weather
* Regional variation
* World state variation
* Striking sunrise, sunset moments
* Epic constellations
* Aurora borealis
* Skull moons
* Vortices
* Lee Petty, our art director, often suggested a rain of flaming frogs. I'm pretty sure he was serious.

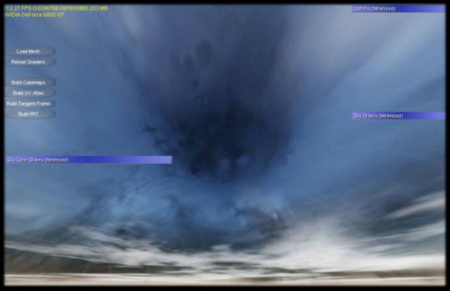
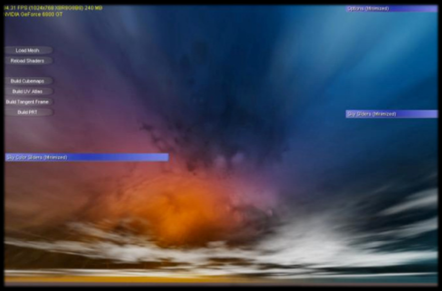
Time to take our ability to create lit particles, and discuss how we developed the styled sky and environment rendering.



Concepts like these provide a good summary of our initial goal. Frazetta-like colors and patterns creating the sky, all seamlessly changing.

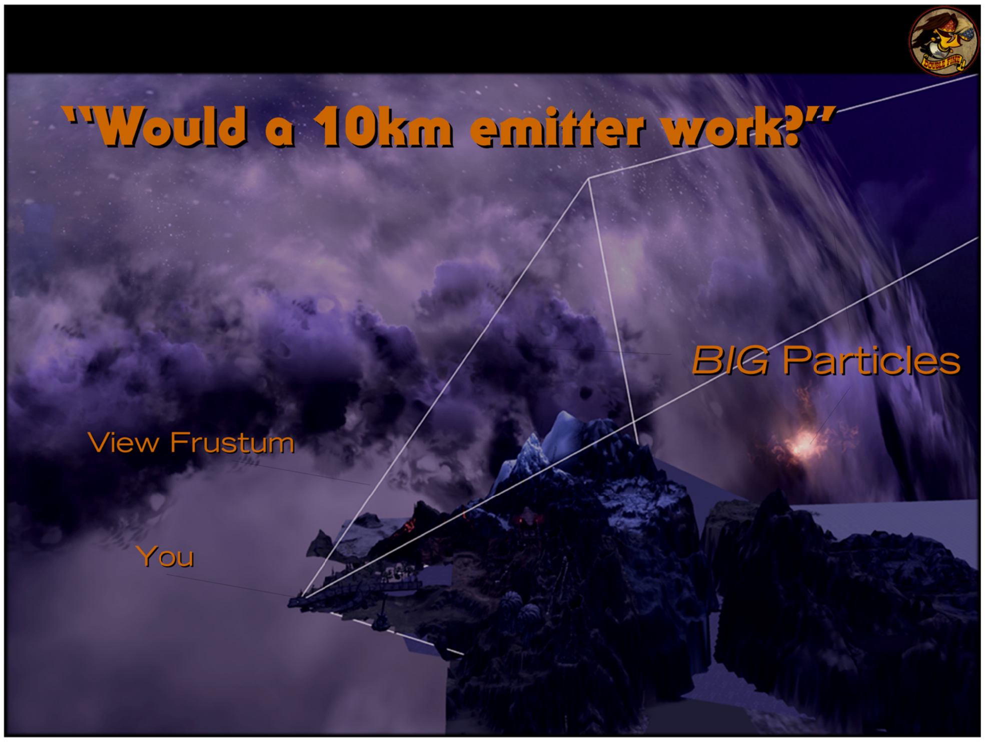
Initial Approaches to Skies

- First attempts were lacking
  - Cubemaps for sky and image based lighting
  - Custom shaders with cloud textures
  - Complex/physically based light scattering models
  - Most damning: static nature, authoring time

Summary is that the tech was really getting in the way here, especially in terms of artist time and the how predictably they could create images from these somewhat black-box systems.

What we really needed was a madman.. with no respect for the rules… or gpu costs…



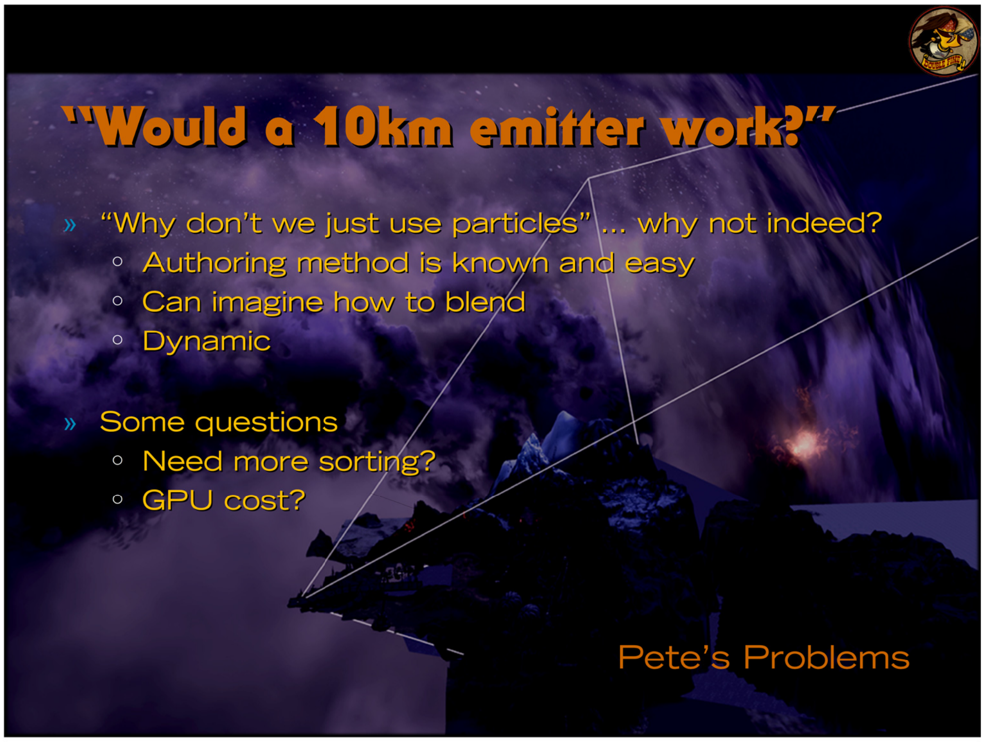
Only when we were too fed up with the old approaches did we forget all the obvious reasons particles wouldn't work.

But equally evident it would after we gave it a try and hashed out some of the problems!

We also ended up adding a few extra emitters and other parameters particularly for the sky particles, but these were minor.

Many of the edge cases were handed by using existing features (sorting forcing/ordering, etc)

(Sky specific things included emitter that spawns particles at configurable shell with furthest point at far plane, probably the most useful to have right away)



"Would a 10km emitter work?"

- "Why don't we just use particles" ... why not indeed?
  - Authoring method is known and easy
  - Can imagine how to blend
  - Dynamic
- Some questions
  - Need more sorting?
  - GPU cost?

Pete's Problems

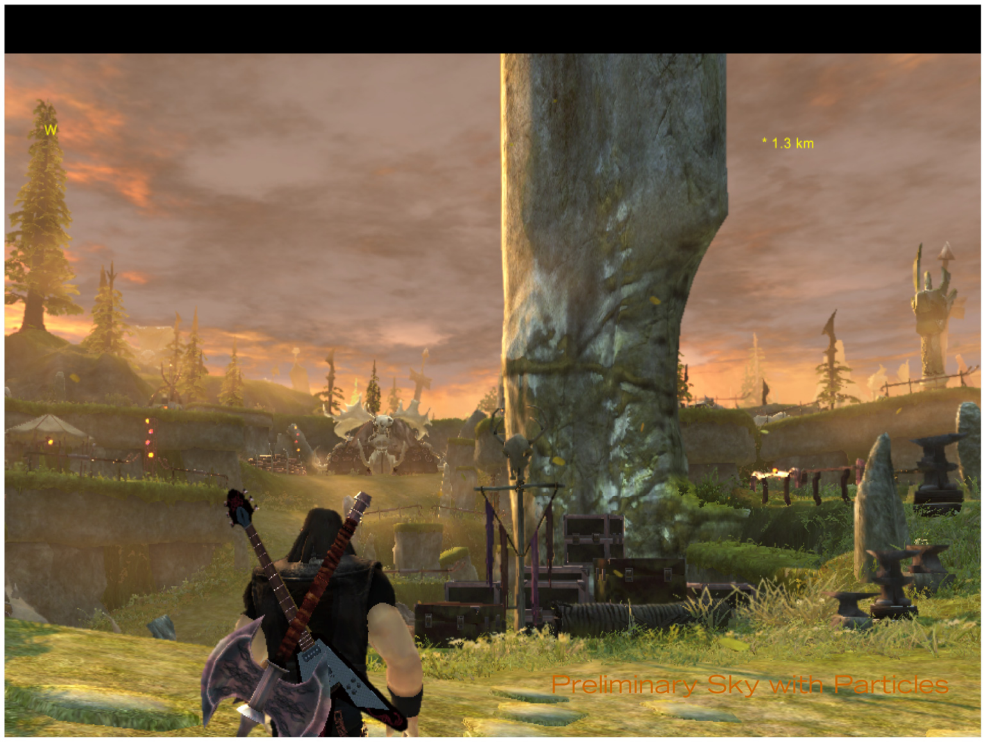
Only when we were too fed up with the old approaches did we forget all the obvious reasons particles wouldn't work.

But equally evident it would after we gave it a try and hashed out some of the problems!

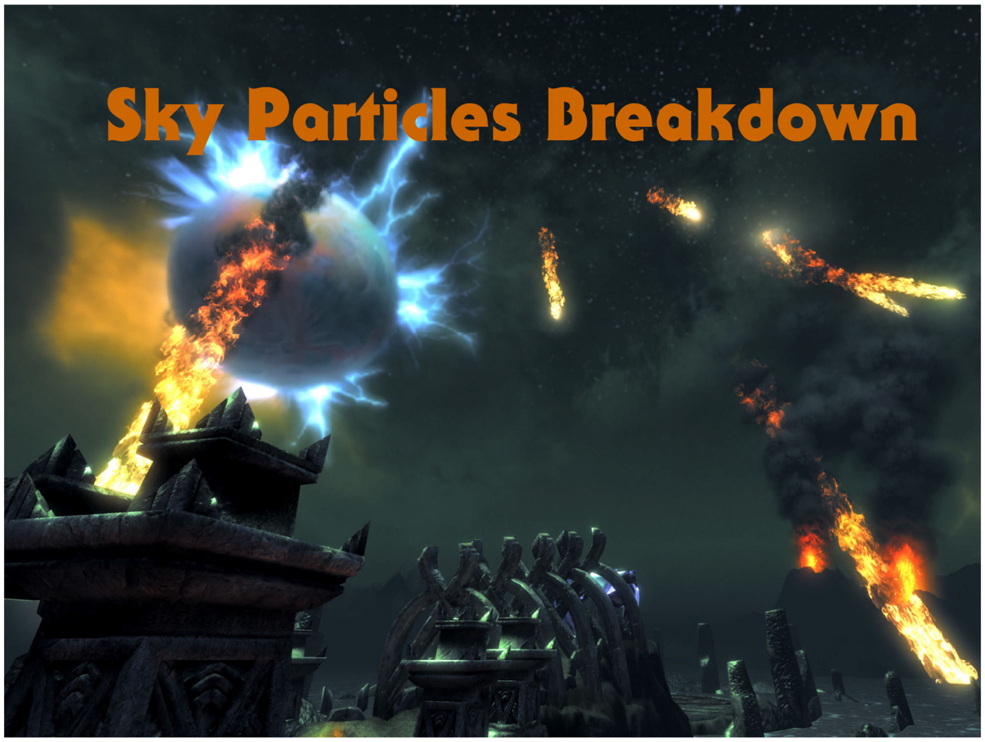
We also ended up adding a few extra emitters and other parameters particularly for the sky particles, but these were minor.

Many of the edge cases were handed by using existing features (sorting forcing/ordering, etc)

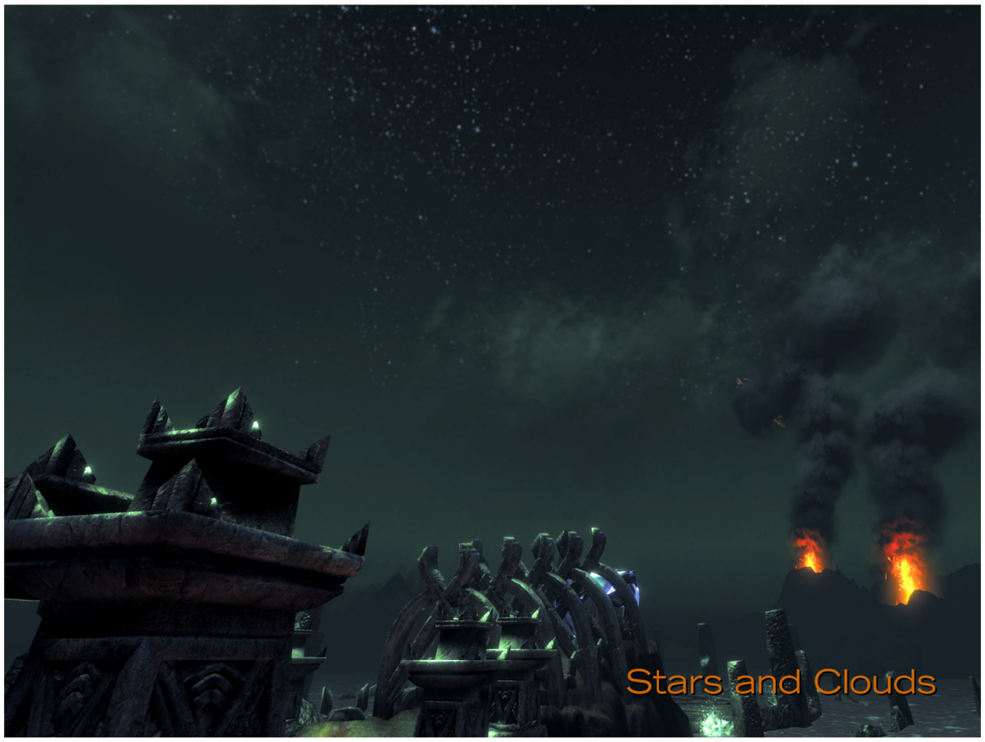
(Sky specific things included emitter that spawns particles at configurable shell with furthest point at far plane, probably the most useful to have right away)



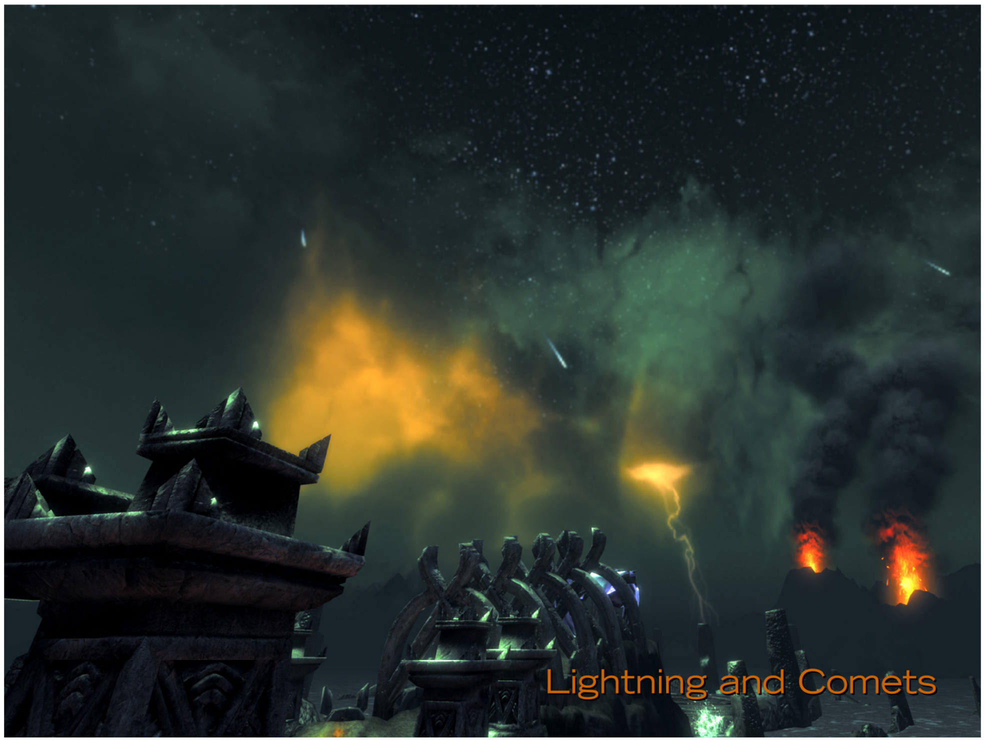
Results of some initial particle sky tests, convincing enough to keep us going.



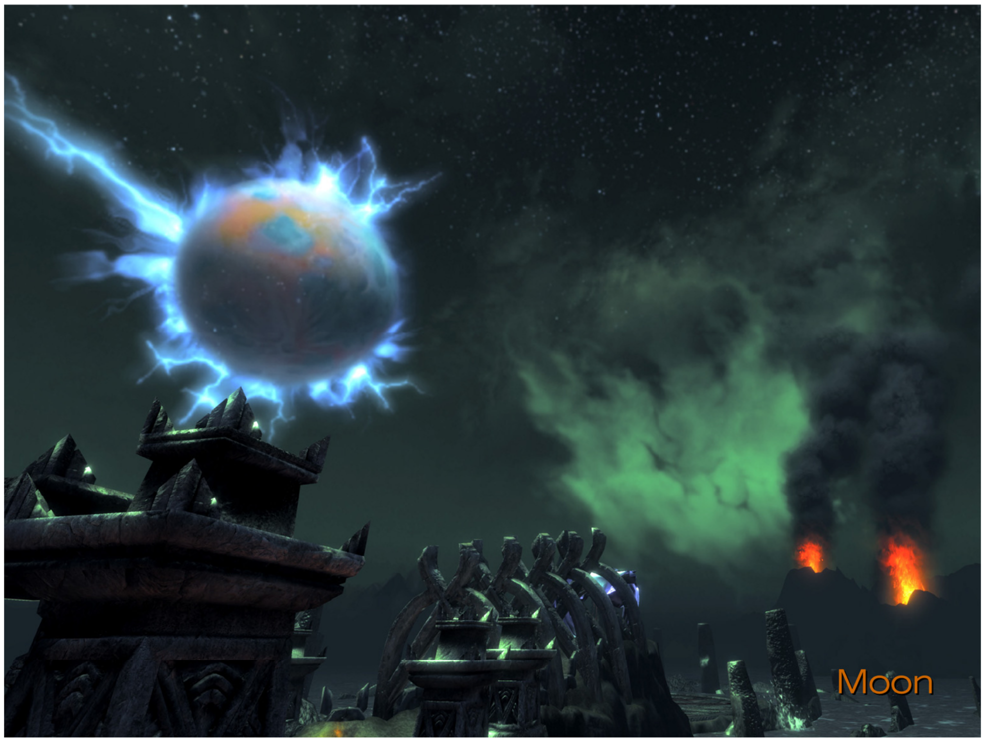
From a sequence during the player’s first introduction to the world.



Start with some particle events that add stars and clouds.

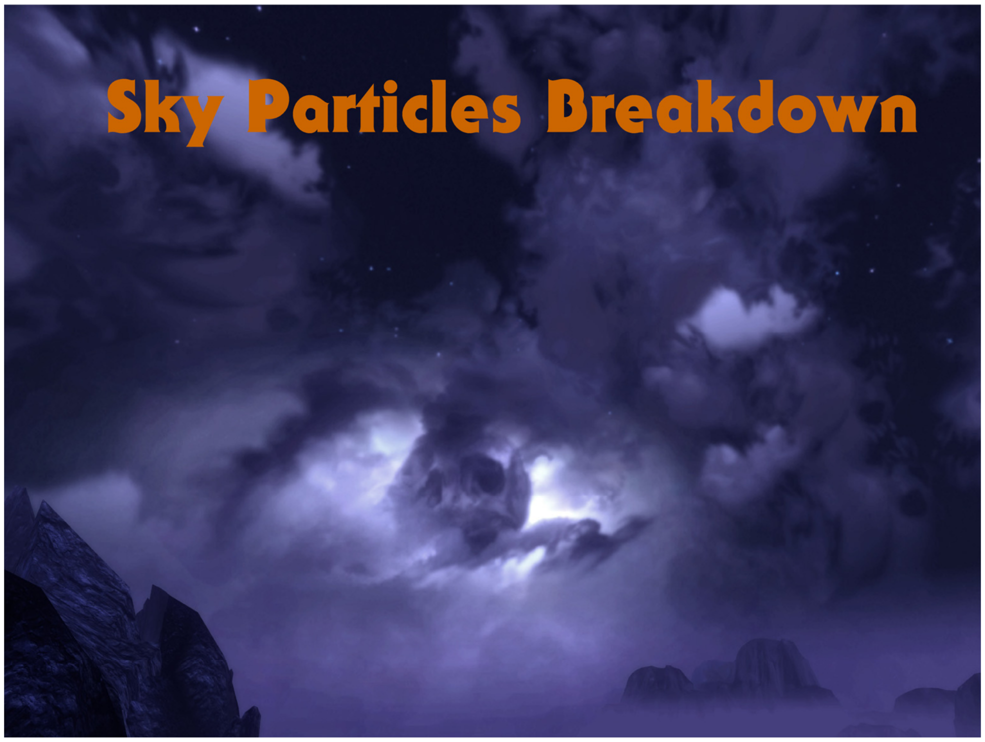


Add some random particle events for lighting, bright flashes using point light events.

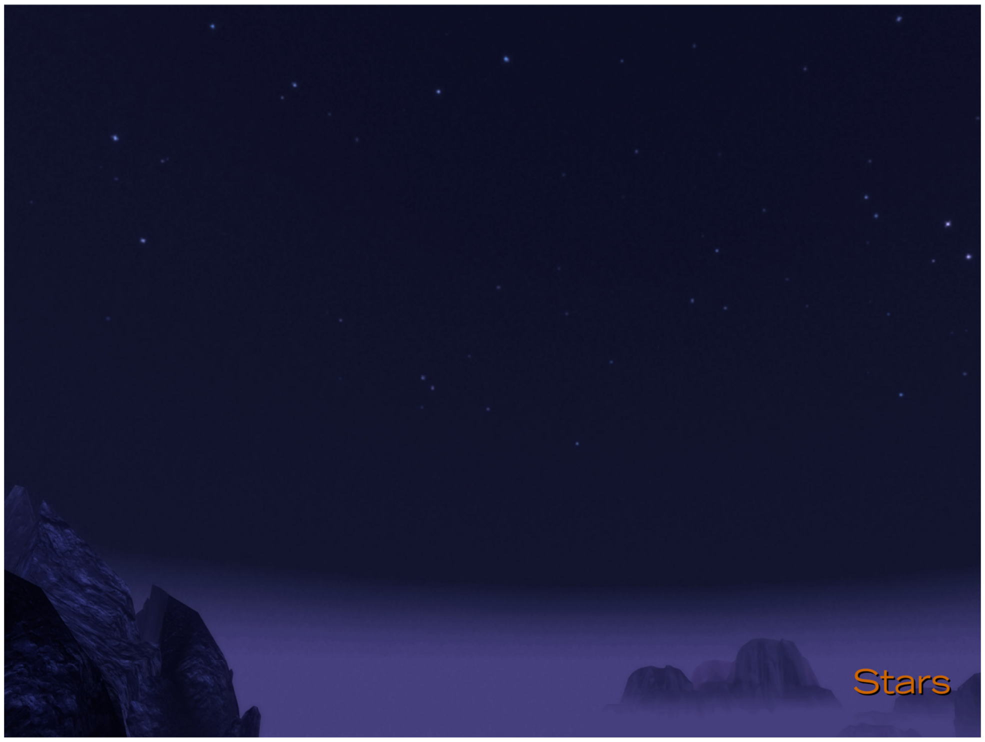


Several layers of particle events to create the crazytime moon.

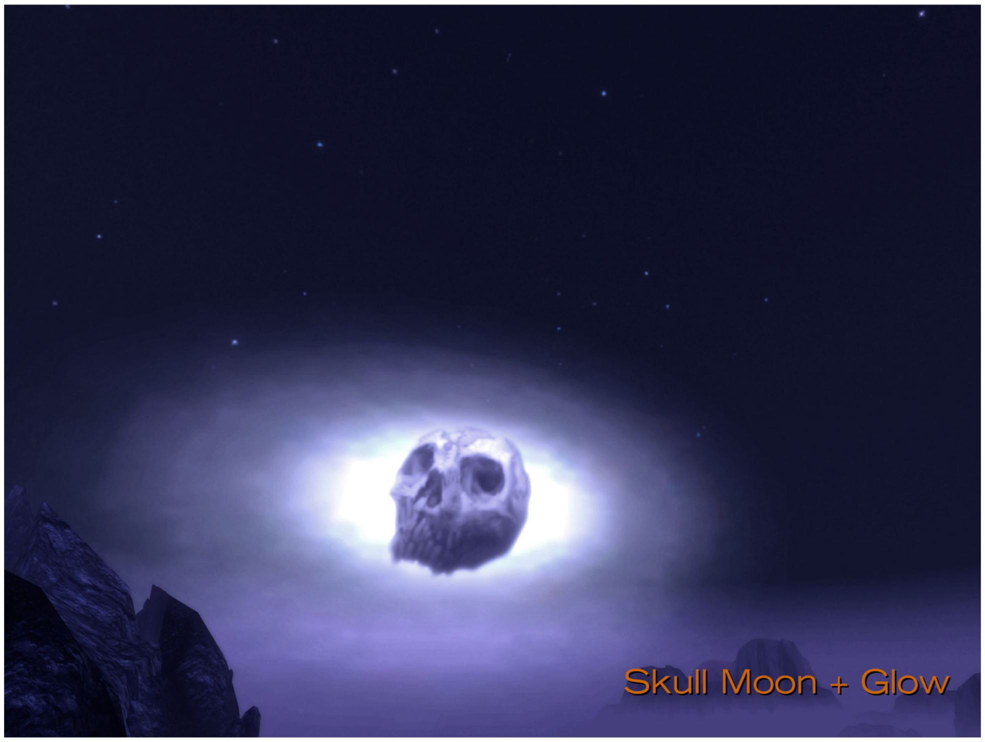
And add some rigid body events that themselves have effects to leave trails.



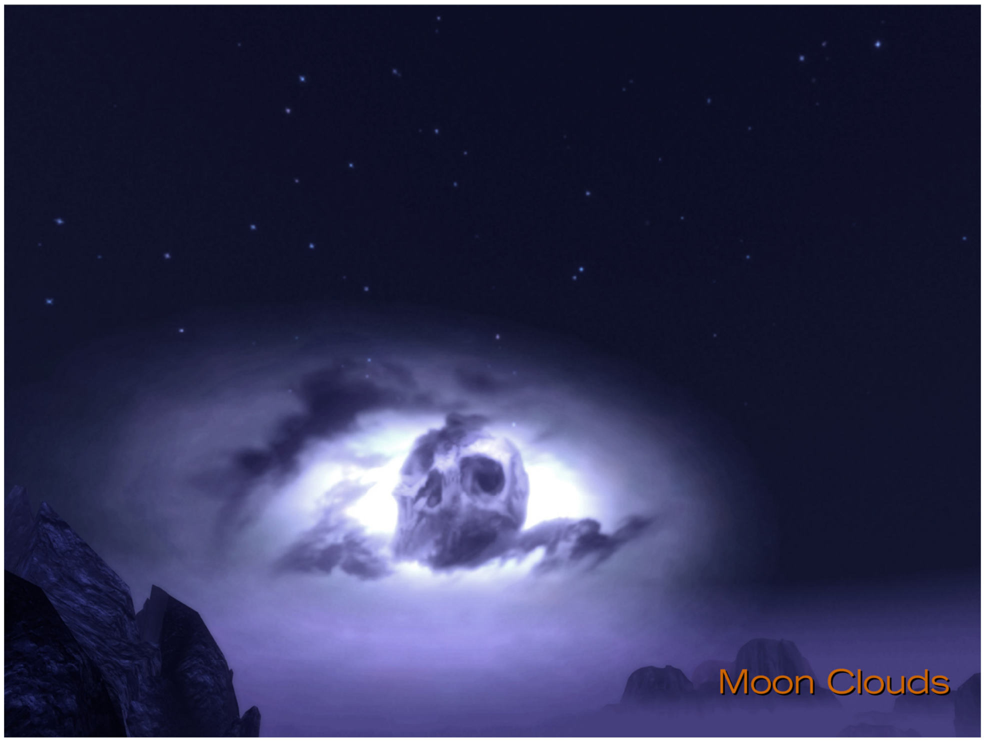
A more painterly sky that needed to have specific elements present.



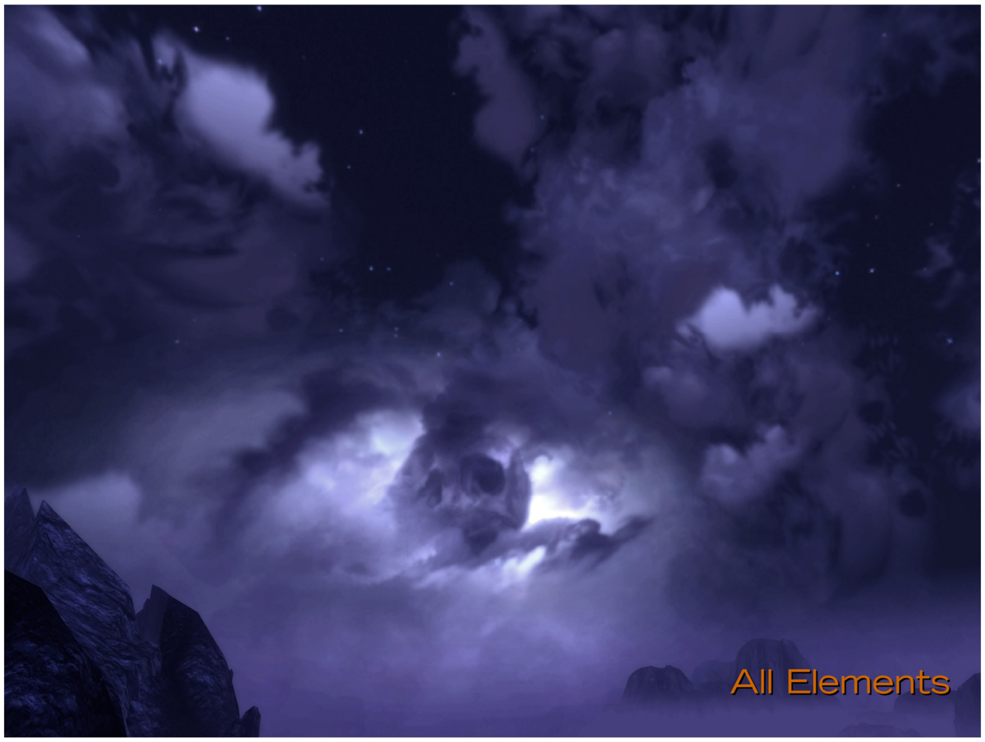
Particle events for stars.



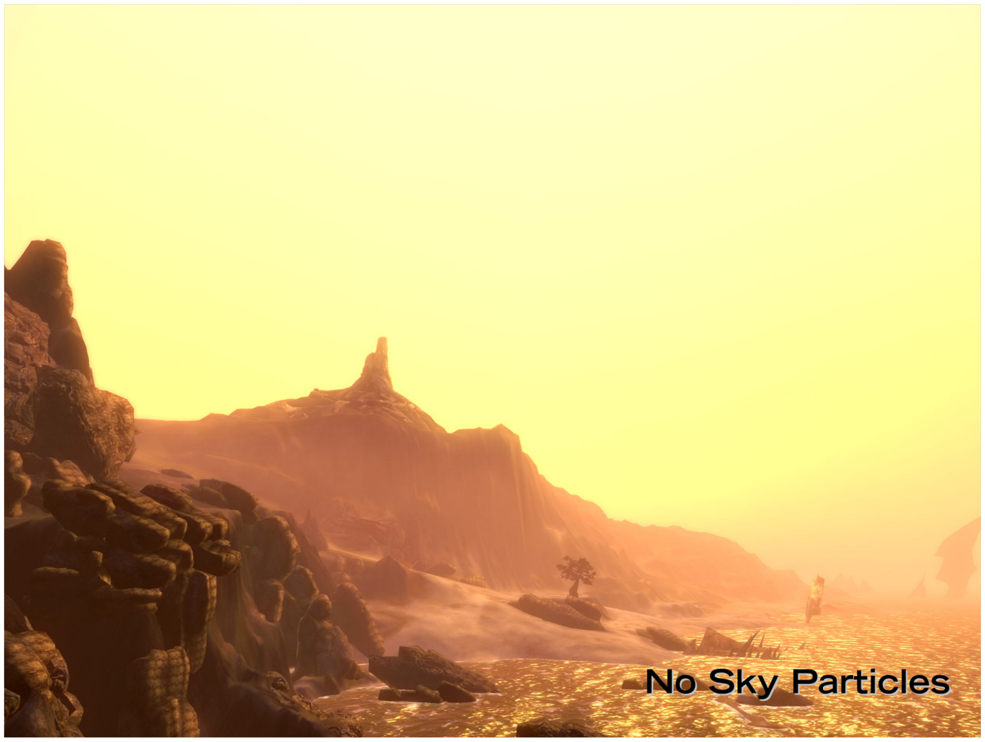
Layered particle events to add a background glow and moon. These are set to always draw behind all the other particles besides the stars. The glow looks pretty silly for now..



Particle events for clouds, start covering up the glow particles to give the clouds a backlit look.

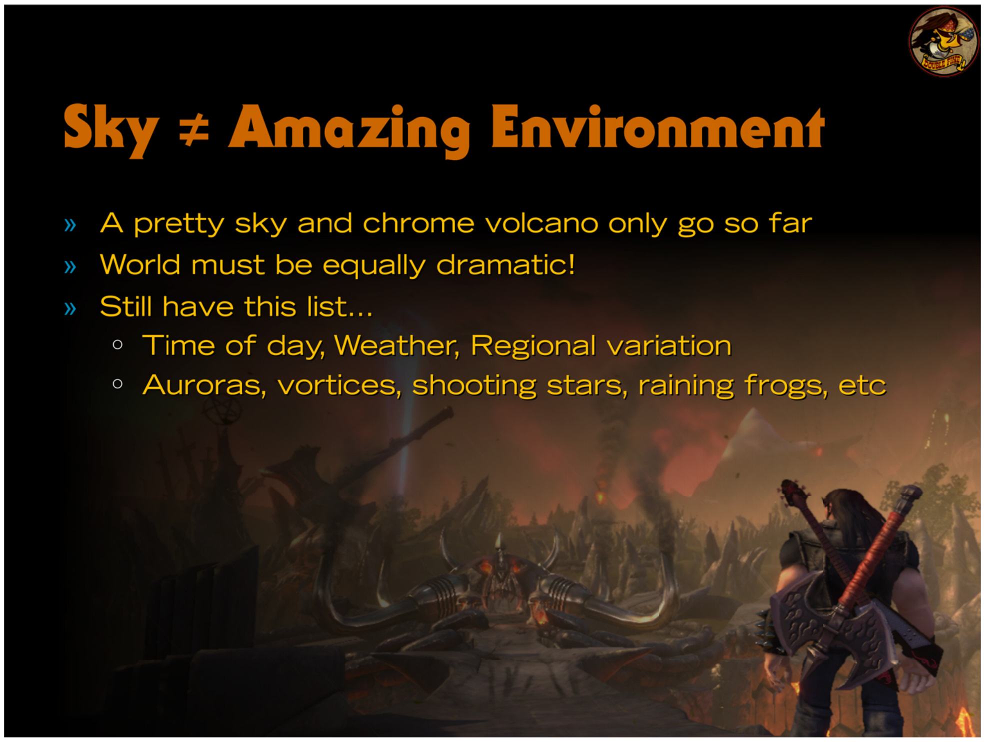


The glow finally looks less silly, and gives a nice looking backlit look to the clouds.


No Sky Particles

Layered Cloud Particles

Sometimes a few layers of lit cloud particles just look great.

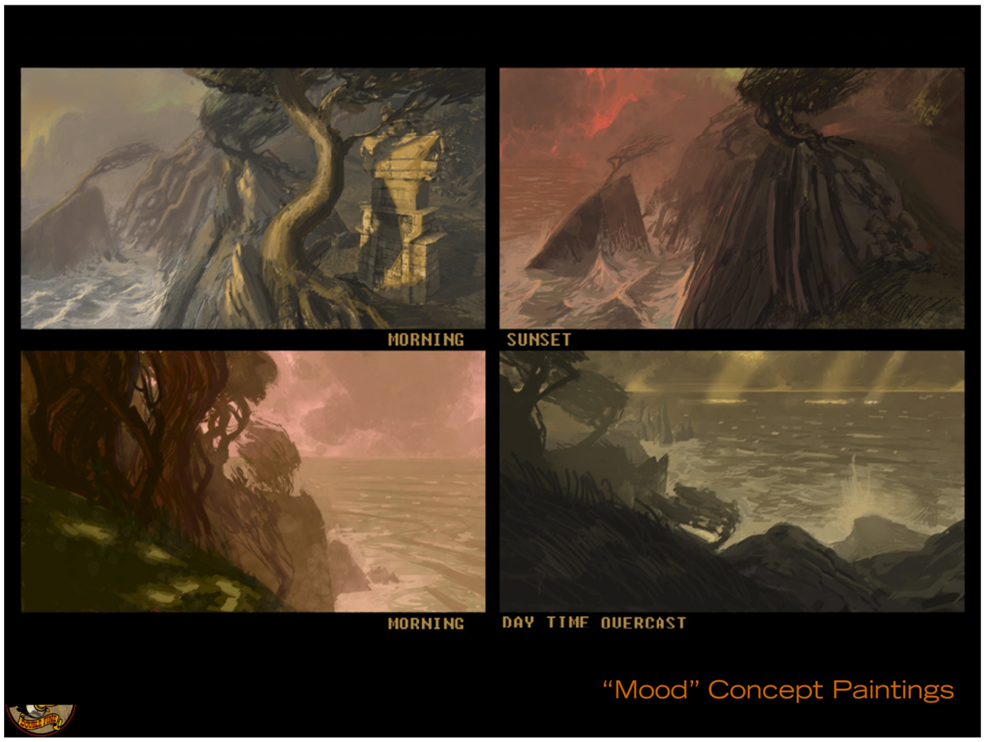


Sky ≠ Amazing Environment

- A pretty sky and chrome volcano only go so far
- World must be equally dramatic!
- Still have this list…
  - Time of day, Weather, Regional variation
  - Auroras, vortices, shooting stars, raining frogs, etc

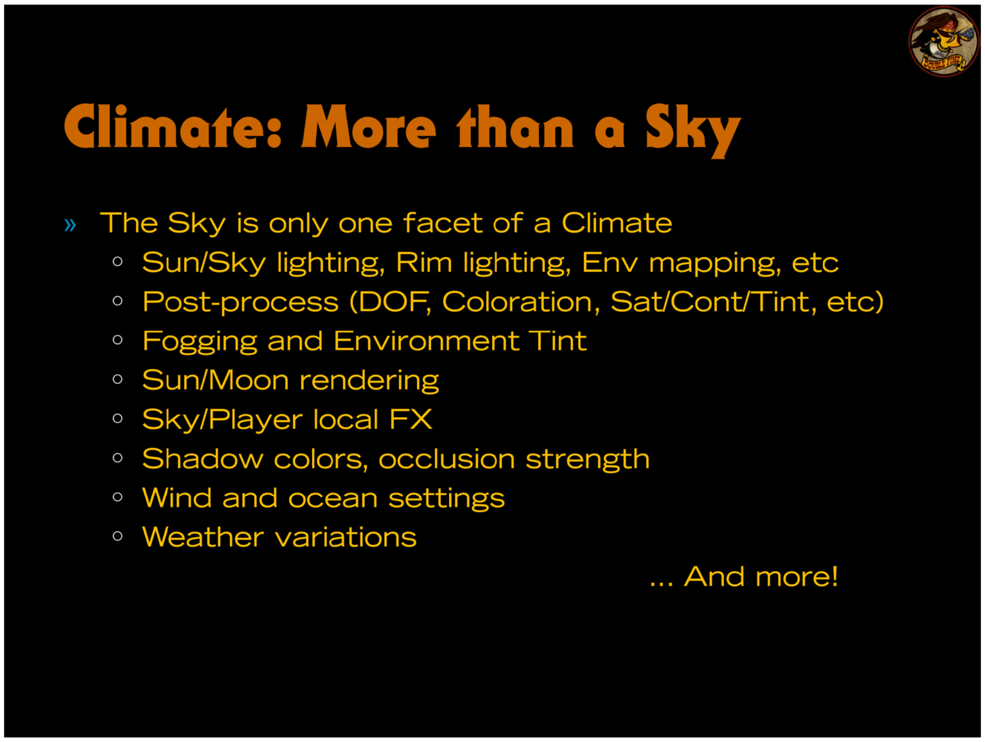
This was evident even when looking at concepts/reference art… and here we are.

And it's also time to start looking hard at the rest of the giant list of requirements we have for skys/climates.



Mood paintings make it more evident that the sky plays a large part, but there is much more than just the backdrop going on.

Color palette, lighting, fog, etc all play a role.



Climate: More than a Sky

- The Sky is only one facet of a Climate
  - Sun/Sky lighting, Rim lighting, Env mapping, etc
  - Post-process (DOF, Coloration, Sat/Cont/Tint, etc)
  - Fogging and Environment Tint
  - Sun/Moon rendering
  - Sky/Player local FX
  - Shadow colors, occlusion strength
  - Wind and ocean settings
  - Weather variations

… And more!

A pretty big pile of parameters that can modify various rendering parameters all throughout the game. The ability to modify so many things was some time in coming (and reaches throughout our material system/renderer, etc) but was easy to add artist controls to once the underlying code was in place.

Organizing it into various sections (Lighting, Fogging, Sky, Environment FX, Post, etc) helps keep it more maintainable.

Some of the many weather features that artists could tune include

All lighting functions such as sun & sky color, intensity and texture;

Lit Sky Particles.

Precipitation (camera based effects). This allowed us to add more human scale effects around the player, weather it be rain, dust, ash, or swirling blood mist.

Vertical Fog. Given the scale of our game, vertical fog was a necessary component to create depth without entirely fogging out he background.

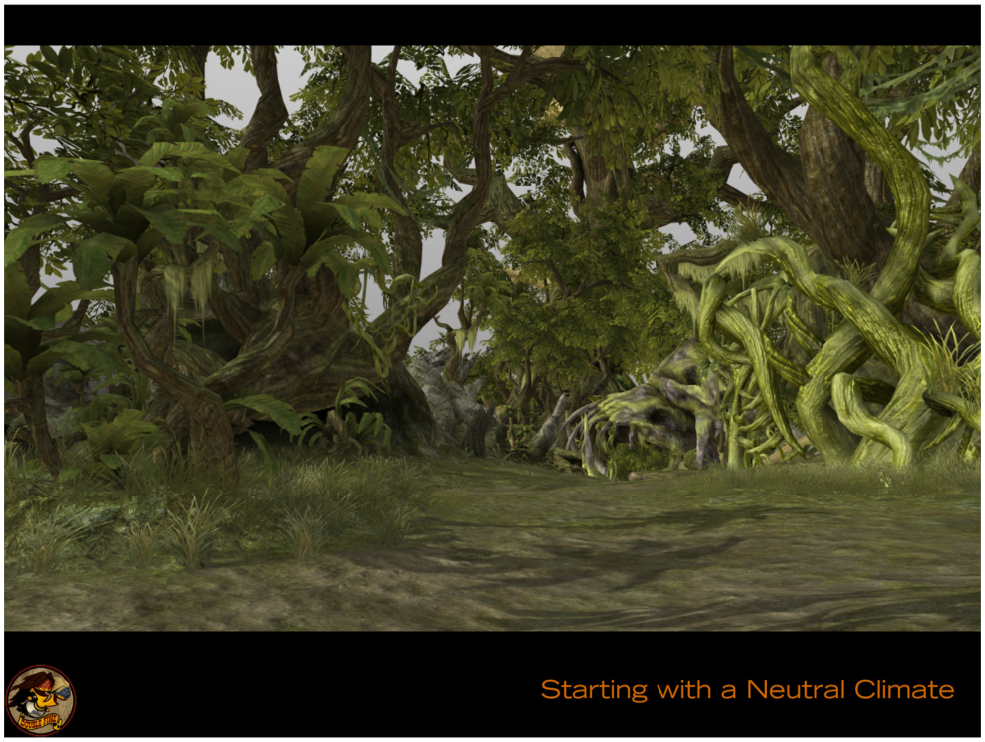
Post effects controls. We had a rich suite of post effects for developing the mood of any given time of day including Color correction, DOF Blur, Contrast and Saturation adjustment.

Our weather files also had hooks into all of the standard game material parameters, so we could more directly affect any objects or characters in the environment. For example, if we wanted to help give the illusion of wetness in a rainy weather, we could increase the rim lighting and environment mapping intensity of all of the objects in that weather.

Climate Elements Breakdown

It’s easier to illustrate what all this does by detailing the progression of an image from neutral settings to the environment’s natural settings.

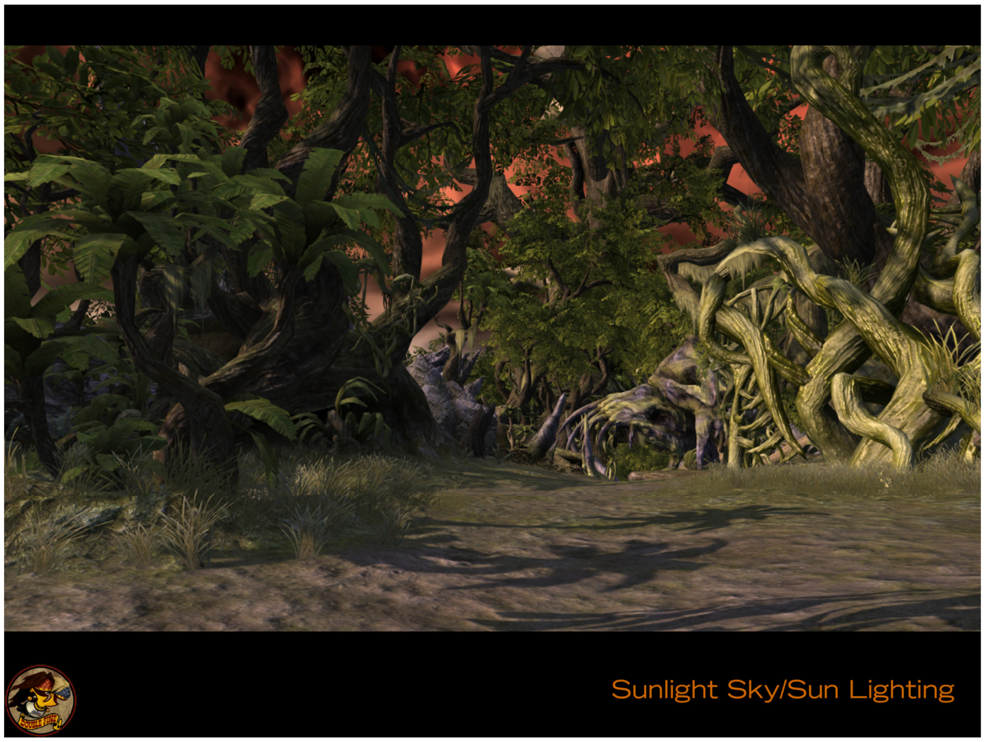
Note all these are with the time of day fixed and a single weather variation selected for all images.


Starting with a Neutral Climate

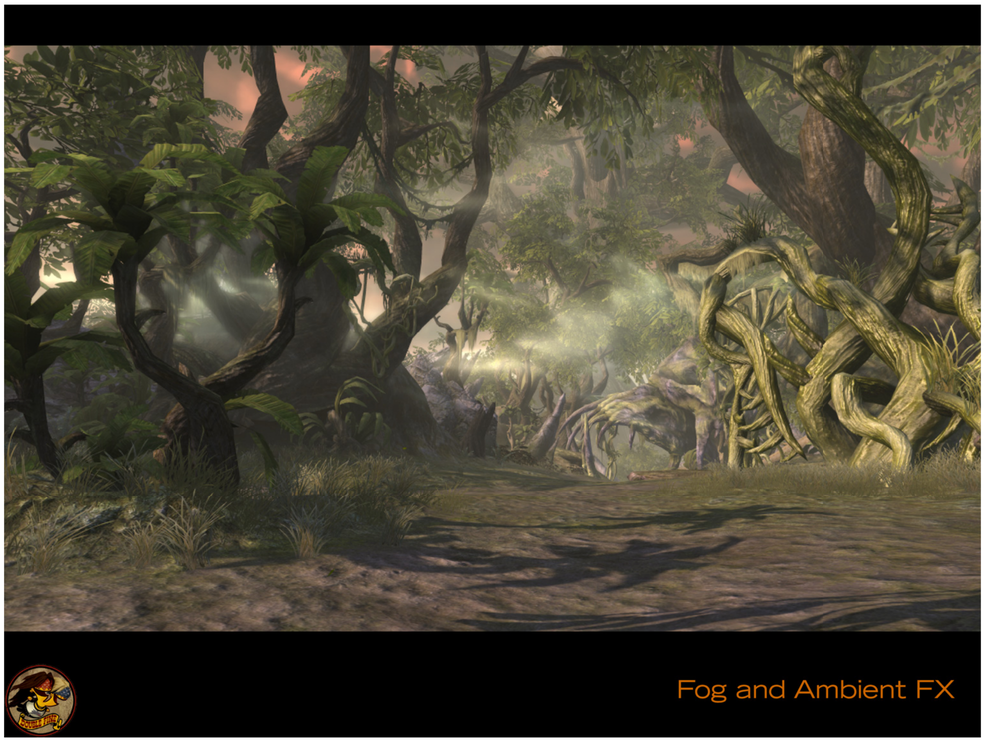
Neutral, white light. Not much contrast between sun and sky.

Sky Particles

Adding a dramatic sky behind all the trees.

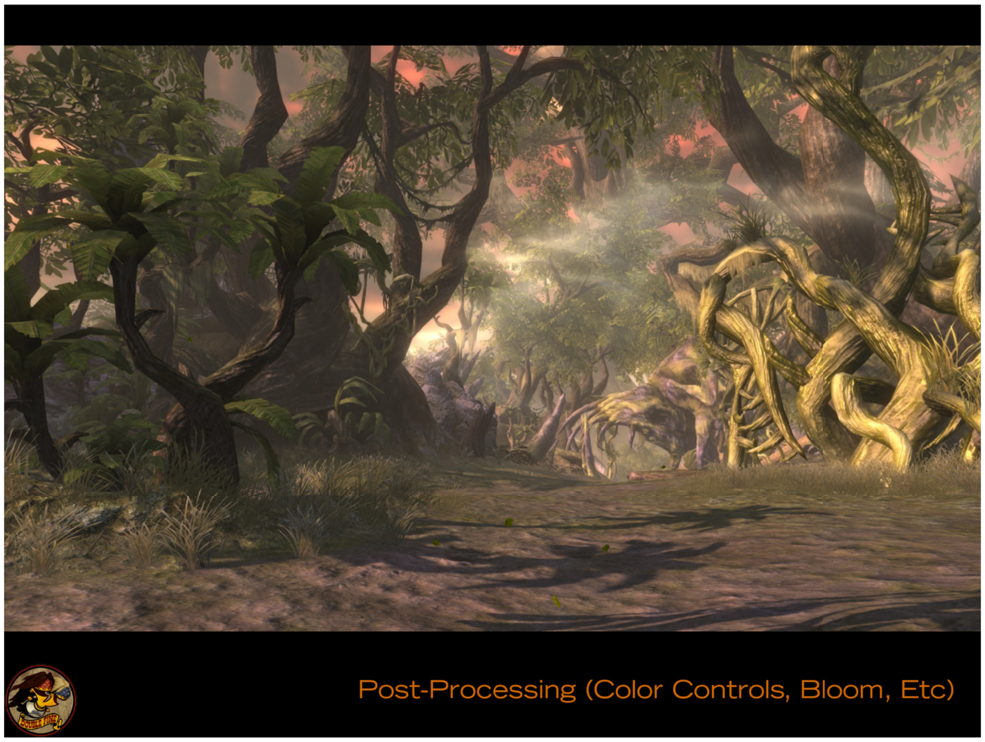


Changing the sun and sky lighting textures, tuning rim lighting and environment mapping. Larger contrast between sky and sun lighting helps to show this is a sunrise moment. Note the sky also responds to the lighting change.



Fog and Ambient FX

Fogging and ambient FX centered around the player help add depth and mood.



Post-Processing (Color Controls, Bloom, Etc)

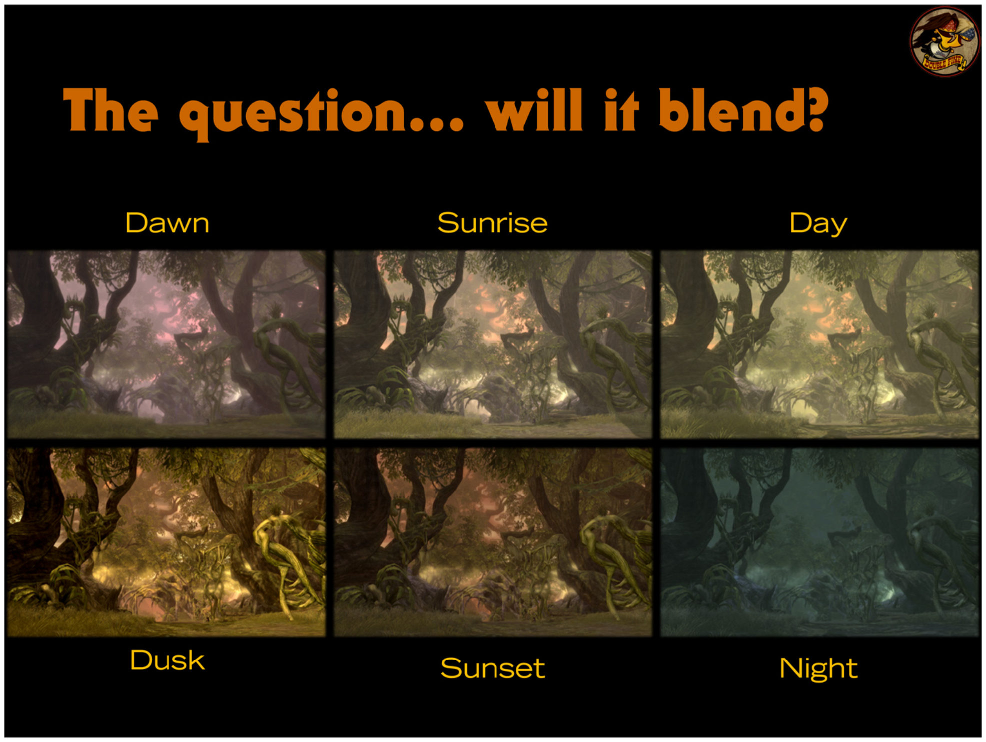
Post processing settings, like color correction and bloom, help glue the image together.

Integration with the World

- Climate sets all components of a moment
  - Each authored moment is a *Snapshot*
- A “Living world” is transitions between snapshots
  - Transitions occur between times of day, regions, weather variations, games states, cutscenes, etc…
  - Much more intuitive than keyframing!

Being able to craft these specific moments is great, but a “Living Brething Heavy-Metal World” isn’t just a collection of moments, but is more defined by the transition between all those moments.

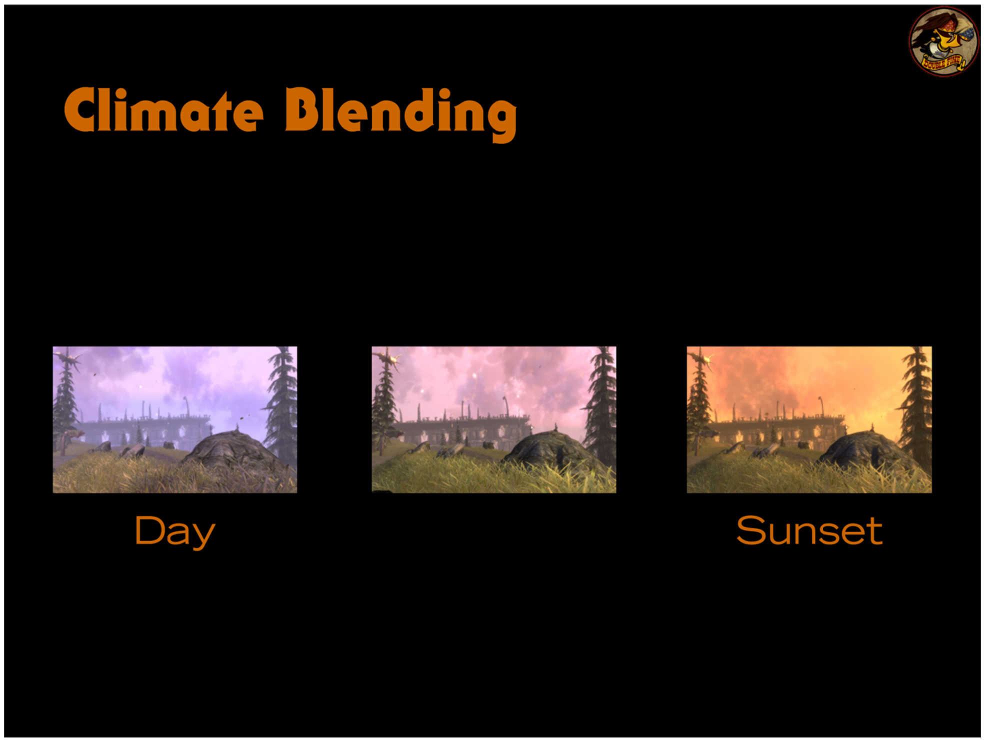
Note this does rely on your lighting system being amenable to making tons of modifications constantly without any popping (and in an inexpensive manner), etc… But that’s another talk!



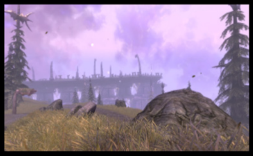
The question... will it blend?

Several authored snapshots for the jungle climate, during a clear day. Note that ie dawn on a clear may transition to sunrise on a cloudy/rainy day.

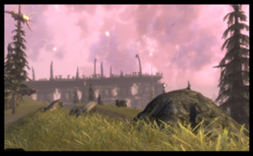
One of the many benefit from authoring using Snapshots is being sure to be able to precisely author specific moments. Although you have the freedom to add interpolation code specific to each property, artists will often add additional snapshots to smooth out unpleasant transitions. The snapshots shown above are certainly not uniformly spaced in time.

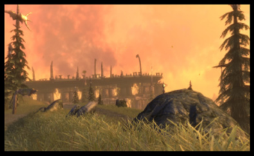


Climate Blending



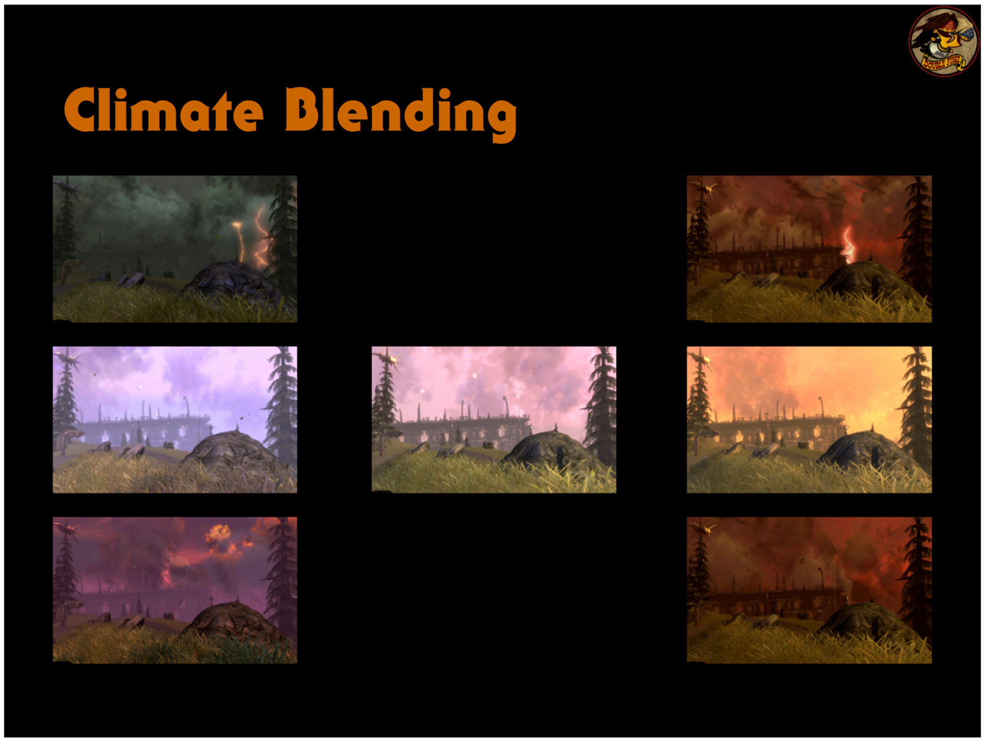
Day





Sunset

Once the climates to blend are picked, the blending is very much like an animation blend stack, where climates are pushed on/popped off, have specific blending weight (calculated differently based on how the climate was pushed) and can blend properties they set. Each property can have custom blending (ie, HSV color blends, blending textures for lighting/cubemaps/etc).

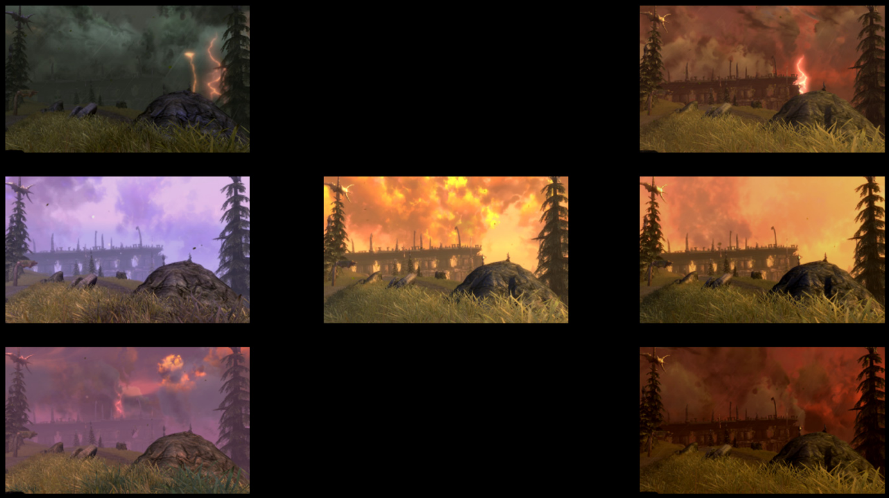


Climate Blending

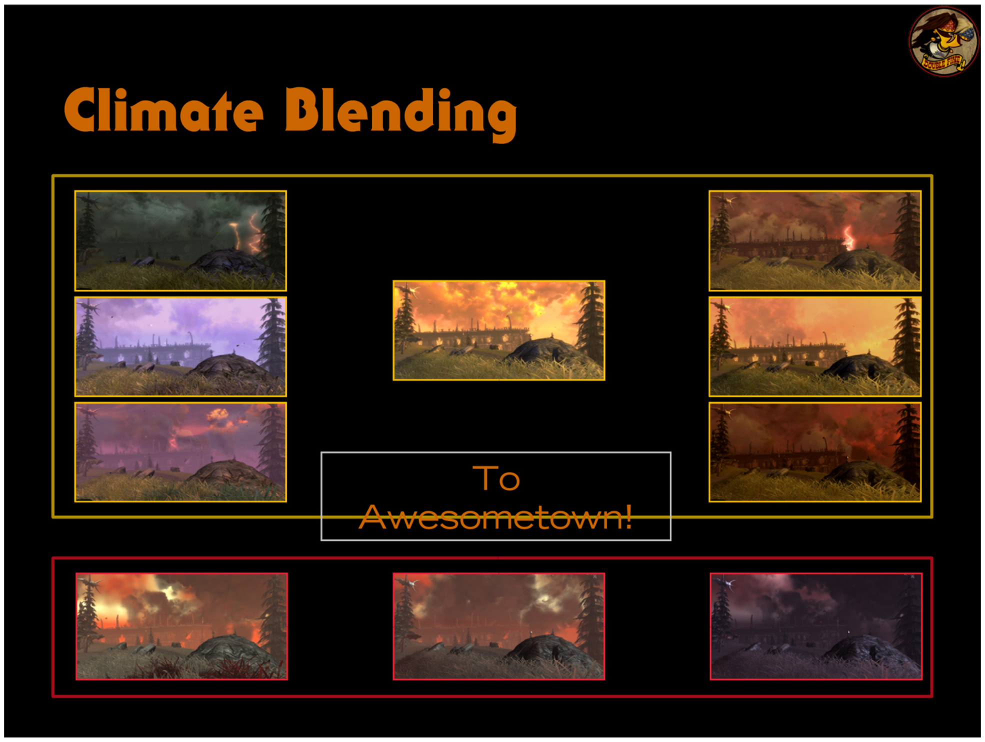
To pick the climates, the artists are actually authoring is a disguised a graph of all the possible climate settings, along with the edges that define the valid transitions. Obviously this includes Dawn to Sunrise, but also includes Dawn to every variation of weather at dawn, from all those variations to all the other Dawn variations in the neighboring region of the map, etc. But they don't need to deal with the details of authoring a multidimensional spline that would blend between all those states, at worst they need to add an additional climate in the right spot to smooth out blending.

The engine then picks which states to transition between based on times/position/etc, factoring in any authored random probabilities of all the climate variations at a particular time/weather setting, and pushes them onto the stack. The climate stack may also be modified explicitly by code/script/cutscenes/etc directly.

Climate Blending



The number of possible combinations is very large.



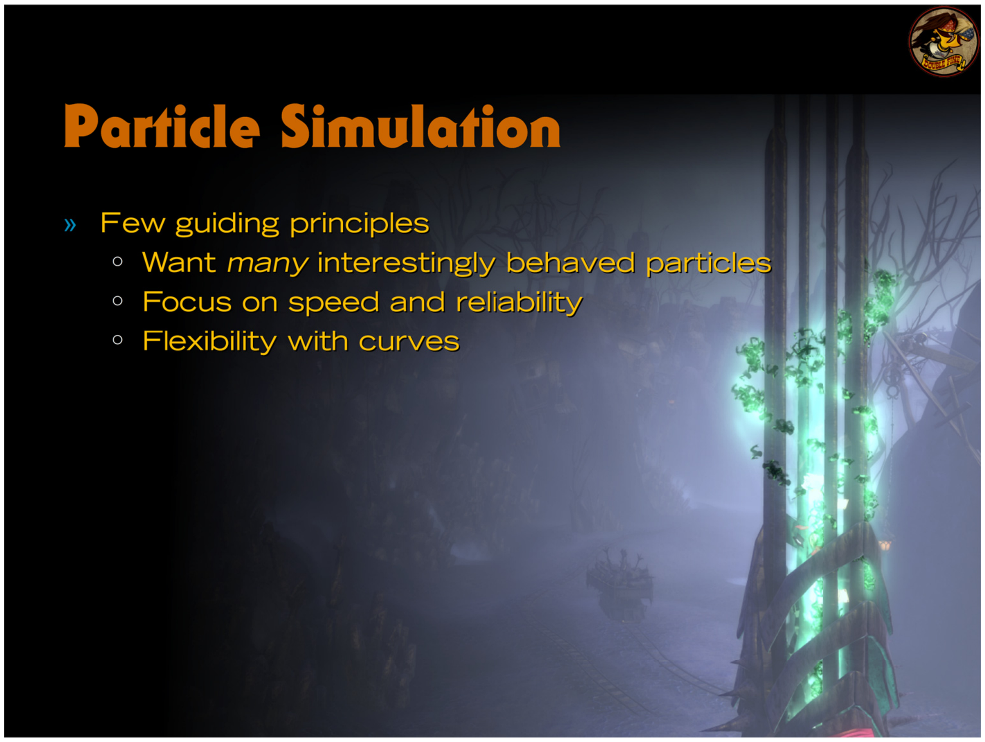
Climate Blending

To Awesometown!

The number of possible combinations is very large… And gets very hard to visualize. A great example of why we author it the way we do.

All this in motion does it more justice.

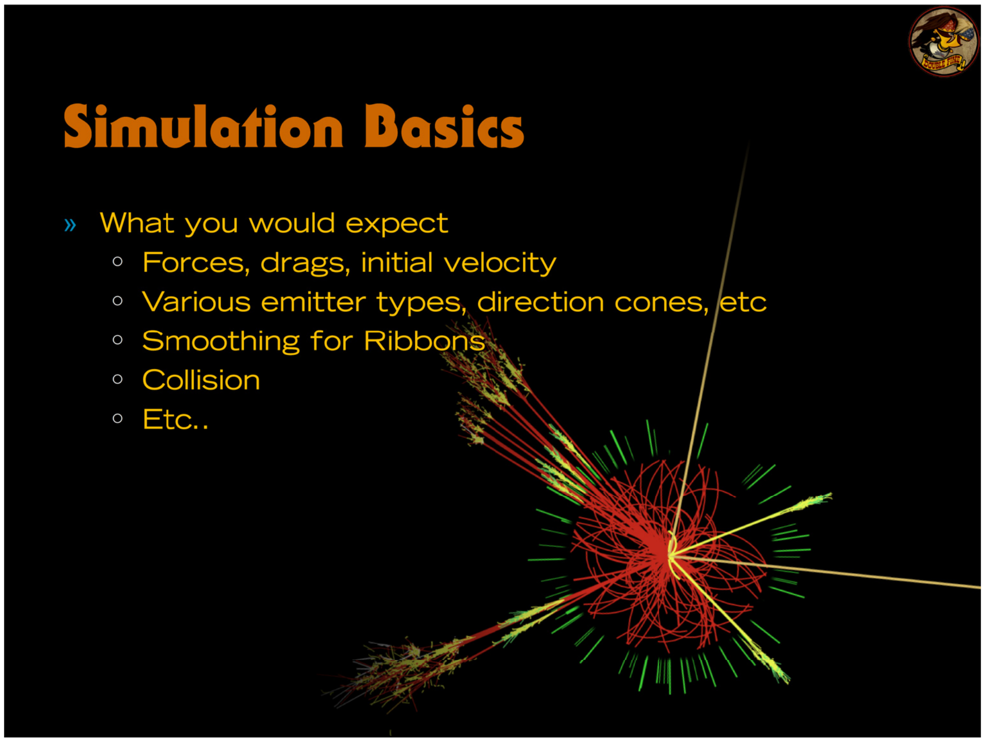
Particle Simulation



Particle Simulation

- Few guiding principles
  - Want *many* interestingly behaved particles
  - Focus on speed and reliability
  - Flexibility with curves

Drew is only happy when he increases the particle count or turns on more features.



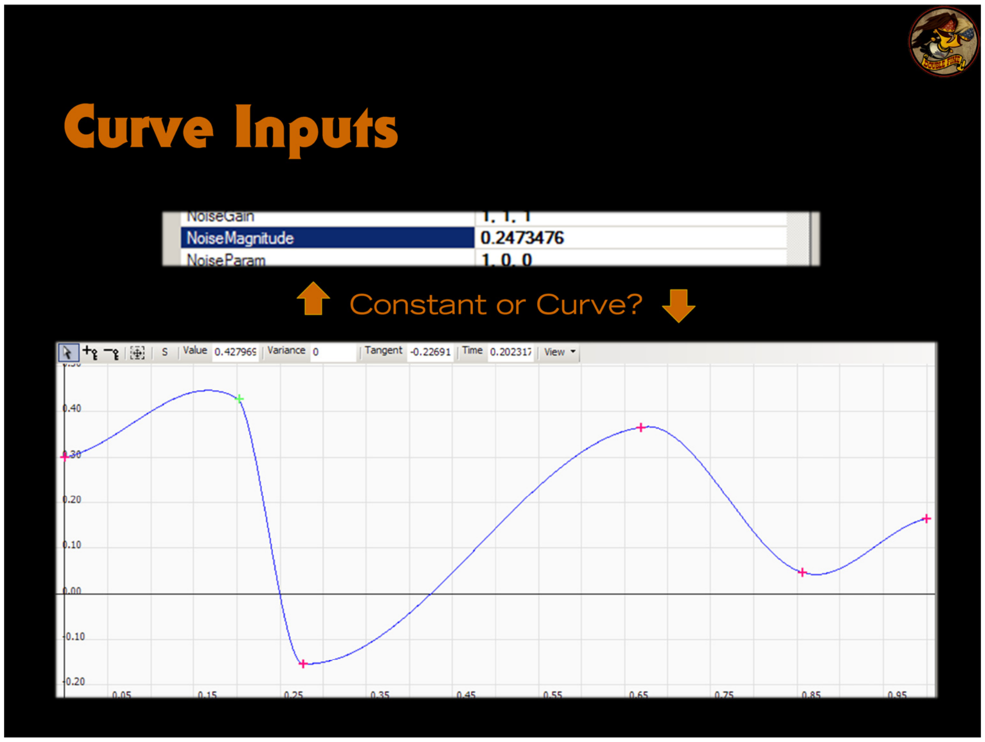
Simulation Basics

- What you would expect
  - Forces, drags, initial velocity
  - Various emitter types, direction cones, etc
  - Smoothing for Ribbons
  - Collision
  - Etc..

The most interesting bits are those we added to try to hit certain fx concepts, but ended up being much more generally useful than we imagined.

(Image: Simulated production of a Higgs boson in the ATLAS detector. *2008, Courtesy of CERN.*)

Curve Inputs



NoiseGain	1, 1, 1
NoiseMagnitude	0.2473476
NoiseParam	1, 0, 0

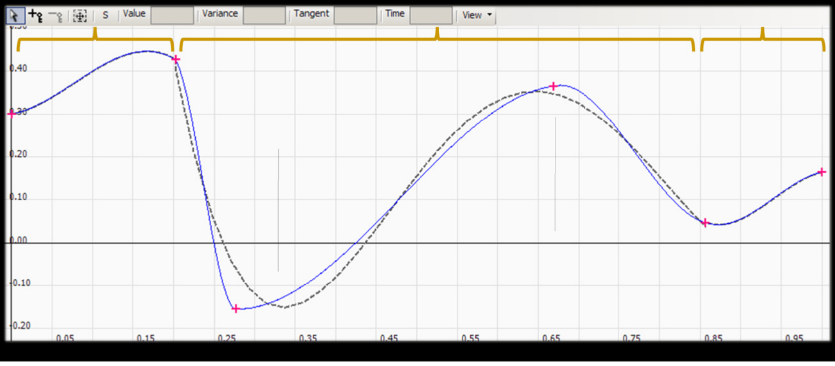
Constant or Curve?

Value 0.427965 Variance 0 Tangent -0.22691 Time 0.202317

Which would you prefer, a single scalar value to control a feature, or a spline so you can control the influence of the feature over the lifetime of the particle. We know what are artists prefer time and time again: the spline.

Curve Inputs

- Curve controls are more helpful than any feature
- Invest time to make them fast, low memory
  - We use 3 abutted 4-th degree polynomials
  - Inaccuracy is not an issue!



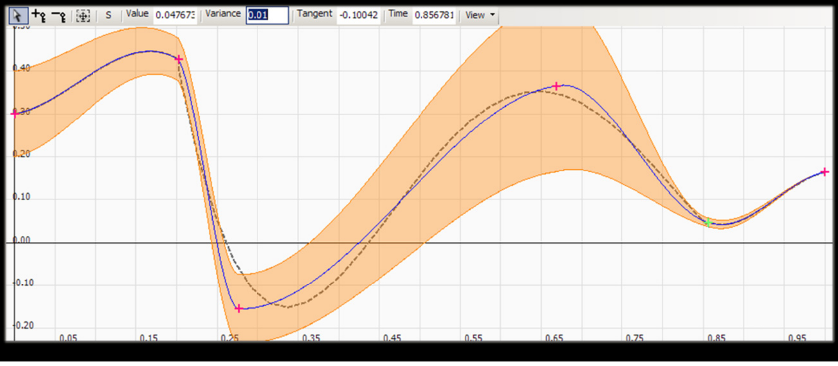
Investing in fast curve approximation and evaluation was key. Makes every single simulation feature immensely more useful. Once you have them, then most of your emitter, simulation and other parameters benefit greatly from ability to modify over the particle's or overall effect's lifetime.

For us, we least squares fit your polynomials to the spline authored by the artist, even brute force testing for the best locations to place the knots works interactively. Evaluating at runtime can be made quite cheap, most of your curves can share the same coefficients.

Dotted line in the picture is the fit curve. Annotated the image with the curve segments highlighted above. Yes, it is a little bit off – but it honestly doesn't matter. Show the fit curve and you're set.

Curve Inputs

- Variance
  - Additional method for per-particle variation
  - **value = curve + varianceCurve*(per particle -1..1)**



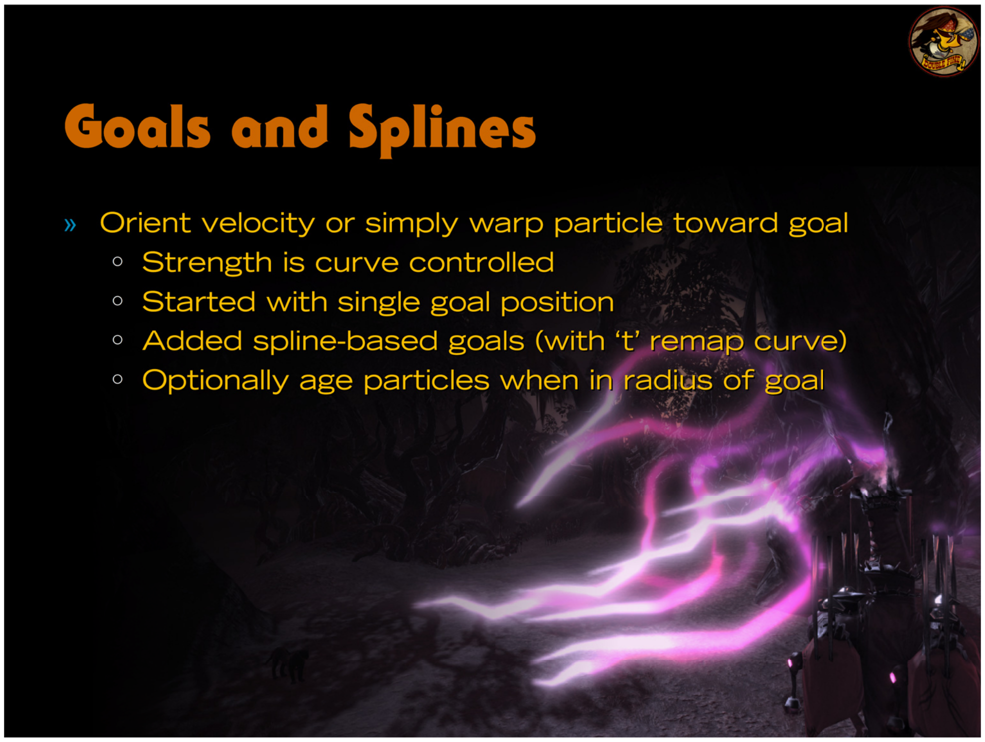
Value 0.047673 Variance 0.01 Tangent -0.10042 Time 0.856781 View

Adding an additional curve, which is the magnitude of the variance allowed from the main curve, is a great way to add per-particle variation in your simulation. A flat curve with high variance can simply add per-particle random values for your simulation or shader (ie, a curve at 0.5 with variance of magnitude 0.5 gives per-particle random numbers in the range of [0,1]).

We save some memory by storing the variance at lower precision than the main curve.

CLOSE UP
Converted Fans go out of top and over to other mouths at the base.
mouths chomp as if slowly eating
FAN DESIGNS
A glowing mist outline w/ black figures
B cloud-like forms
C ghostly figures w/ eerie trail
D abstract figures w/ different coloured foggy outline
E brightly lit figures w/ firey flames
F super misty foggy form w/ dark center
G detailed forms blurred in areas & misty in other areas
H brightly lit figures w/ blurs & slight mist
I twisted streaky mist/fog w/ dark figures
DOUBLE FINE

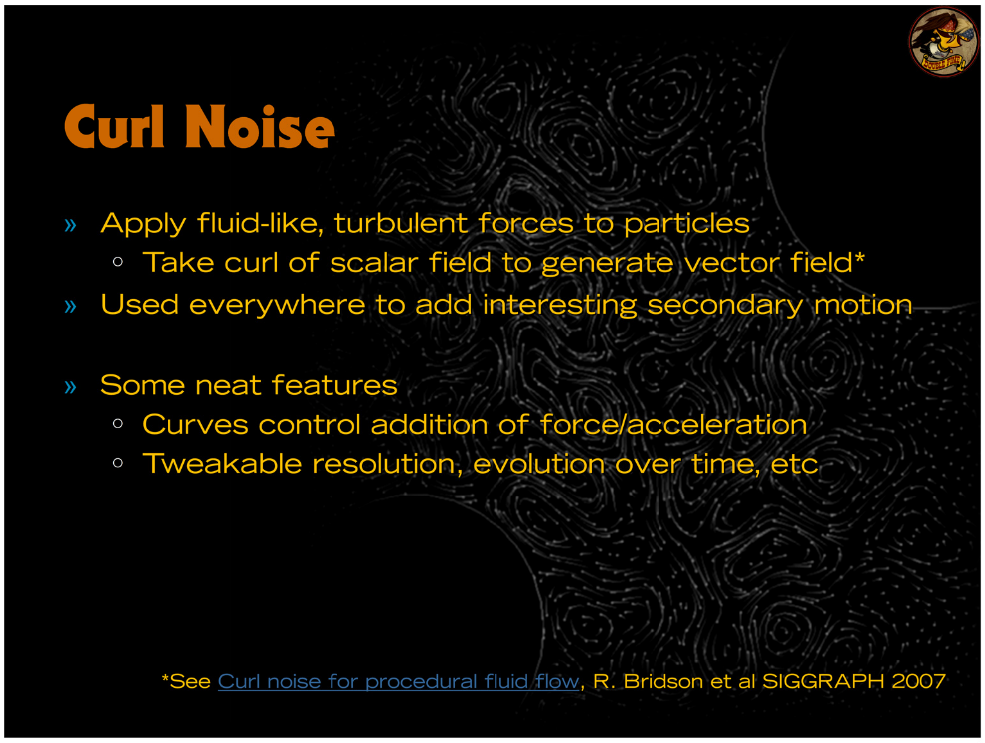
We'd have struggled to get these sorts of FX without adding any additional simulation code. It was just our hope that we could add things that wouldn't only only speak to these sorts of behaviors, but be more generally useful.



Goals and Splines

- Orient velocity or simply warp particle toward goal
  - Strength is curve controlled
  - Started with single goal position
  - Added spline-based goals (with ‘t’ remap curve)
  - Optionally age particles when in radius of goal

Goals – simply orienting velocity or warping position towards a goal position. Interesting bits are how you control the strengths over time (both positive and negative strengths are useful), how you determine what the goal position should be, and what you do once the particle gets there. Aging particles when they are in a radius is a really simple and efficient way to change the particle’s behavior once they are close to the goal.



Curl Noise

- Apply fluid-like, turbulent forces to particles
  - Take curl of scalar field to generate vector field*
- Used everywhere to add interesting secondary motion

- Some neat features
  - Curves control addition of force/acceleration
  - Tweakable resolution, evolution over time, etc

*See Curl noise for procedural fluid flow, R. Bridson et al SIGGRAPH 2007

This process is of course by no means unique to fx, in all facets of video games the risk of implementing a feature is much less if you KNOW it's going to be awesome and what the fallout from it could be. Yet another reason why encouraging this type of low impact experimentation can yield goodness.

Curl noise for procedural fluid flow, R. Bridson, J. Hourihan, and M. Nordenstam, Proc. ACM SIGGRAPH 2007.

Drew's Favorite Feature Ever

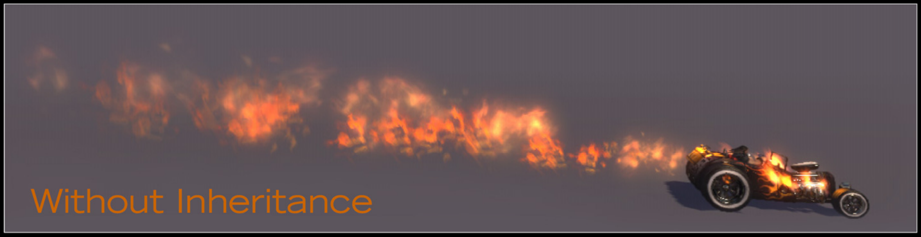
Example video of type of motion you can achieve. So many variations can be given by altering the resolution (tile rate) of the noise field, how it evolves over time, how much particle is affected by the field, etc.

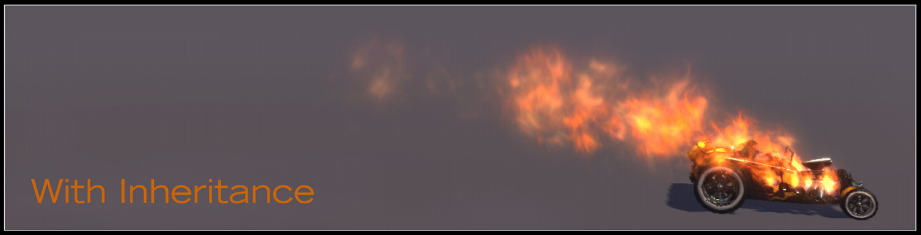
A giant plume of smoke rising into the sky would generally use a low tile rate and low evolution speed.

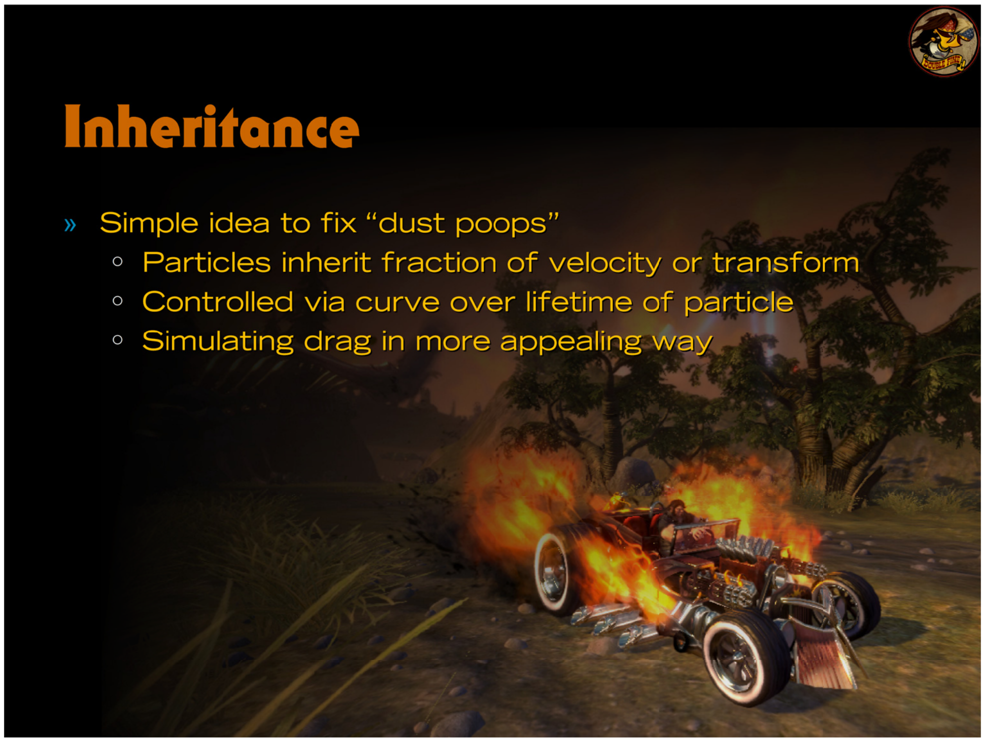
Fast moving leaves in a storm would have a medium tile rate with a faster evolution speed.

Sparks rising from a hot, turbulent fire would have high tile and evolution speed (coupled with a lifting force).

Transform/Velocity Inheritance







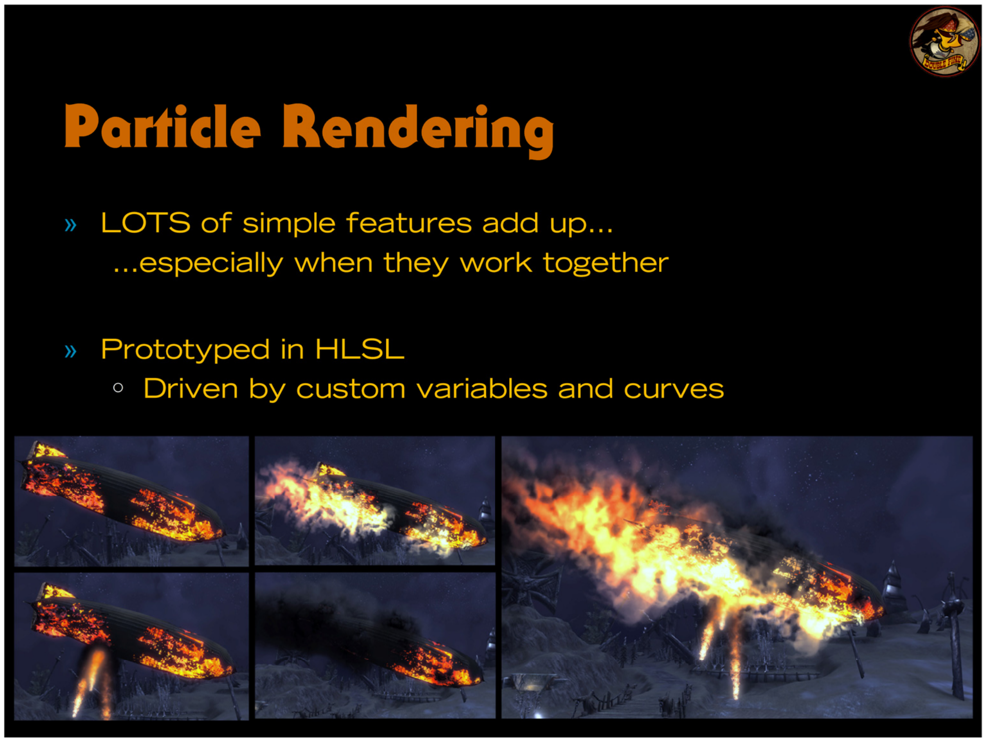
Inheritance

- Simple idea to fix “dust poops”
  - Particles inherit fraction of velocity or transform
  - Controlled via curve over lifetime of particle
  - Simulating drag in more appealing way

Very useful when you have a system attached to a moving object, but don’t want to create additional particles, or want to make them feel like they are slowing getting caught in the wind and being accellerated. Almost all all dust/fire/etc systems have this enabled, unless they are static.

Driving it negative helps the system be pushed away from object when it is moving. Ie, can help fire feel more “vortexy” and hot.

Particle Rendering



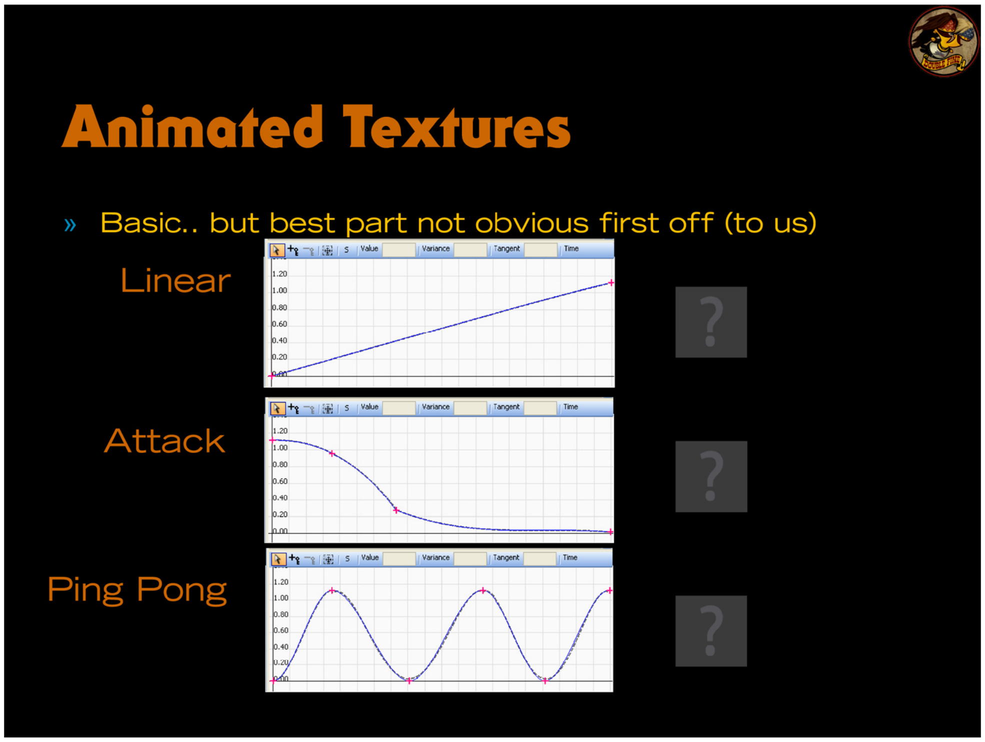
Particle Rendering

- LOTS of simple features add up…
  …especially when they work together
- Prototyped in HLSL
  - Driven by custom variables and curves

The general idea is that we make it easy for quick iteration and experimentation, but essentially limit how much can go on at once by only providing a small set of free variables to use when prototyping new features. Once the limit of these curves, shader inputs, etc are used, to add additional features some must be either promoted to officially supported, or discarded.

We try to make the promotion easy (scripts to rename parameters, organized shader code, etc), but be sure to discard them if they don't seem like they'll would work well with others/the end effect is possible to achieve through other means.

Patience and some diligence is needed, but we've found doing things in this way truly helps keep things organized and lets you be both flexible and able to experiment, while keeping the number of un-optimized/not fully supported features to a minimum.



Animated Textures

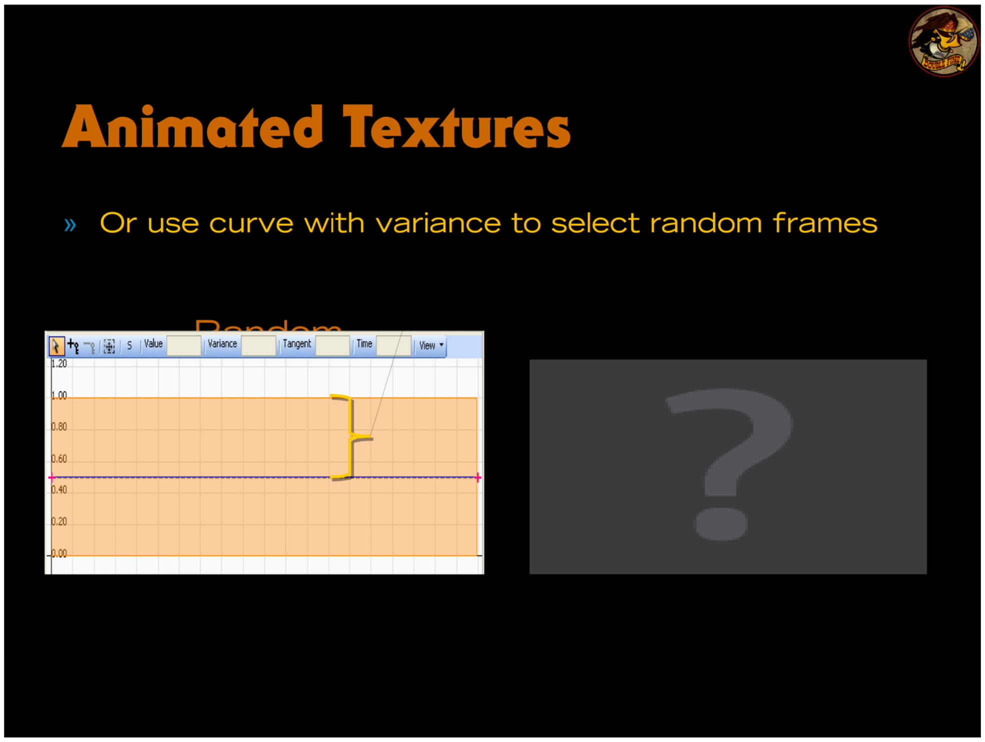
» Basic.. but best part not obvious first off (to us)

Linear

Attack

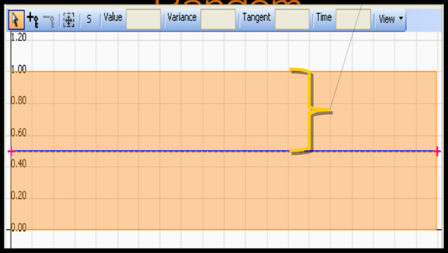
Ping Pong

Using a curve to determine which frame to use from your filmstrip makes animated textures much more useful than a simple ramp.



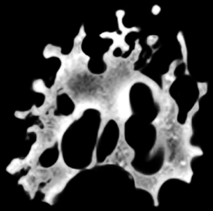
Animated Textures

» Or use curve with variance to select random frames



An aside – we used to spend a good amount of time building textures from AfterFX/etc output frames. As our resource system has the ability to register any sort of file extension to be converted into a resource, we added a way for ".filmstrip" files (containing a list of texture names and final output texture parameters) to be converted into an atlas automatically without needing to do it by hand. Few hours of programming time saved tons of time in the long run.

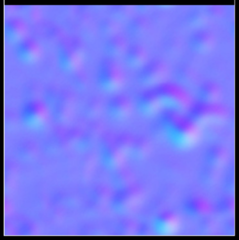
Particle Texture

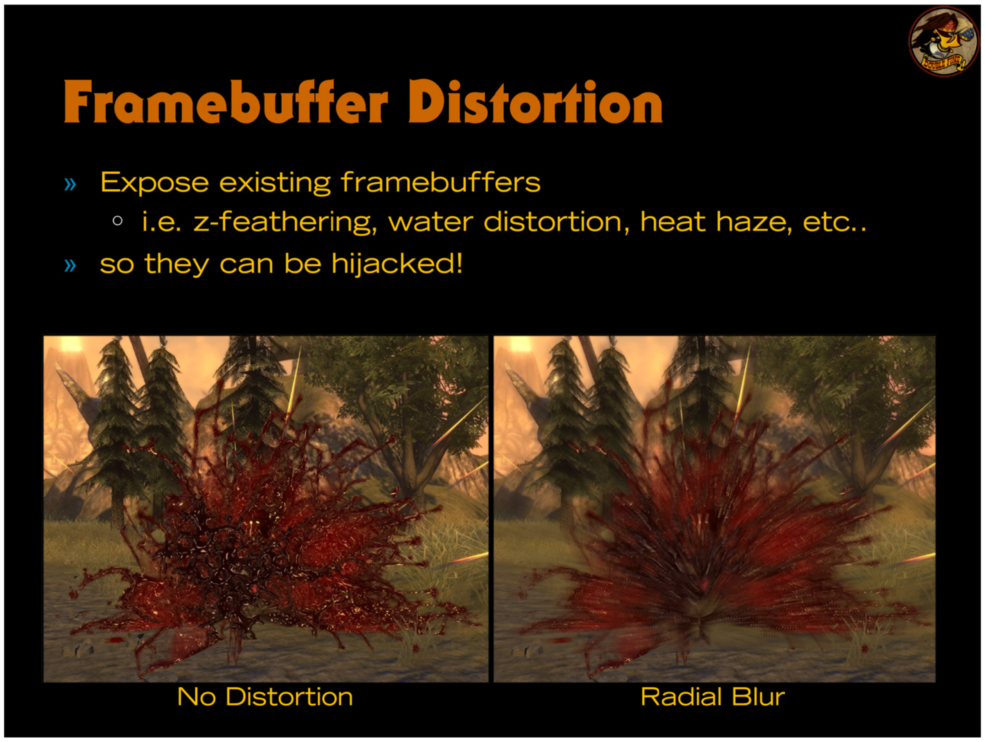


placement

s on particle quads
n-space or particle-space
on accumulates and/or scrolls

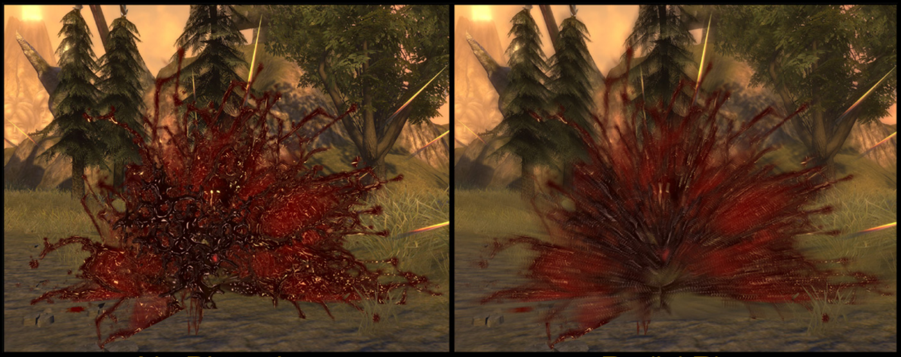
Distortion Texture





Framebuffer Distortion

- Expose existing framebuffers
  - i.e. z-feathering, water distortion, heat haze, etc..
- so they can be hijacked!

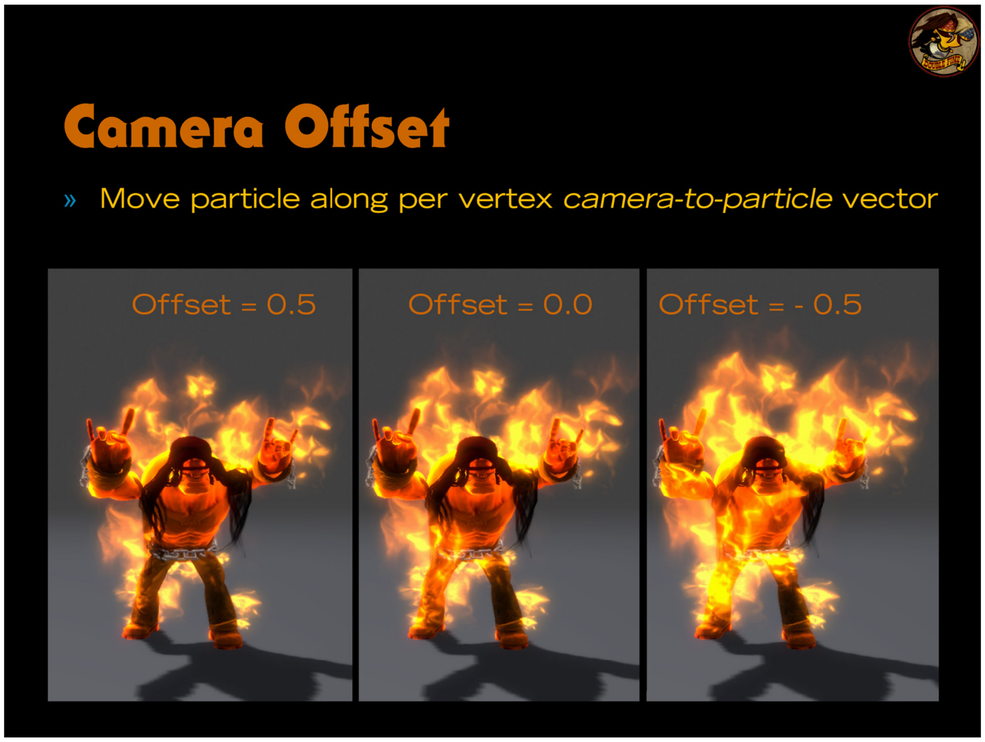


No Distortion

Radial Blur

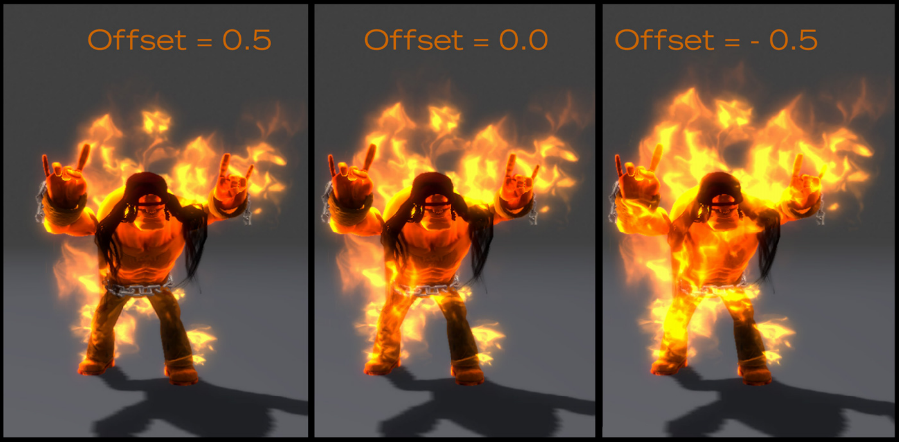
Radial and directional blurs sample along a direction based on uvs or other parameters. Distortion adds based on additional texture.

In most cases, additional curves are used to control strength over time, width of blur, etc.



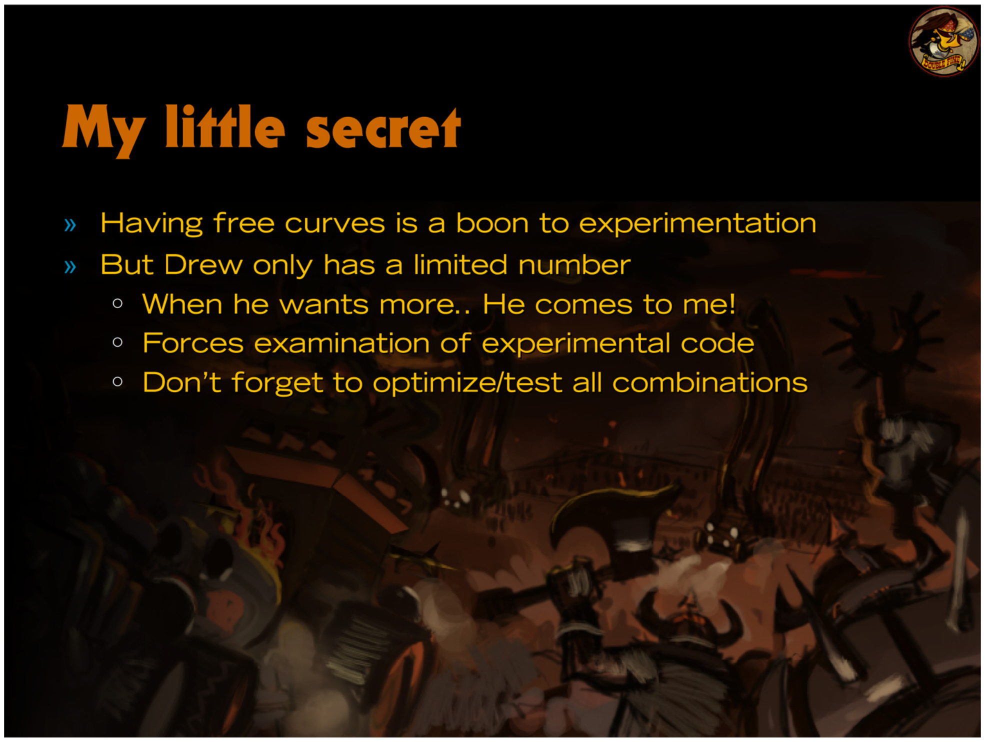
Camera Offset

» Move particle along per vertex *camera-to-particle* vector



Super cheap, removes need to add more particles, often removes need to enable z-feathering/soft particles.

Just beware that large values will subvert any cpu-side particle size culling that would normally prevent massive particles from eating too much fill.



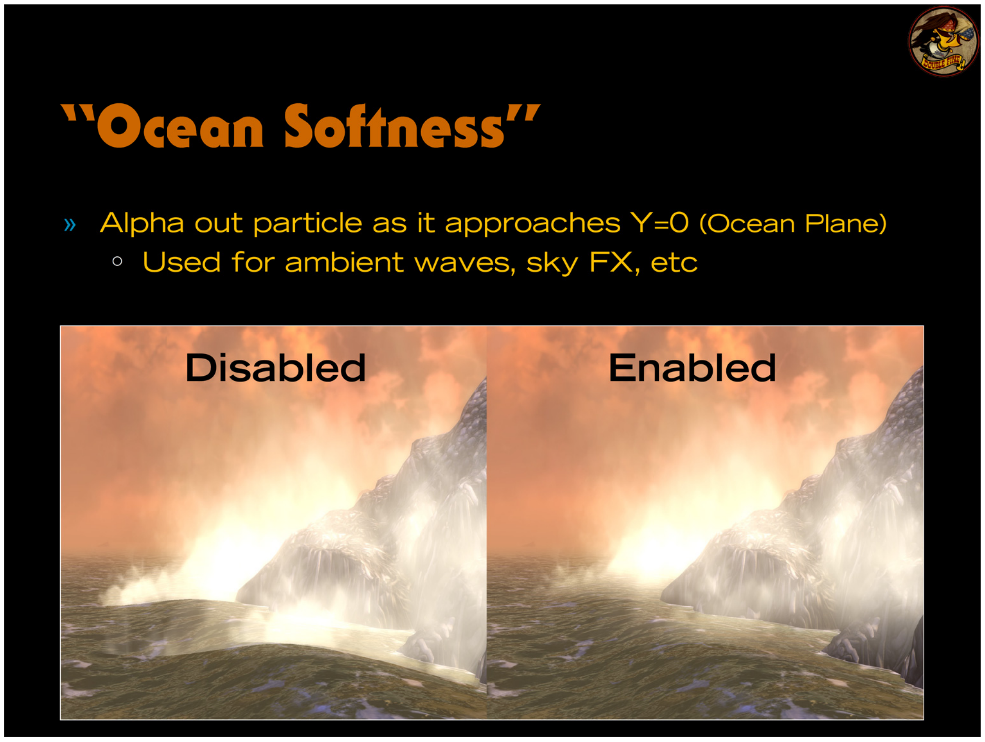
My little secret

- Having free curves is a boon to experimentation
- But Drew only has a limited number
  - When he wants more.. He comes to me!
  - Forces examination of experimental code
  - Don't forget to optimize/test all combinations

This type of workflow forces you to periodically examine and be sure to optimize and well integrate your features with the others, while keeping it easy to experiment and freely play with new features (without needing to modify the "main" particle shader).

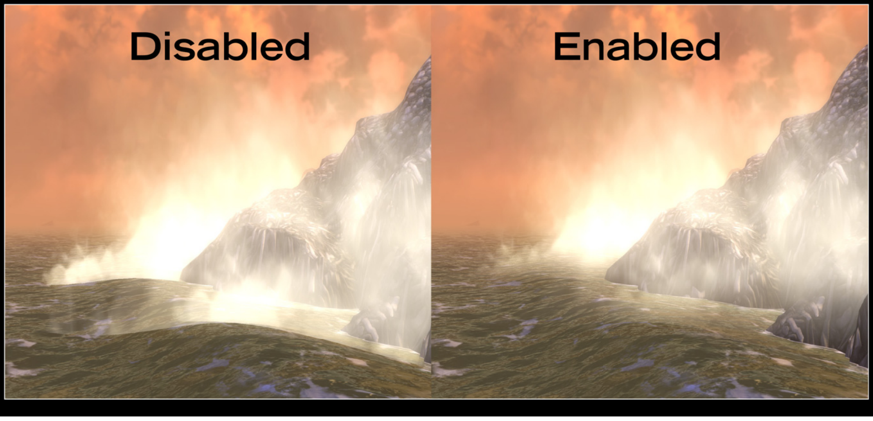
At times, we'd even choose not to integrate a cool feature since it would be too difficult to integrate with others, or we found it could be equally well emulated with a combination of previously exposed parameters. Not having this excess of bells and whistles helps vfx artists work quickly by reducing the overall complexity of the systems they need to work with.

It's worth saying that the few features we didn't roll in like this (due to time constraints) are by far the hardest rendering features to use!



"Ocean Softness"

» Alpha out particle as it approaches Y=0 (Ocean Plane)
  - Used for ambient waves, sky FX, etc



A useful feature in its own right, but mostly interesting as an example of the type of optimizations you can get by officially supporting a feature once it is useful! Most of the computation moved onto the cpu in this case.

“Ocean Softness”

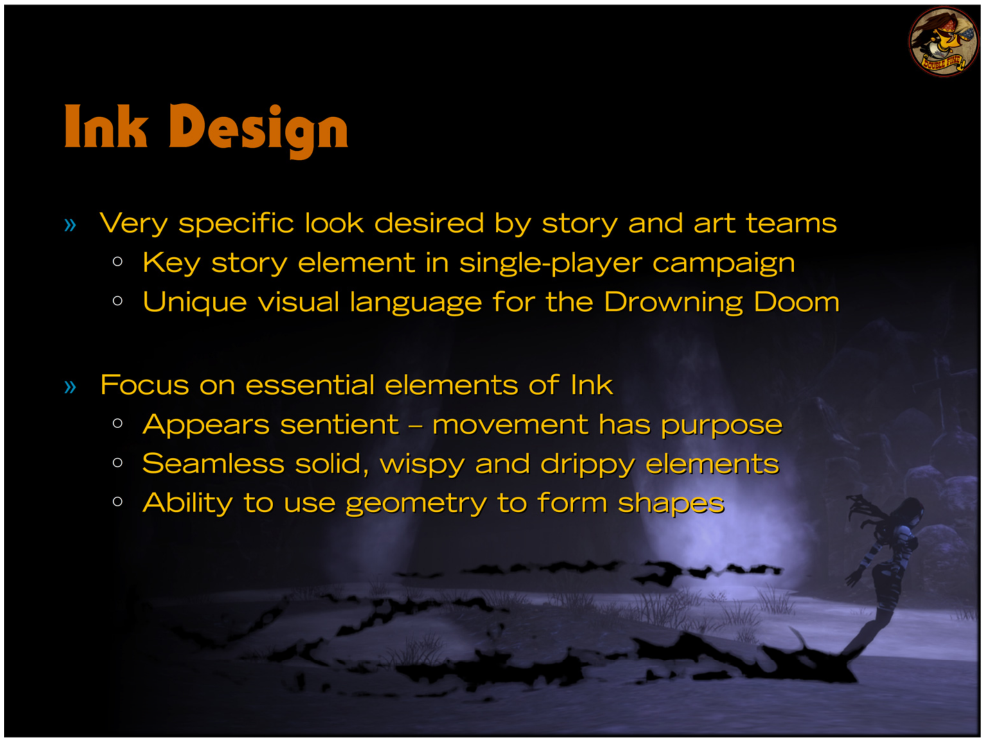
- Prototype was many cycles + dynamic branch
- After merging into mainline: 1 scalar madd

```
half
CalcParticleOceanSoftness_drew(
    float fWorldSpaceY)
{
    float fOceanSoftness = 1.f;
    if (g_vUserVec.w > 0.0f)
    {
        float fOceanHeight = 10.f;
        fOceanSoftness = saturate((fWorldSpaceY - fOceanHeight) * g_vUserVec.w);
    }
    return fOceanSoftness;
}
```

```
half
CalcParticleOceanSoftness_pete(
    float fCamWorldSpaceY)
{
    return saturate(fCamWorldSpaceY * g_vUserVec.w + g_fParticleCamScaledOceanHeight);
}
```

Building up a library of various rendering tricks really allowed us to tackle one of the more difficult fx challenges for BL.

Ink

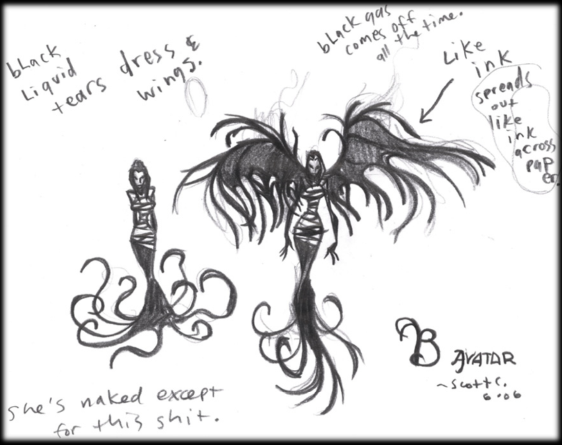


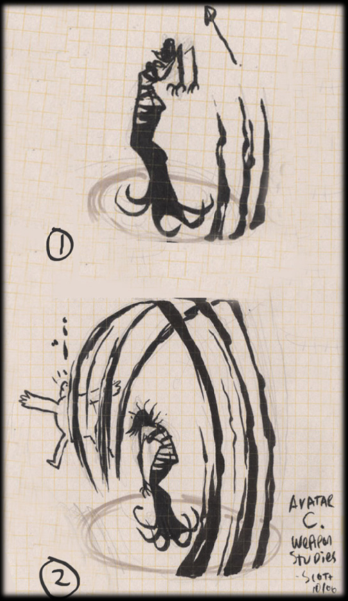
Ink Design

- Very specific look desired by story and art teams
  - Key story element in single-player campaign
  - Unique visual language for the Drowning Doom
- Focus on essential elements of Ink
  - Appears sentient – movement has purpose
  - Seamless solid, wispy and drippy elements
  - Ability to use geometry to form shapes

Ink was tough, we'd made several attempts and never really had it stick – and we were unable to compromise due to the importance to BL as a whole.

We knew the basic elements we needed to tackle, try approaching each independently, using tools we've developed up until now. Keep mind on places we'd fallen in prior approaches.


black liquid tears
dress & wings.
black gas comes off all the time.
Like ink
spreads out like ink across pap
she's naked except for this shit.
AVATAR
~Scott C. 6·06

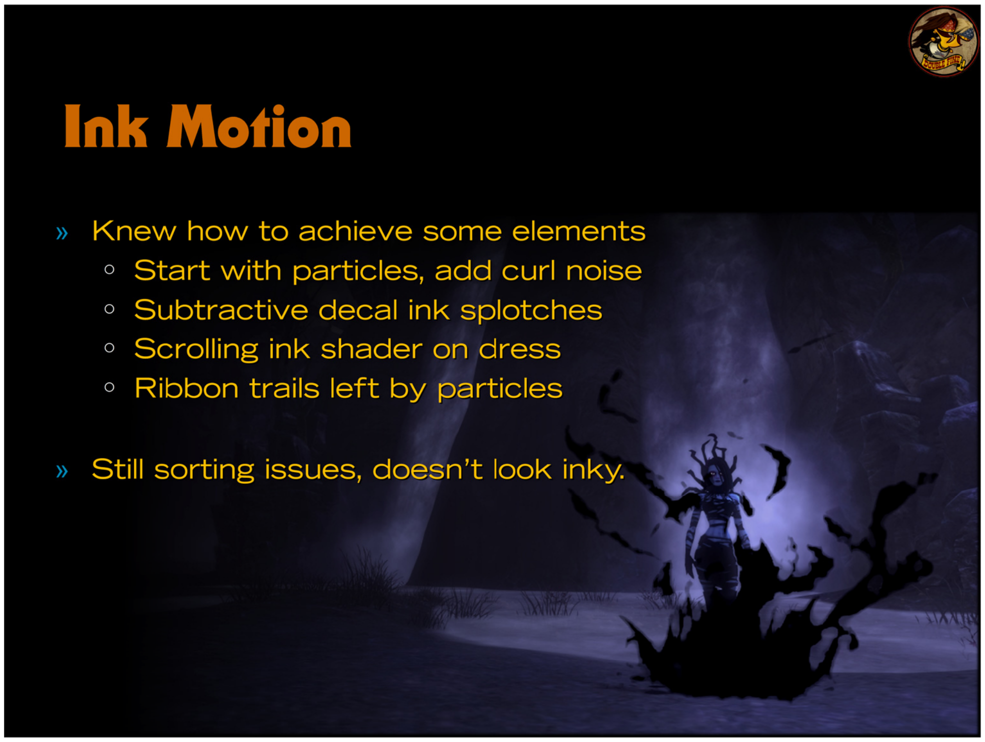


Ink Concept Sketches

Many of the concepts for ink as a weapon showcase the ink emerging, forming a particular shape, and then falling/dissolving. All the while the edges should remain "drippy/dissolvey".

At idle, the ink should feel ready to move at any moment, and be forming interesting, non repeating shapes and patterns.

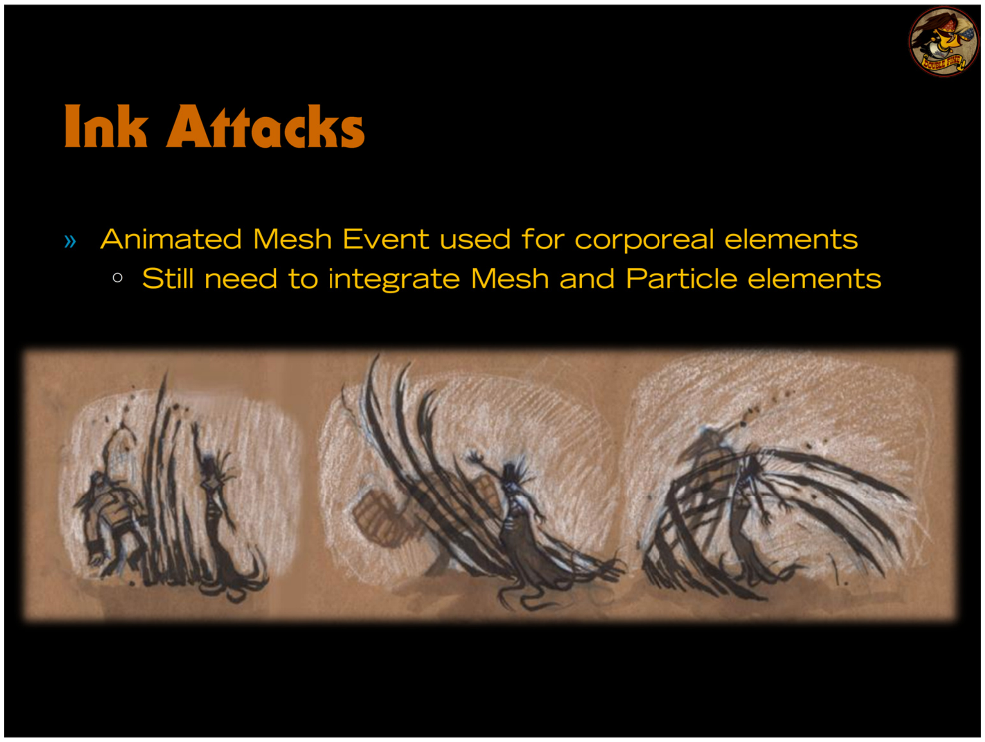
The concepts were purposefully not very detailed – since we had a good idea of how to describe the ink, but not sure how to exactly visualize it, we didn't request concept art that might be terrifically difficult to hit but rather worked to develop something that had all the qualities we knew ink would need.



Ink Motion

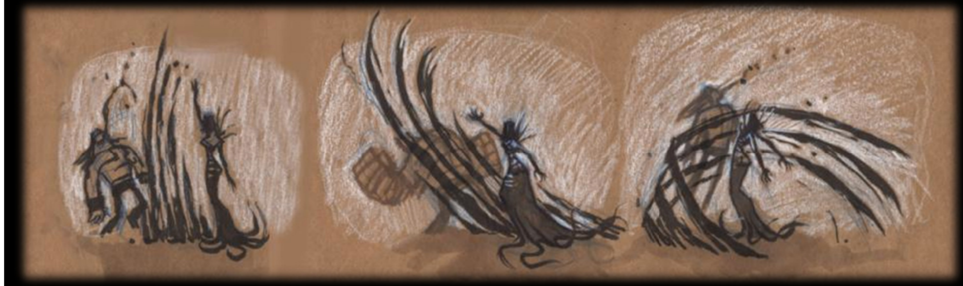
- Knew how to achieve some elements
  - Start with particles, add curl noise
  - Subtractive decal ink splotches
  - Scrolling ink shader on dress
  - Ribbon trails left by particles
- Still sorting issues, doesn't look inky.

Start with fairly solid particle simulation feature set, go after basic motion first.



Ink Attacks

- Animated Mesh Event used for corporeal elements
    - Still need to integrate Mesh and Particle elements

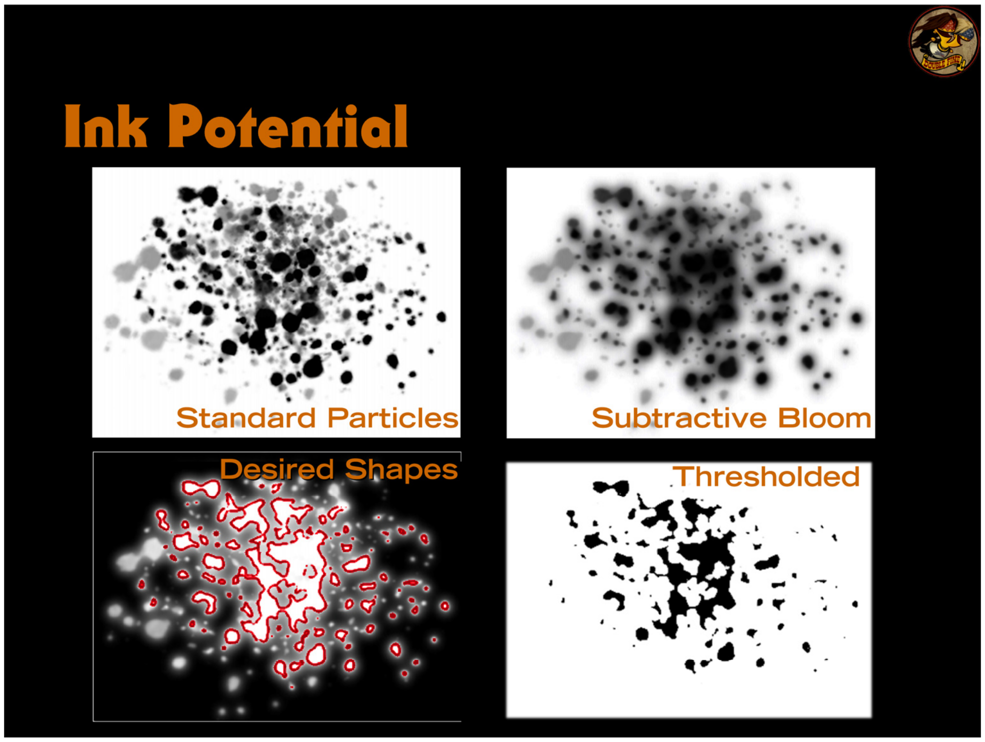


Last part is how to make sure that geo and particles mix – and it's easy: render ink-shape geometry to the ink buffer as well.

Additional particles emitted from surface to leave droplets/ribbons behind.

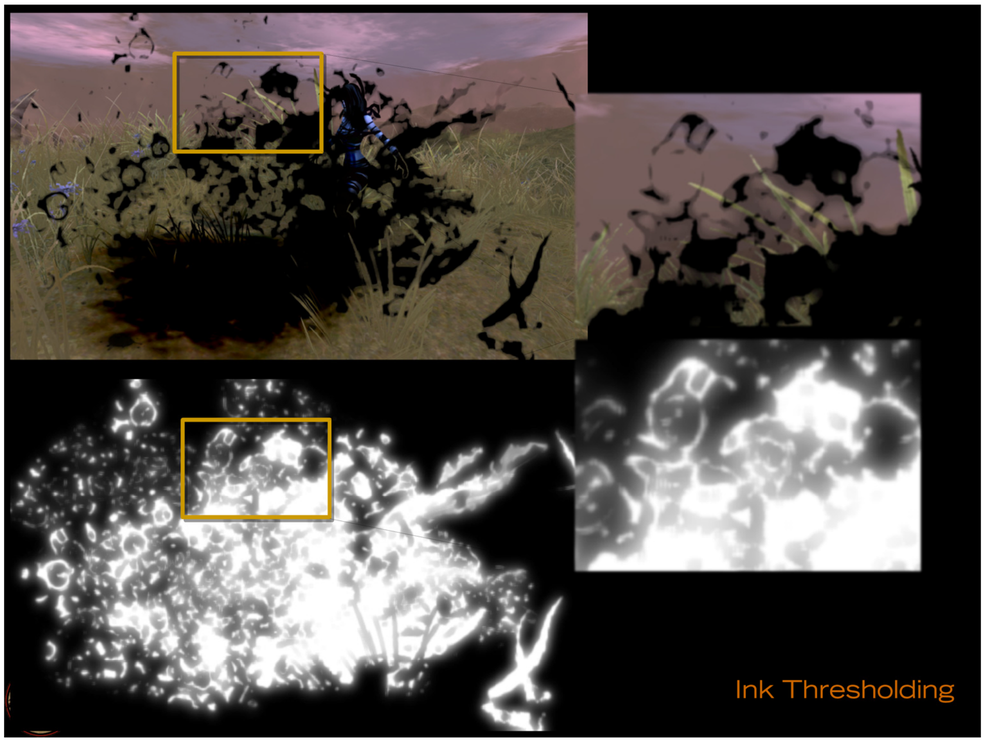
Ink Rendering

- Need to fuse geometry and particles
  - Silhouette most important: matte surface ok
  - “Fluid-y” shapes break up over time
- Lead to approaching problem in screen-space



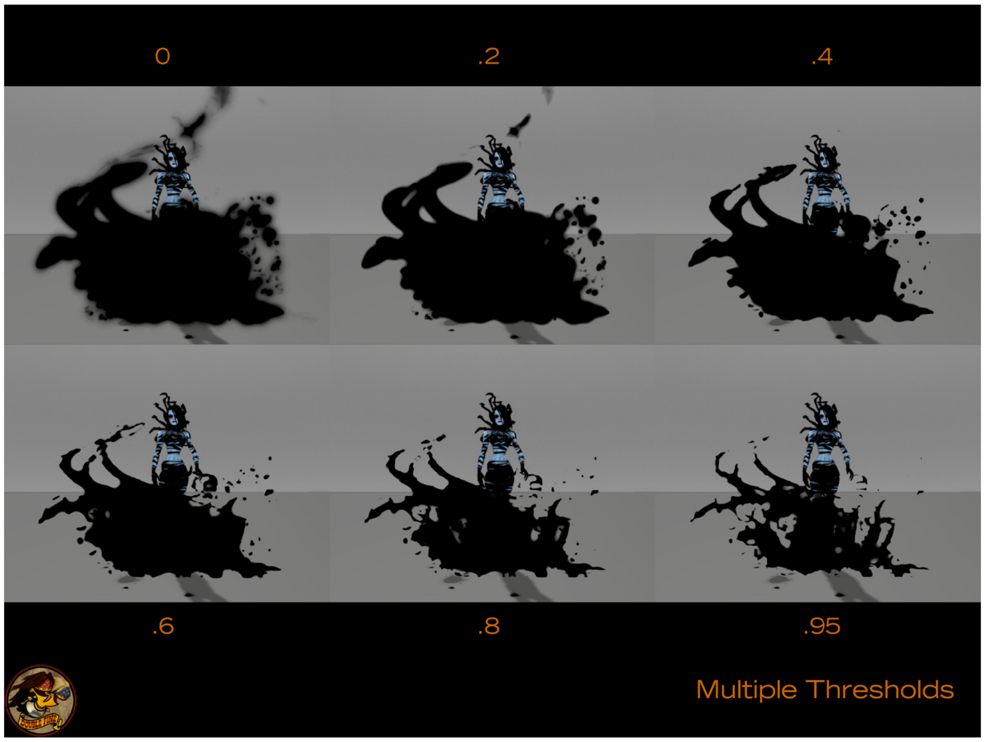
Ink Potential

What we're really authoring here is a potential field, which we then threshold to extract various shapes from. Having layers of particles all overlapping, where shapes are formed via the summation of several particles, truly gives a fluid/surface-tension like feel to the final result.

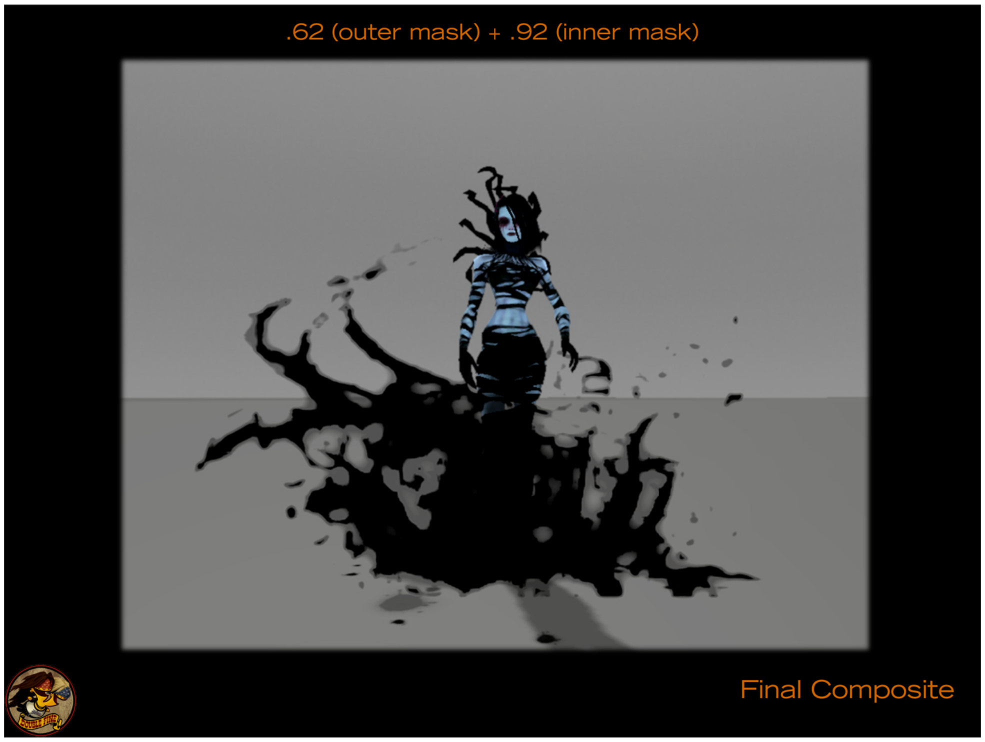


The overlapping layers of noisy textures gives interesting shapes in the final image, and in motion these shapes merge and detach as the underlying textures interfere with each other's texture.

UV displacement on the underlying textures also works *really* well! More later.

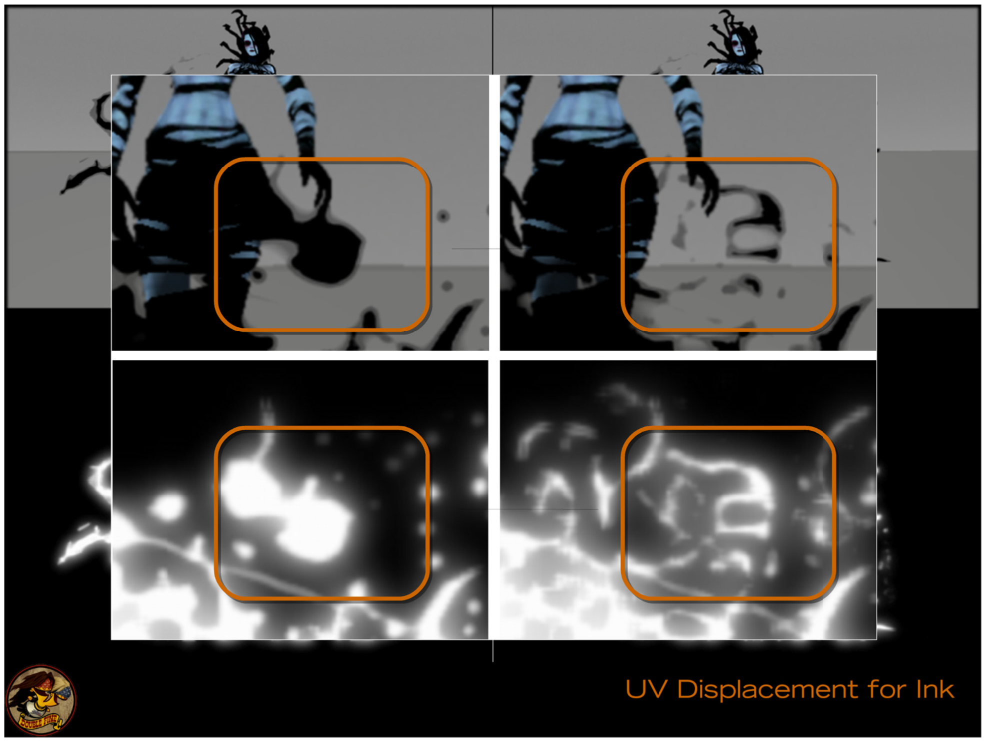


Various thresholding levels.



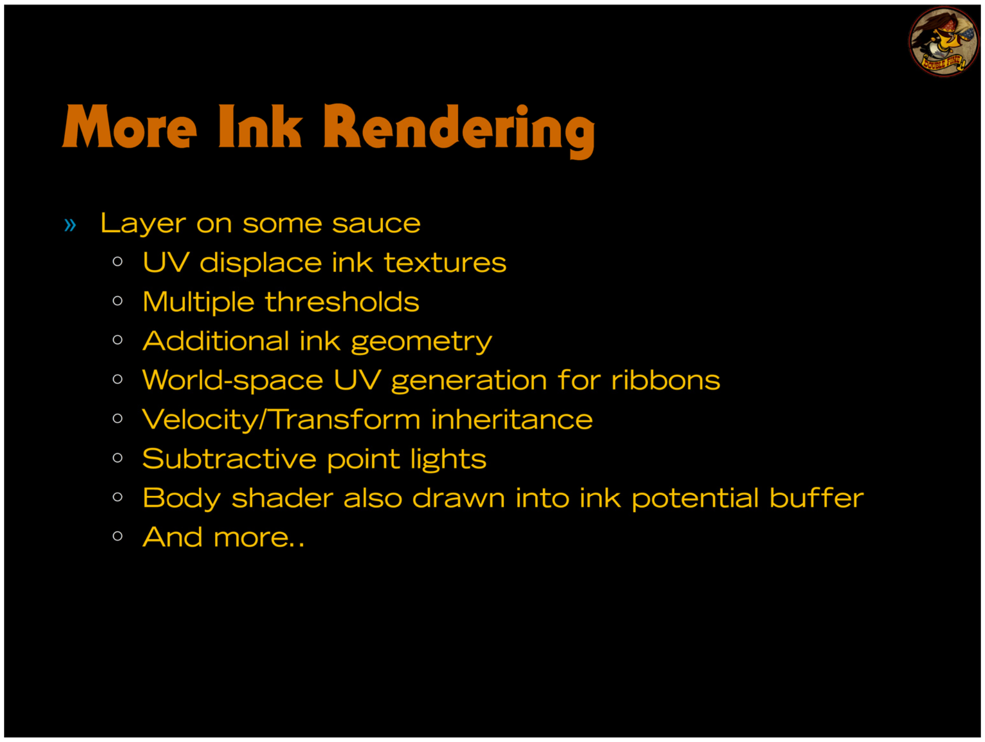
Final image is a composited version of two thresholds with different strengths for each, but flat influence across the body.

Different colors can be given to each layer to give different looks (or gradients across the layer body), but flat works best for ink.



UV displacement – can really push it, and since you don’t directly see the results, it just adds lots of interesting noise in the final result.

This way we have inter-particle fluid-like interactions from the potential field, and intra-particle fluid like behavior from the uv disp.



More Ink Rendering

- Layer on some sauce
  - UV displace ink textures
  - Multiple thresholds
  - Additional ink geometry
  - World-space UV generation for ribbons
  - Velocity/Transform inheritance
  - Subtractive point lights
  - Body shader also drawn into ink potential buffer
  - And more..

Lots of additional elements end up contributing to the final look (some of which we have already talked about). It’s quite nice that all the techniques interact predictably.

Acknowledgements

Questions?

Pete Demoreuille
pbd@pod6.org

Drew Skillman
drew@drewskillman.com

Pete accidentally went to the end of the presentation again, didn't he.