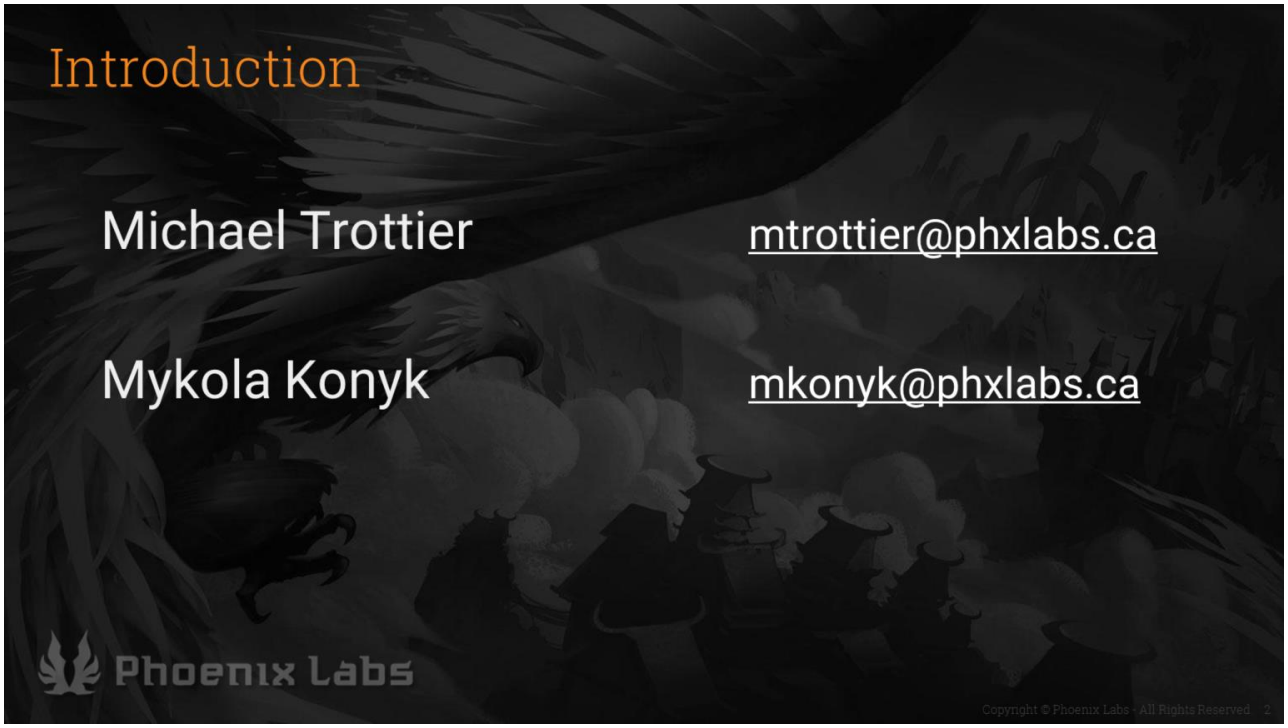

Phoenix Labs
Procedural Islands of 'Dauntless'
Technical Artist Bootcamp
Michael Trottier, Senior Technical Artist
Mykola Konyk, Senior Software Engineer



- Michael bio:
  - 15 years at Bioware prior to Phoenix Labs
  - Artist on KotOR and Environment lead on Mass Effect 1 - 3
  - Technical Art Director on the early stages of what is now Anthem
- Mykola bio:
  - Worked at SideFX on Houdini + Engine
  - Previously worked rendering engineer on a broad range of

titles.

- The system and content has been developed primarily by a small group at Phoenix Labs. Michael Trottier(tech art), Mykola Konyk(engineer), and Jen Morgan(world art), Cory Lake(technical art), Glenn Barnes(Art direction)

Overview: what we're talking about / what the viewers can walk away with. (fill in outline slide)

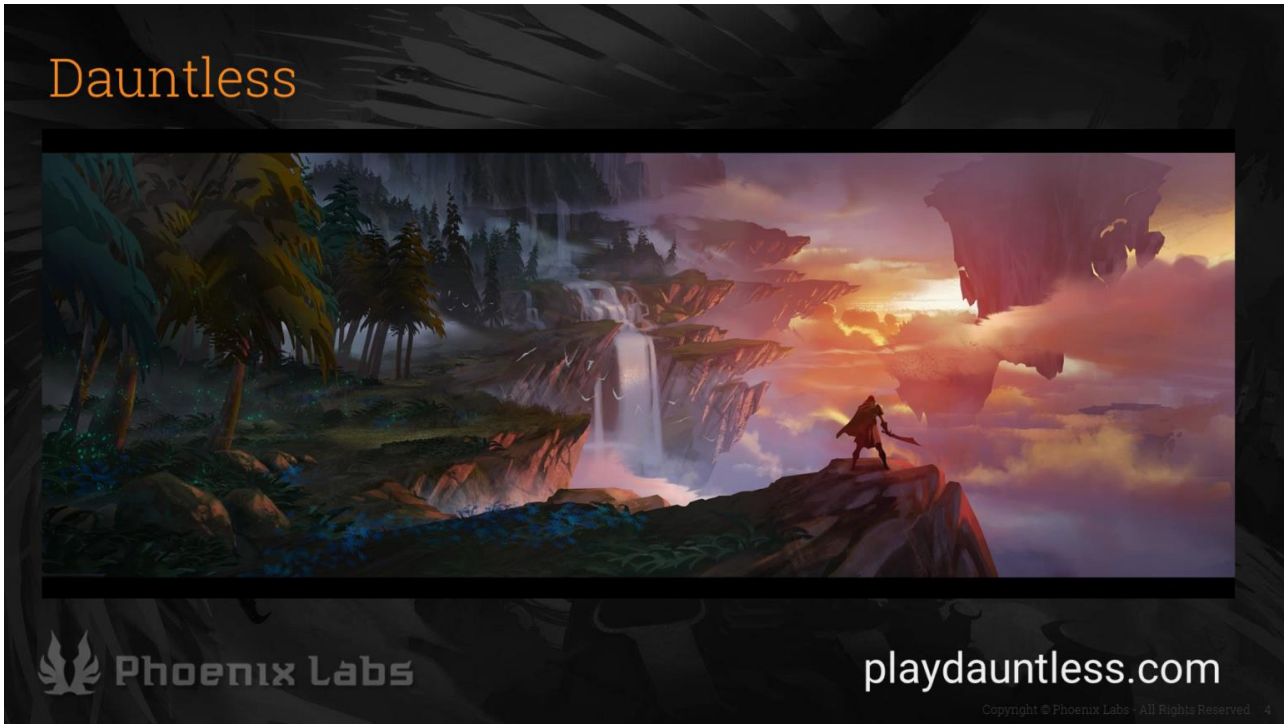
- Introduction - say hi
- Our team and how proceduralism makes sense for all the usual reasons (more with less)
- Dauntless
  - the game (quick summary + invite to upcoming open beta)
  - The setting - the shattered isles

(some quick inspirational concepts)

- Goals/Requirements + terminology slide?
  - give a clear understanding of our target
  - Show current status
- Houdini basics (networks, attributes, assets, vex + hdk, houdini engine). Houdini terms you are going to hear a lot throughout our slides
- Our technology -
  - Data goes back and forth - describe recipes, packed data
  - We've branched from houdini engine ~2 years ago. We'll try and point out where our tech may differ from base plugin.
- Use cases (each one describing a building block and demoing an advanced use case):
  - Point instancing -
    - Instancing plain ol' static meshes - building block
    - Our foliage

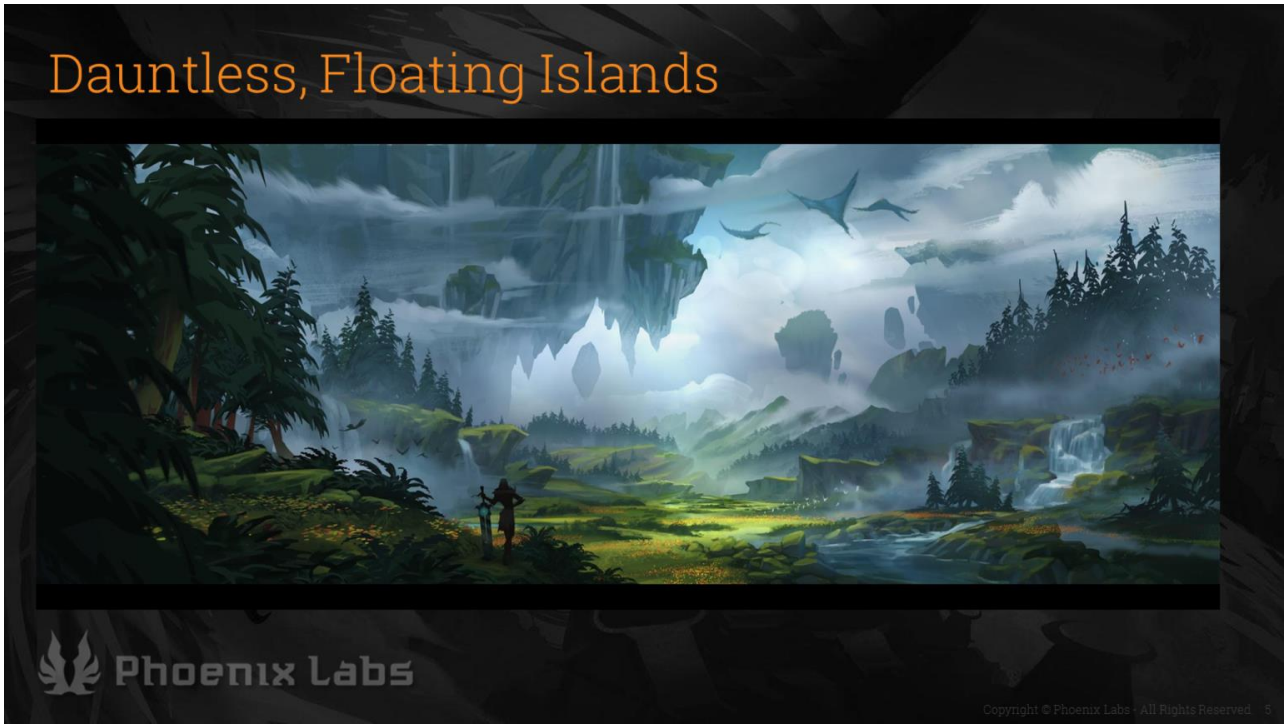
        - quilter
    - Geometry generation -
        - Custom mesh building block
        - land massing + early pipeline explanation
    - Landscape generation
        - Landscape building block *note that we may differ from the standard plugin, possibly open source the plugin if there's interest
        - Layers (primarily exposing unified noise layers with controls)
            - Originally a COP implementation but transitioned to heightfields
        - Accumulation
- Lessons learned + conclusions
    - Don't do one uber asset…
    - Rapid prototyping
        - Python and transition to HDK + Vex
    - Constant discussion / feedback with Jen. Our best stuff had tiny

feedback loops with art
- Decomposition into stages (modular nature)
- Save to hip file (quick back and forth between Houdini standalone and Unreal)
- All inputs and outputs were compatible with normal unreal systems and workflow
- Houdini is largely geo and attributes, getting beyond that to higher level data types



What is Dauntless? Explained briefly.

- Dauntless is a co-op, action RPG coming to PC, currently in early access.
- Battle ferocious Behemoths, craft powerful weapons, and forge your legend in the Shattered Isles
- We invite you to learn more and join our upcoming open beta (playdauntless.com)



Environments of Dauntless / Floating Islands, what are they.

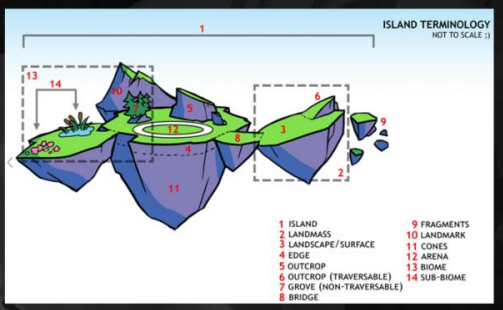
What we set out to achieve:

The shattered Isles play a critical role in dauntless. They're so much more than a passive backdrop for the experience. We (Mike and Mykola) joined the team because creating this epic setting with a small focused team was going to require something beyond the typical approach.

Island art requirements. Style. Biomes.

What we want technically

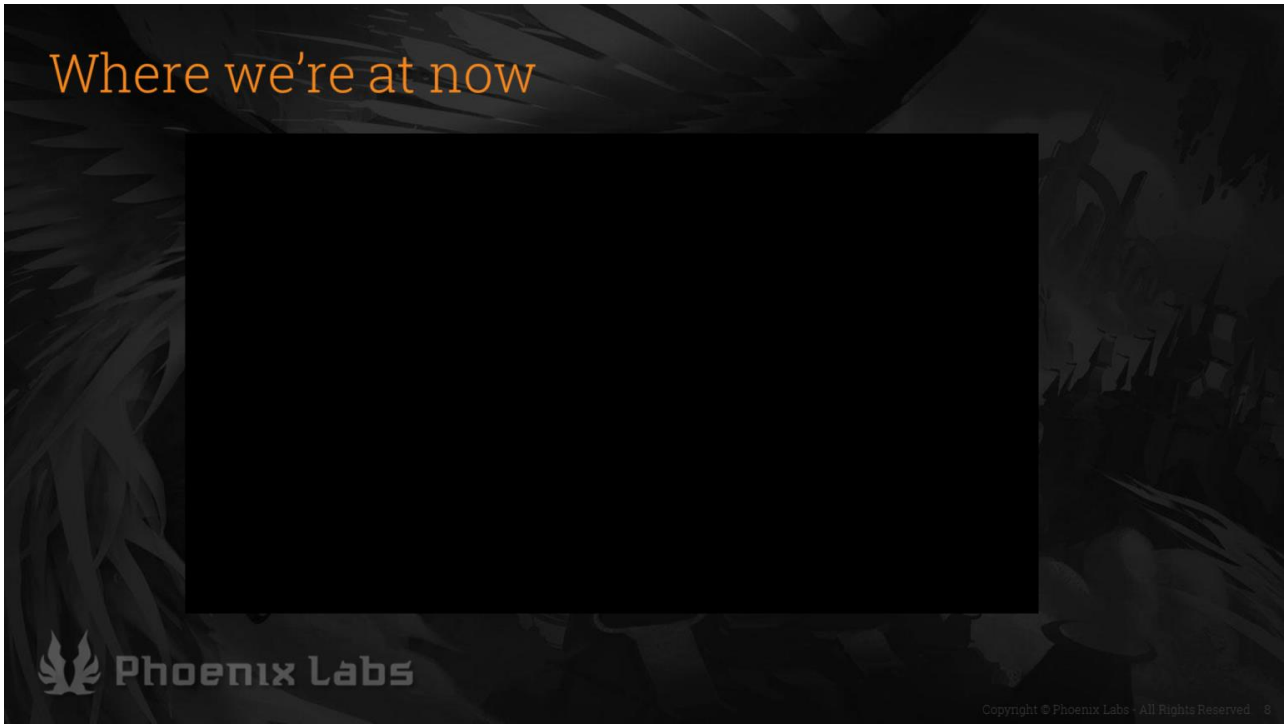
- Directed play spaces
- Unobstructed routes
- Reliable playspace topology
- Unreal compatible output
  - Landscape
- Adaptable components (modular)



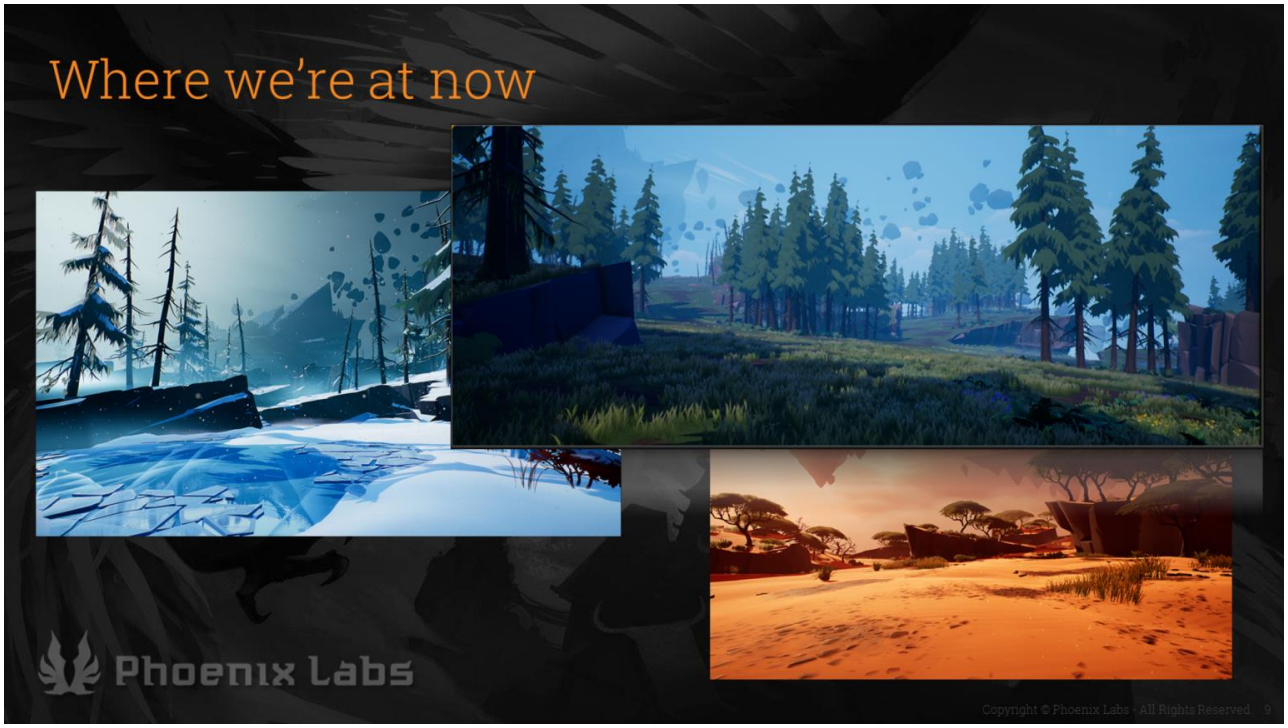
Phoenix Labs

The requirements were not all visual. On top of having believable floating landmasses, we had specific goals for a consistent, directable gameplay experience. Without covering all the details… as input, a user would specify a 3D configuration of land and connections - a high level design. We would need powerful controls to tailor the result. And finally, the output needed to be compatible with a normal unreal workflow (landscape, sometimes specific

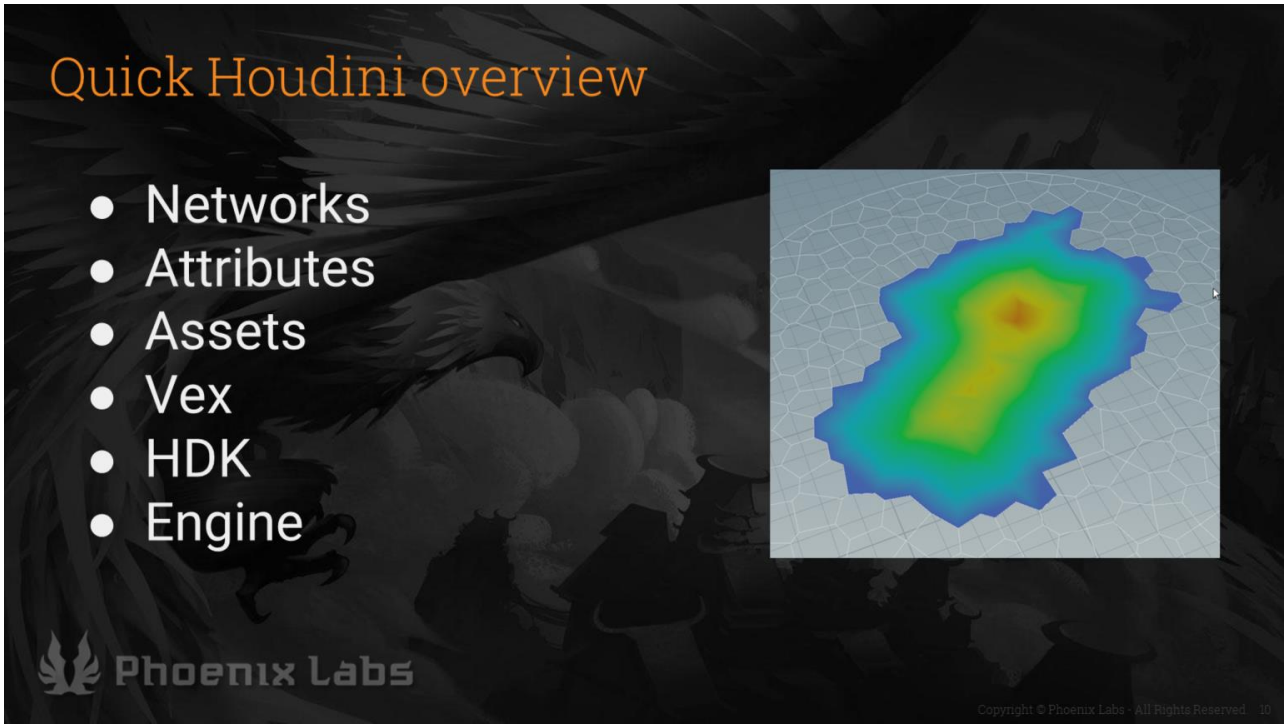
arrangements of instanced meshes, blueprints, volumes, etc...)



Play video demoing current state. Note that this has received polish (part of our standard process) on top of the cooked result.

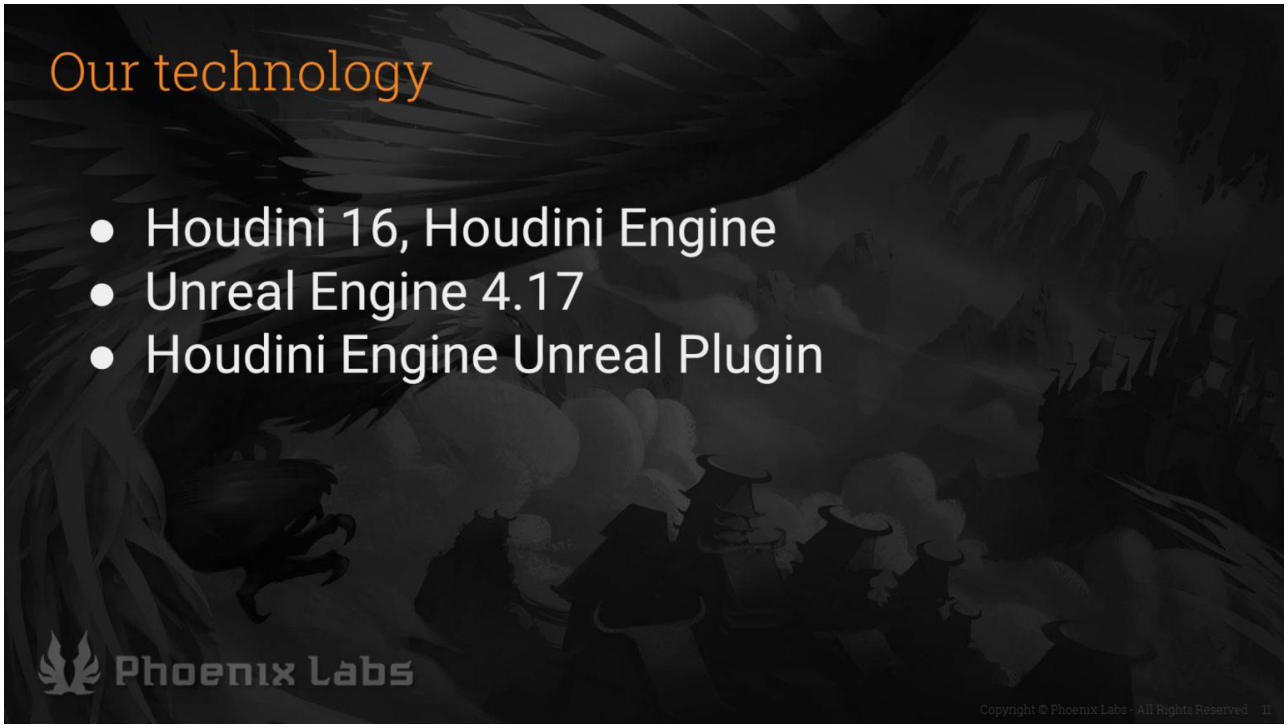


We currently generate all the required pieces for a fully playable dauntless island.
We have an initial 3 biomes (with sub area types) with more in the works.
Evolved design tools for easy direction of a game space

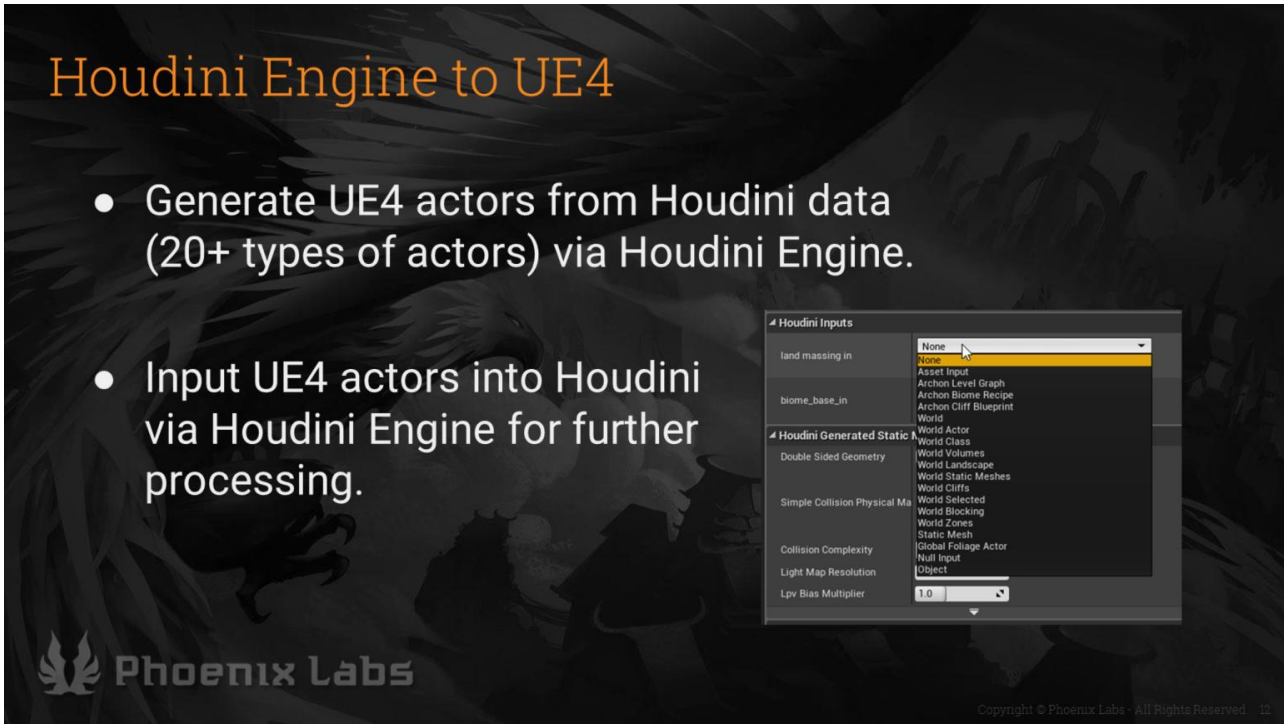


To properly describe some details of our implementation we'll quickly cover a handful of Houdini terms.

Very quick basic building block introducti on (houdini networks, attributes, assets, vex and hdk),
for people completely new to Houdini.

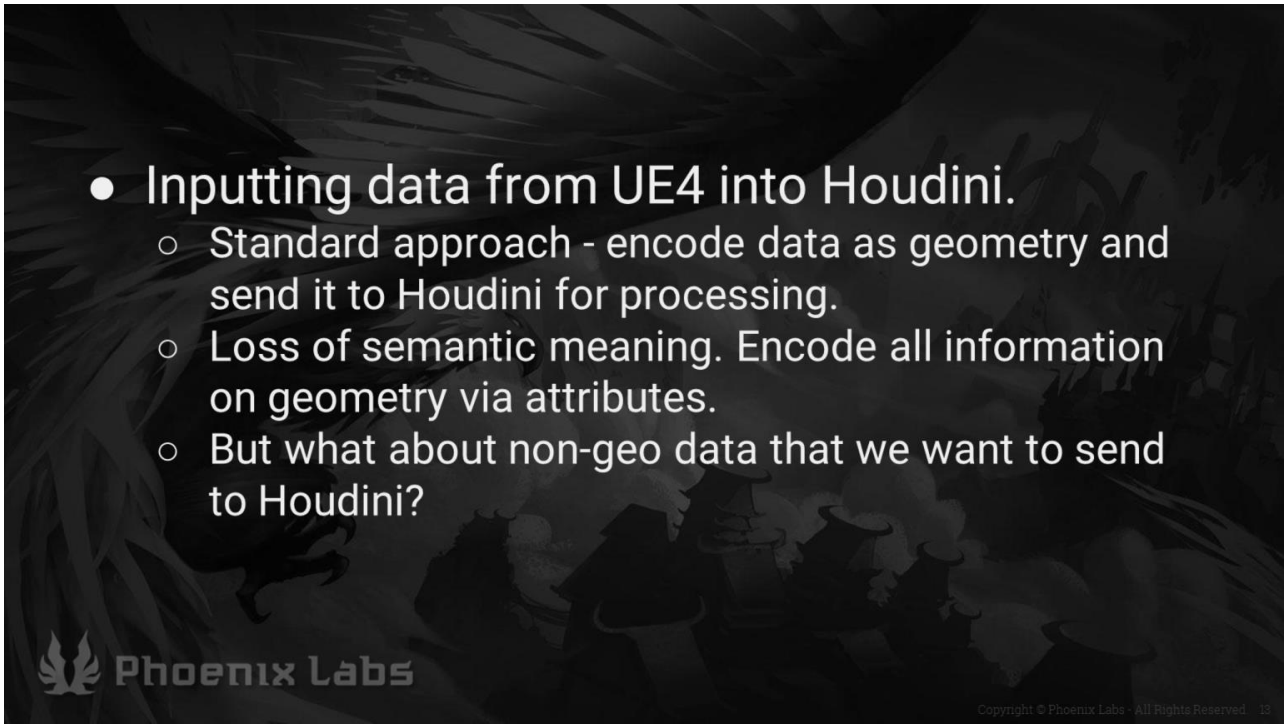


What we use - UE4, Houdini, Houdini Engine.



Houdini Engine use, describe the type of actors we could generate (staticmesh, landscape, foliage, blueprints, lights, and instancing support)

We occasionally input unreal actors back into houdini for further processing. Such as, custom mesh for moss trunks, snow accumulation on meshes, moss in cracks, etc...

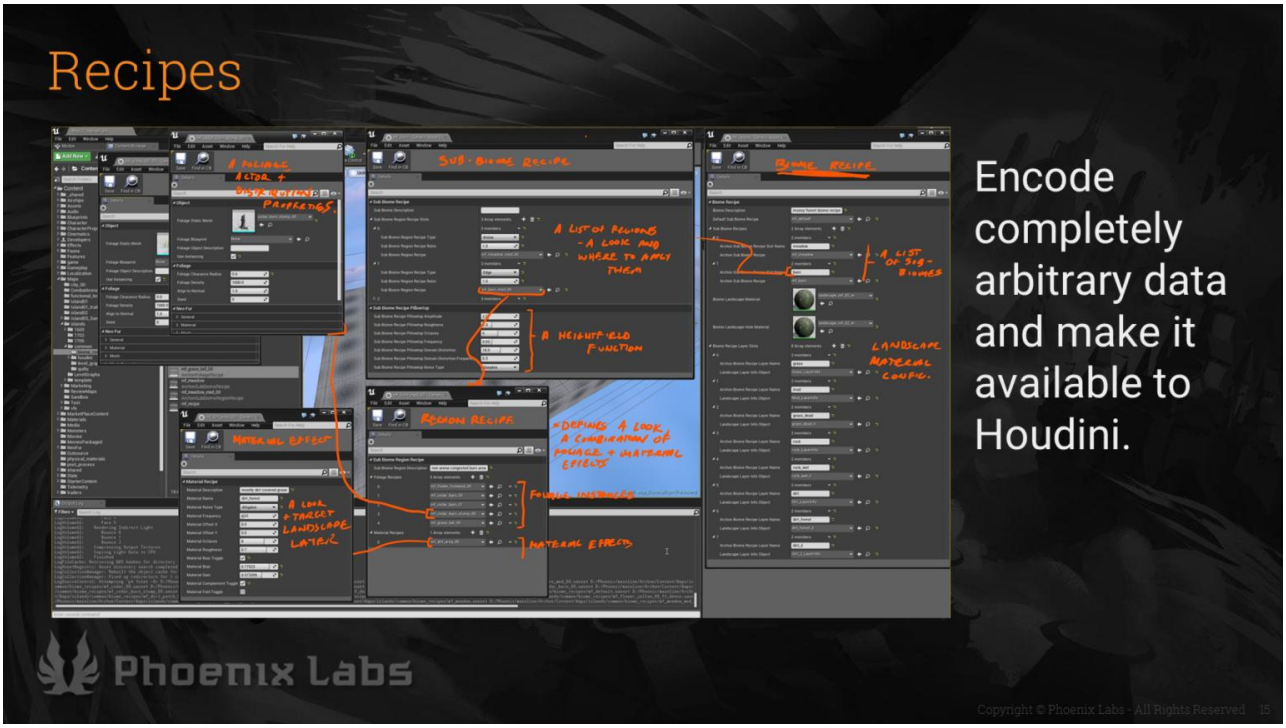


Moving data from UE4 to Houdini works out of the box for a lot of simple types (plain values, asset paths, curves…) but becomes a problem if we wanted to preserve any higher level data types.

Packing and Unpacking Data

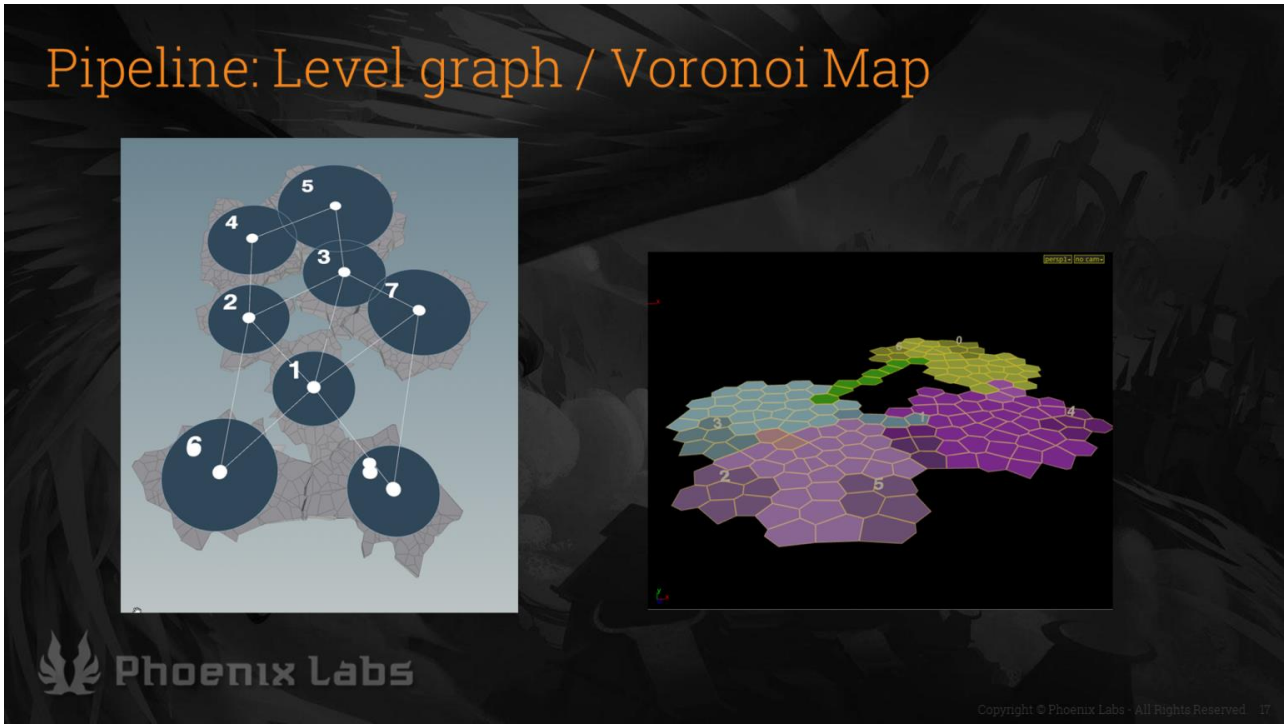
- Convert UE4 actors and objects (and all their properties) to binary JSON.
- Binary json attached to geo as int array attribute.
- Custom geo operators to decode within Houdini.

Phoenix Labs



Houdini presets work pretty well but we wanted to define a singular unreal side source (a recipe) for the biome configurations shown earlier. The details aren’t important at this point, but at a high level - a recipe is a complete visual configuration for a biome and it’s sub-areas.

And here’s a shot from the output of that same recipe.



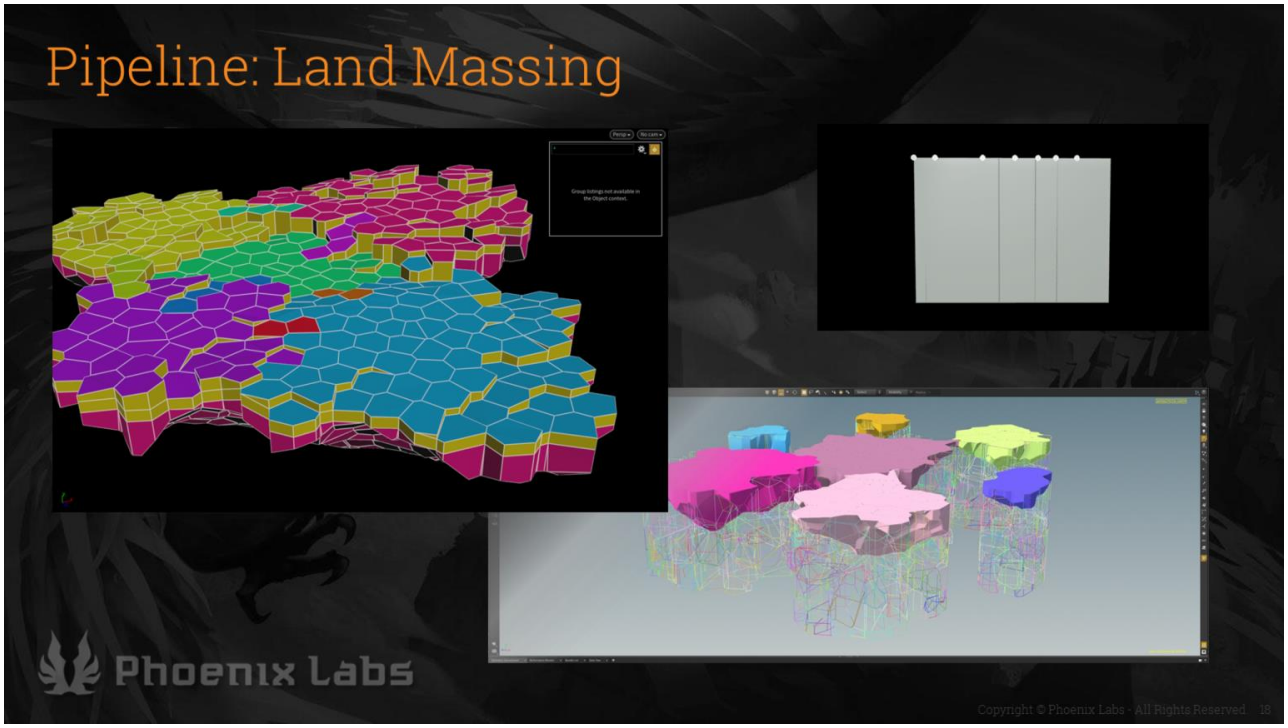
We'll quickly go over our pipeline stages at a high level to give context to some of the later pieces.

Level graph:

- Basic graph describing the zones (chunks of playable land), their position/orientation, and their connectivity. Basically chunks of land and any important navigation connections.

Voronoi map:

- Intermediate 2D stage where we resolve all the map constraints (connectivity, area/shape, and some topological information like outcrops + slope)
- We use a basic 2D voronoi fracture to define the individual cells
- We build bridges at this point and save navigation network information

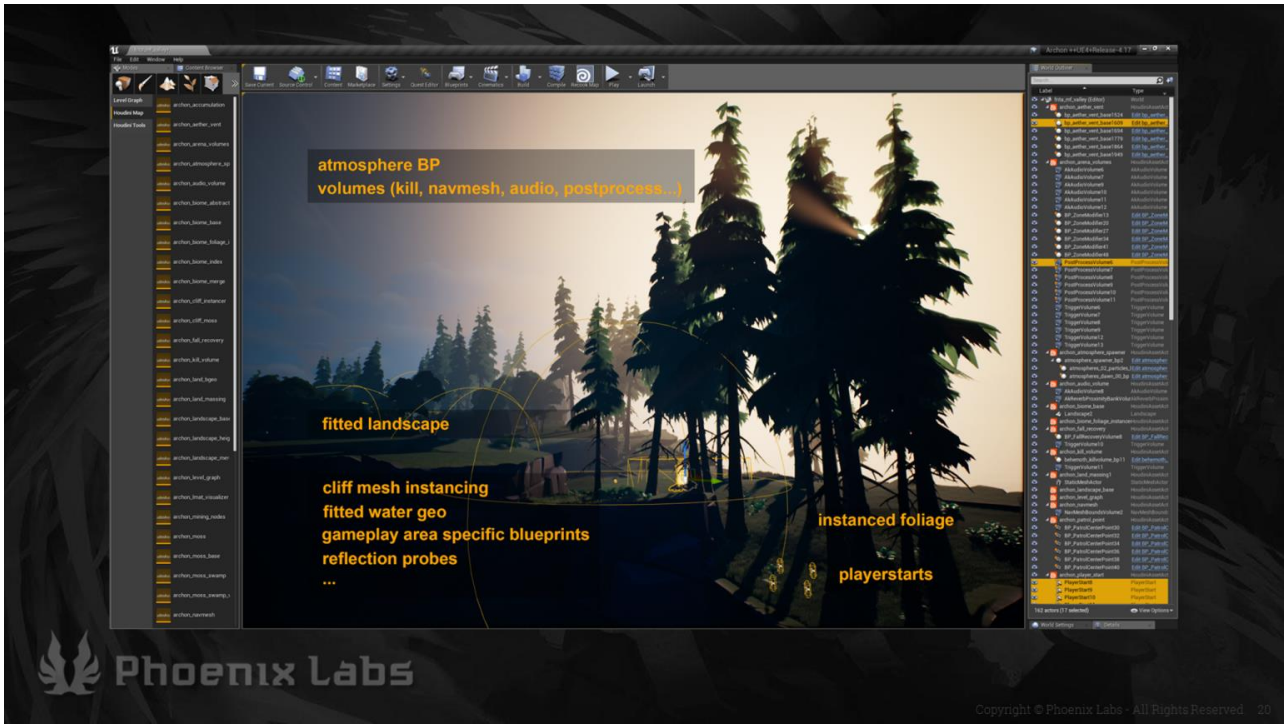


A 3D representation of our level is constructed from the 2D voronoi map.

- Mesh is clean / watertight
- Maintain a list of primitive groups and attributes in order to preserve additional information (adjacency, area type, navigation network)
- Island base is cut using manipulation of the points in a voronoi fracture (pictured as a 2D cross section in the top right)

Outcrops:

- We provide overall density and individual override controls
- We approximate stress vectors for the colliding land masses and orient outcrops accordingly.
- Setting and adjusting land or outcrop positions is the majority of the design work when developing a level.



- From the simple underlying 3D base of the previous stage we generate all the remaining required components of a fully-playable dauntless island.
- Rather than cover all of these very briefly, we wanted to dig into details on a select few.

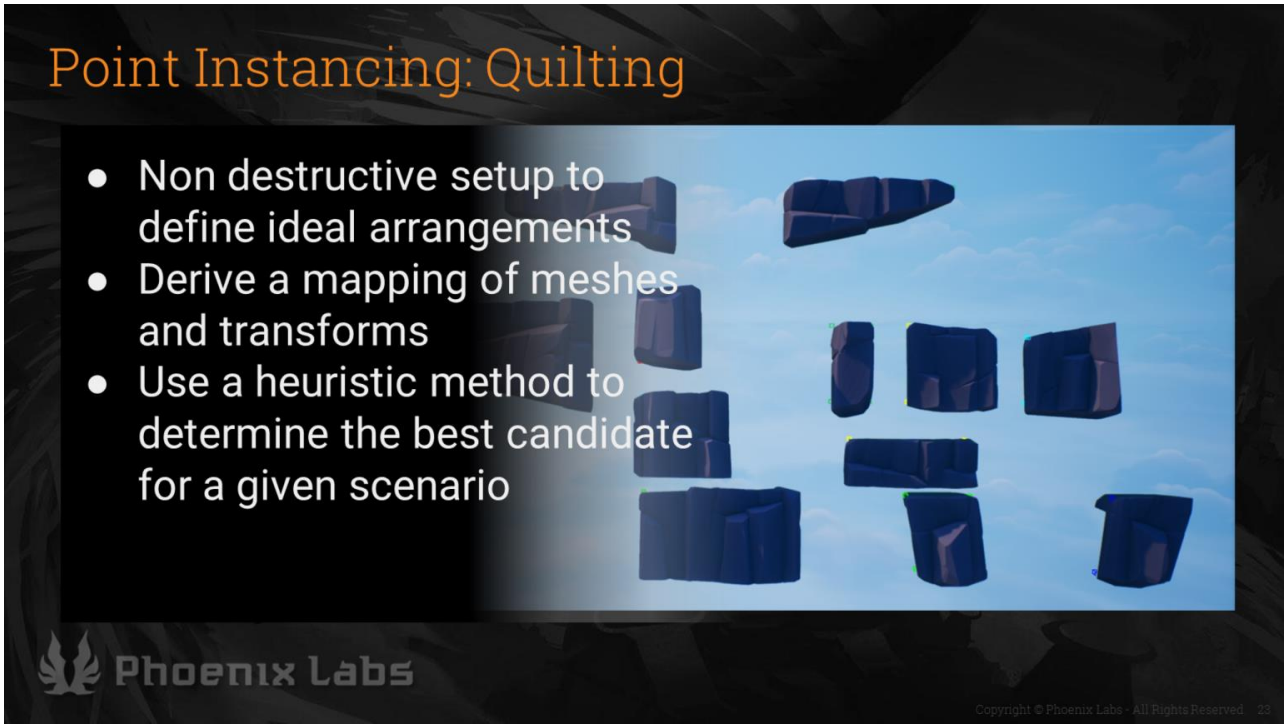
The first few steps with a pipeline that involves houdini engine can be a little overwhelming. To help…
We'll show a basic setup that can be used to create a component of our pipeline and a more complete example of where that can lead.

This asset demonstrates the basic minimum required to scatter instantiated static meshes in any project. We have a simple string point attribute defining the

unreal path to a static mesh resource and another integer attribute defining whether the resulting points should be instanced or individual actors. (Mykola?)

You'll commonly want more complete transform information so we'd typically include the optional attributes scale (float3) and orient (float4)

This is a good time to point out that random distribution of assets was rarely the solution for many of the problems we faced. More often we wanted a very specific arrangement of meshes.
We wanted Jen to be in control of an evolving visual target. Our approach works off the idea that we have sample(s) demonstrating ideal and we can extrapolate from those to solve a new scenario.

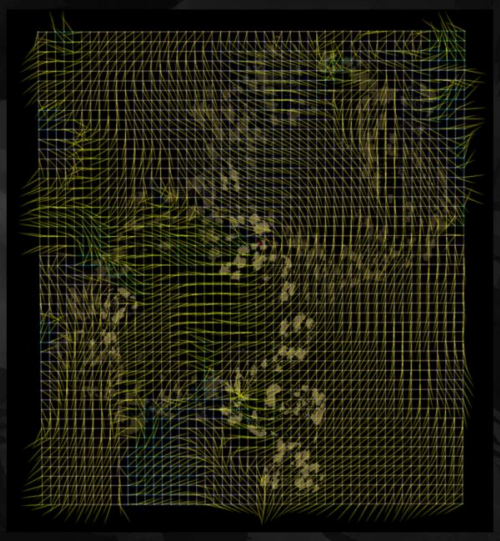


Describe quilting in more detail. How we establish the ideal for a pattern or arrangement (via blueprint) and use that to populate (“quilt”) a new space. We use this for cliffs specifically but the solution is generic. Our solution supports instancing, and has configurable controls to minimize distortion (uniform and non-uniform scale differences).

And here you can see some sample output of our ‘quilter’ assets (prior to any art polish)

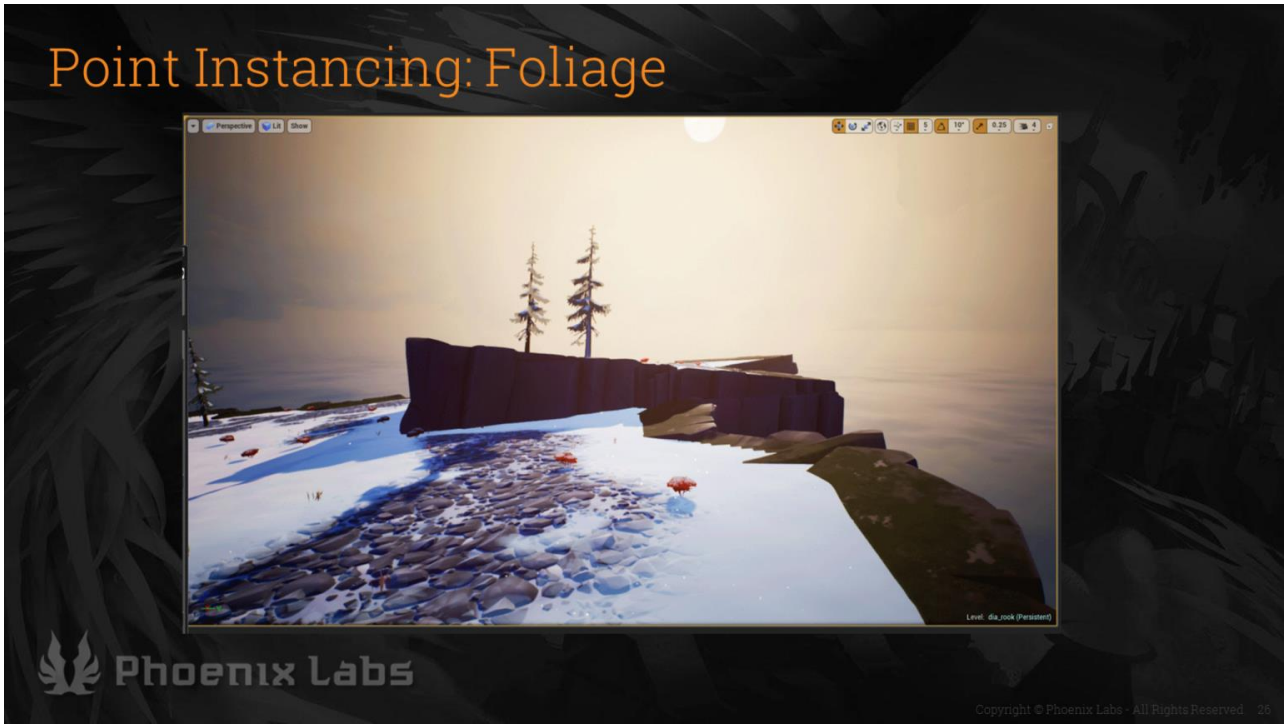
Point Instancing: Foliage

- Very simple growth simulation
- Plant properties set how and where it will propagate
- Used a low res 2D vector field (fake wind) to influence growth direction



Phoenix Labs

We run a *very* simple growth simulation to propagate a number of asset types. The goal was natural looking distribution but it was critical the results could be easily understood (and tweaked) with art. Plants had a handful of key properties that let us control where(material types) and how(alignment, clumping) it grew.

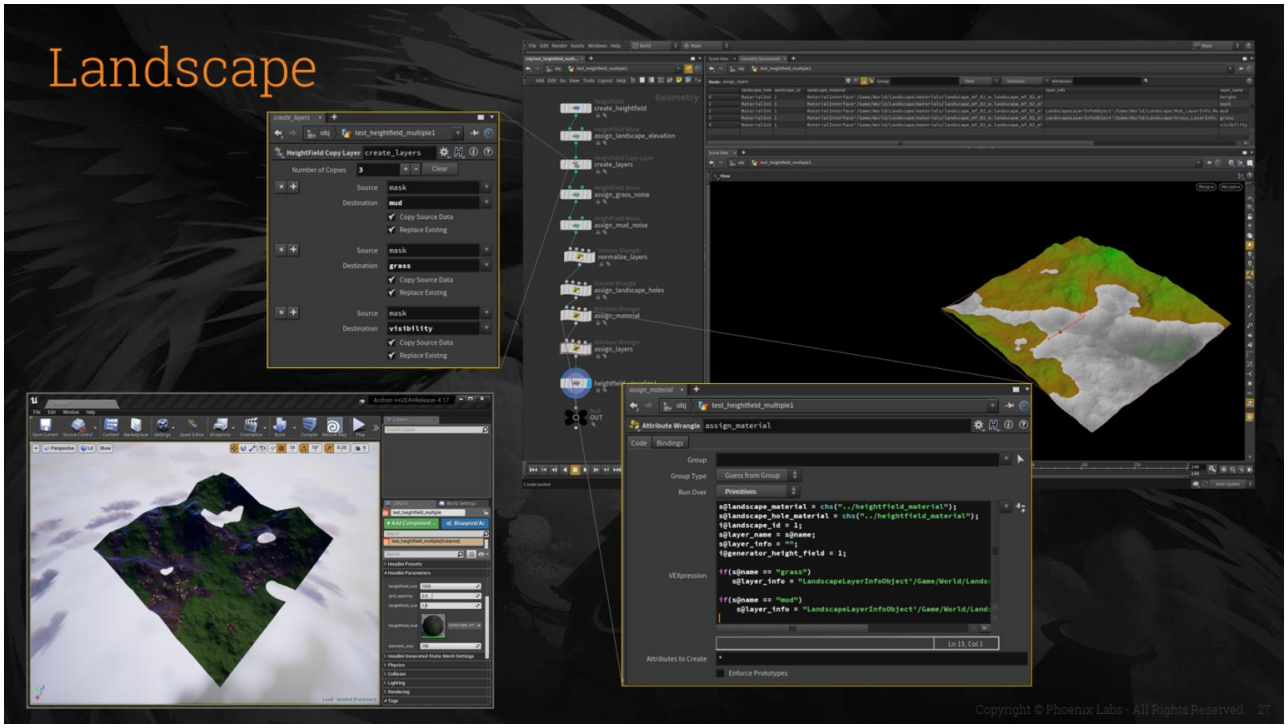


This is a simplified version of Unreals foliage propagation volumes. (We wanted more specific control of the individual point instances within Houdini)

- We simulate a fixed number of generations, each allowing a plant to reseed and age.
- Plants respect a shade radius of neighbouring plants, allowing us to control the intersection and clumping density of adjacent plants.
- Reseeding directions are influenced

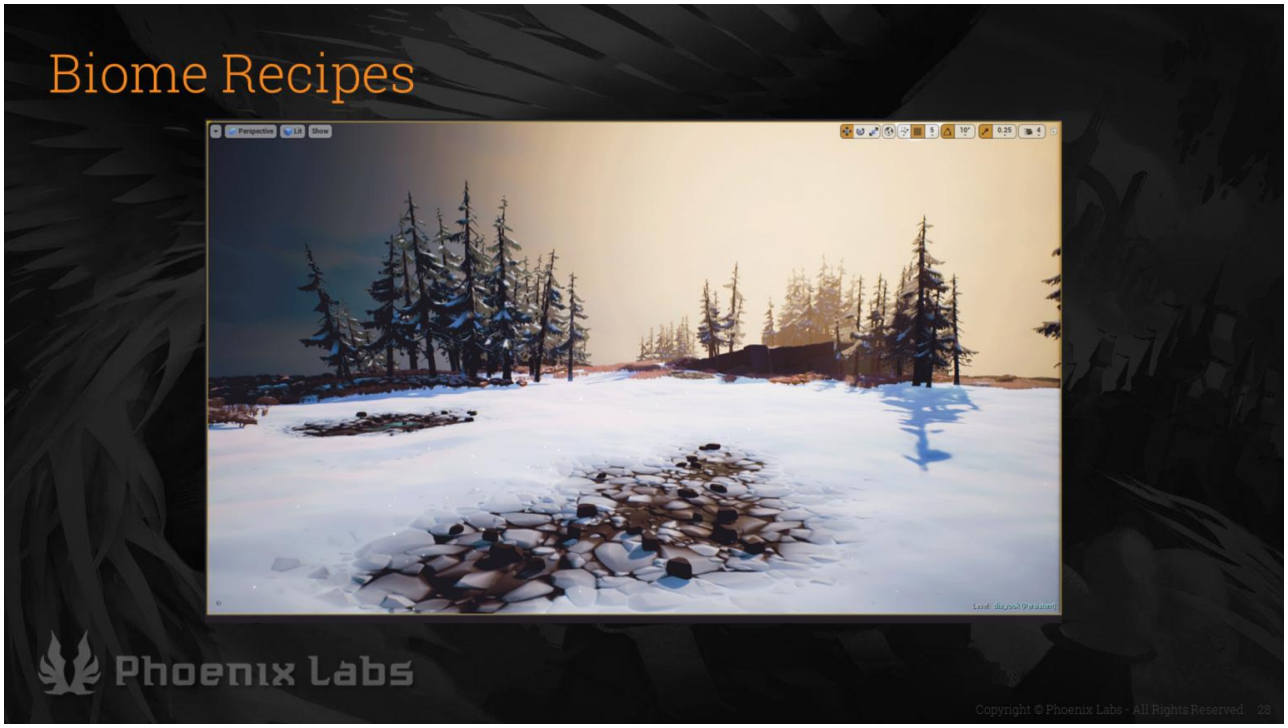
by a simple noise based vector field (“prevailing winds”) to improve cluster shape.

- For performance reasons we allow a fixed number of reseeding attempts.
- Based on plant properties, some are masked/culled. (ie. not allowed in water, too near the island edge…)

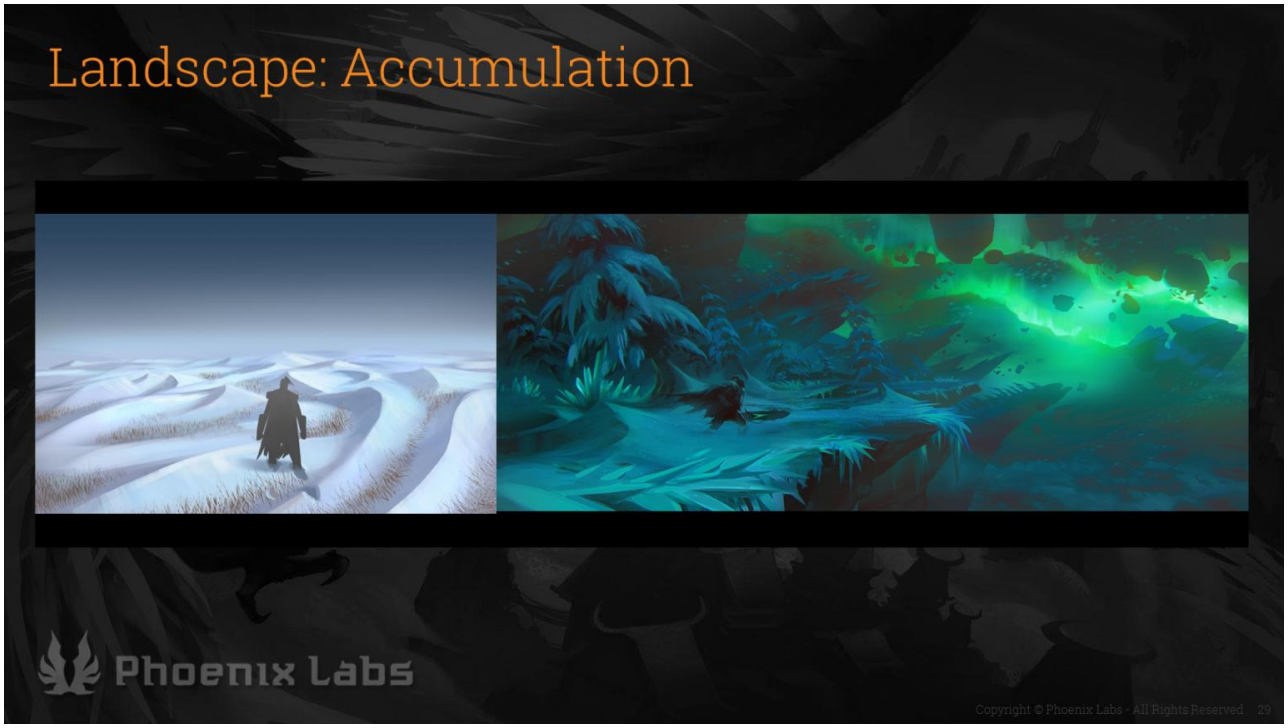


[Mike]
We'll show the bare minimum setup required to output a fairly standard landscape actor (possible differences with our plugin?)
In the create layers node we've set up our two main material layers and provided a visibility mask for holes
There's a few more attributes required than our previous point example but the setup is still fairly straightforward
They're all driven by some standard

houdini noise in this example.



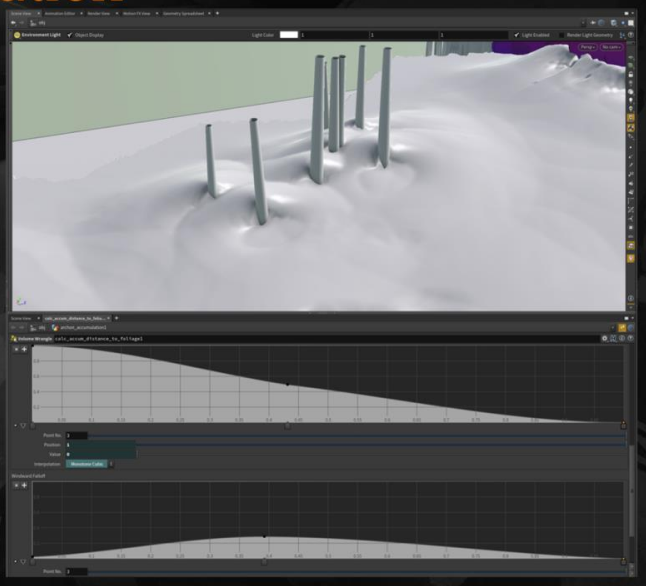
Here’s an example of driving the water level and a few landscape materials inside of the ‘melt’ area of our icy biome recipe.
We pre-compute layer weights (normalize) during generation time. The layers (and their count) are all recipe driven.



Stages of our pipeline.

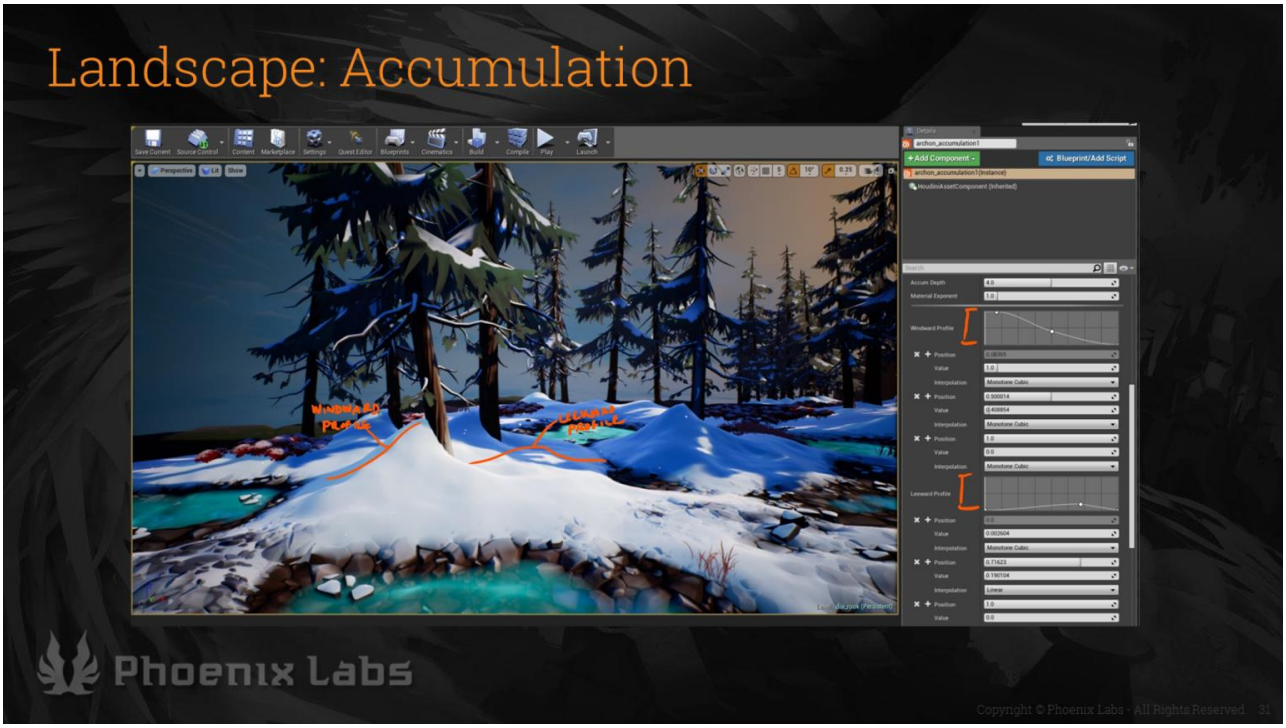
Landscape: Accumulation

- Special landscape layer
- Tied to render target
- “Simulate” natural buildup (sand, snow, etc.) against world surfaces
- Happy to share more details

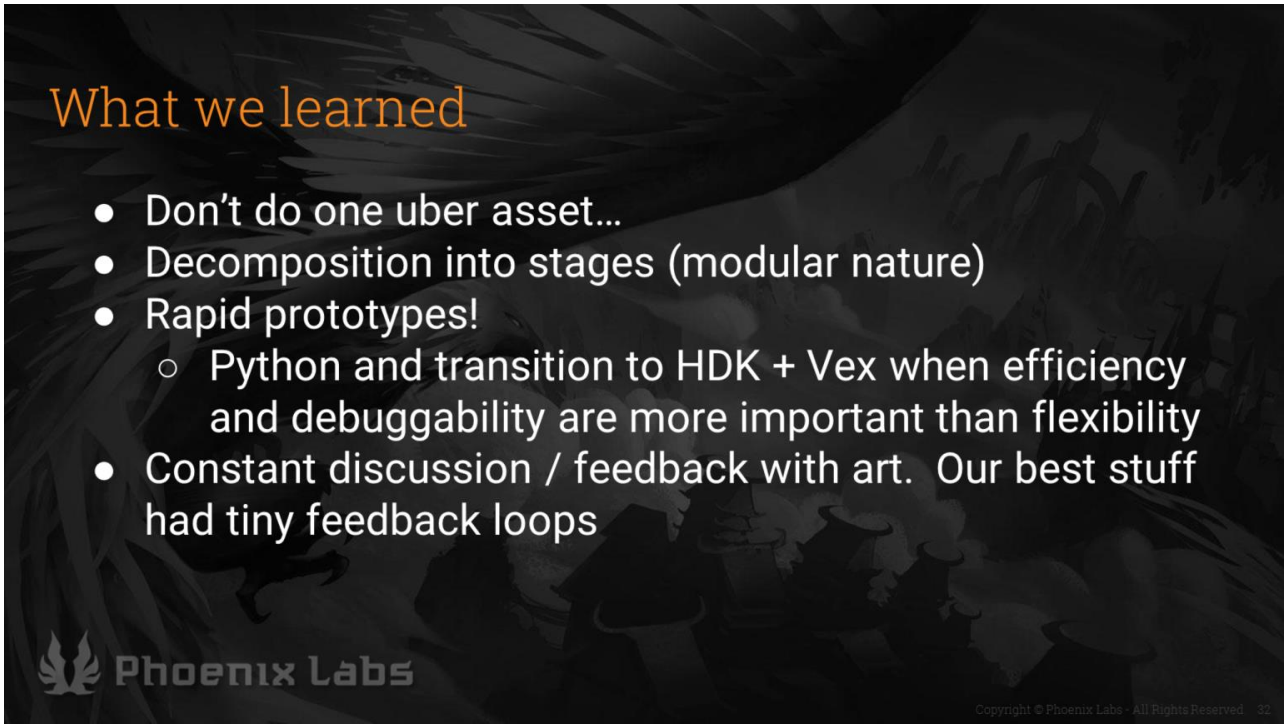


Phoenix Labs

Stages of our pipeline.



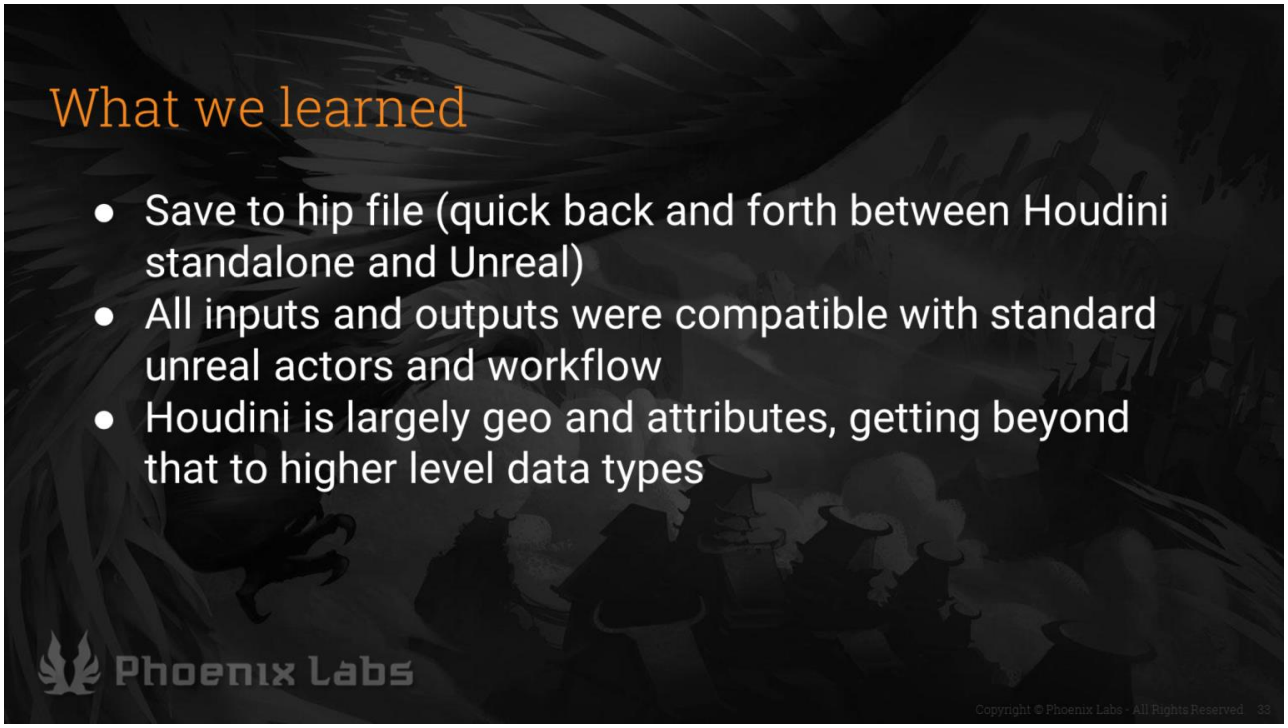
Stages of our pipeline.



- Lessons learned + conclusions
  - Don’t do one uber asset…
  - Rapid prototyping
    - Python and transition to HDK + Vex when efficiency is more critical
  - Constant discussion / feedback with Jen. Our best stuff had tiny feedback loops with art
  - Decomposition into stages (modular nature)
  - Save to hip file (quick back and

forth between Houdini standalone and Unreal)

- All inputs and outputs were compatible with standard unreal systems and workflow
- Houdini is largely geo and attributes, getting beyond that to higher level data types



- You might rephrase that as getting us to think more unreal than getting jen to think more houdini-like.

Thank you!
We moved quickly and couldn’t always cover as much detail as we wanted to.
We’re more than happy to answer questions and fill in gaps.