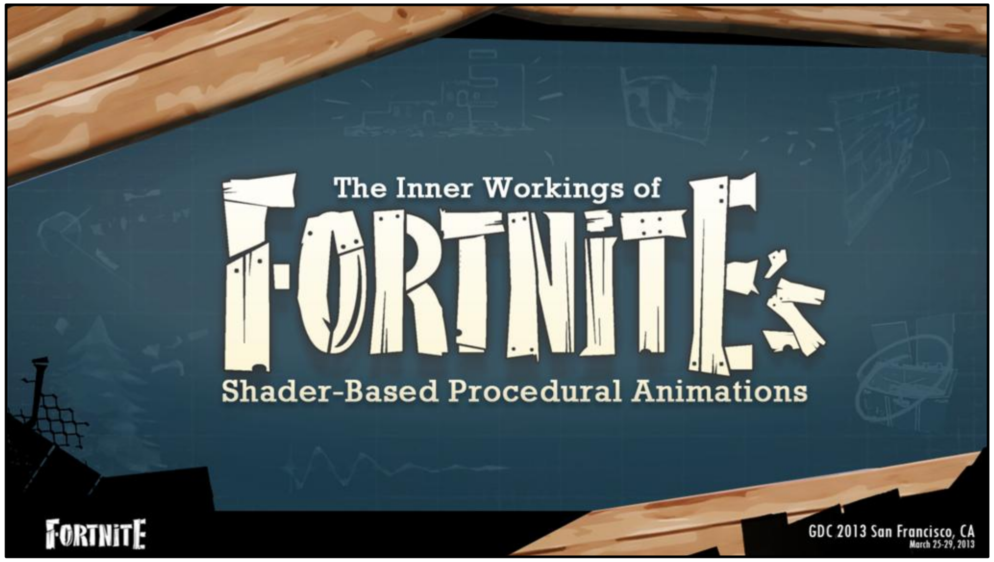


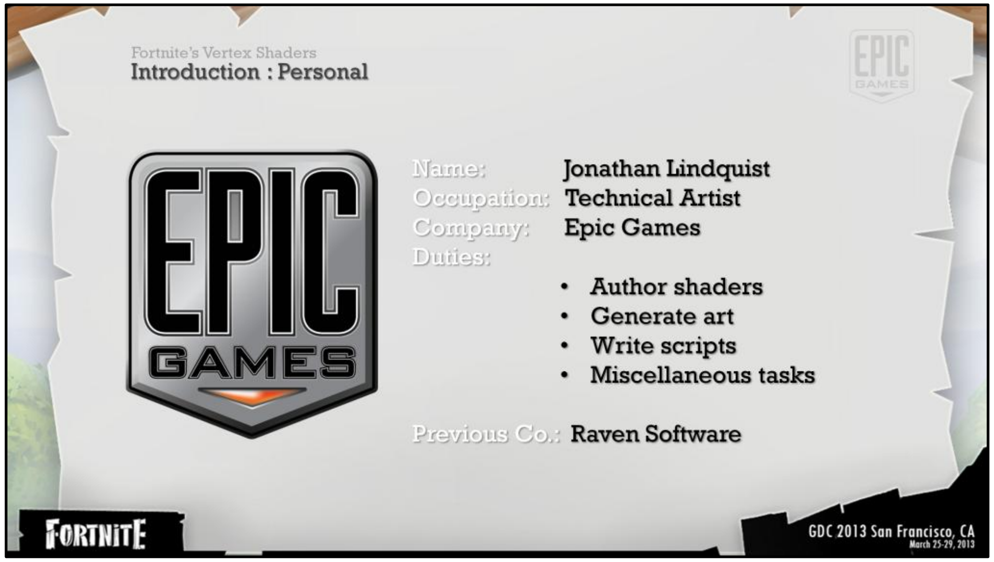
The Inner Workings of FORTNITE's Shader-Based Procedural Animations

FORTNITE

GDC 2013 San Francisco, CA
March 25-29, 2013

Welcome to The Inner Workings of Forntite's shaders.

We are going to talk about some of the challenges we faced on Fortnite and vertex shaders provided efficient solutions due to their unique capabilities.



Fortnite's Vertex Shaders

Introduction : Personal

Name:	Jonathan Lindquist
Occupation:	Technical Artist
Company:	Epic Games
Duties:	

- Author shaders
- Generate art
- Write scripts
- Miscellaneous tasks

Previous Co.: Raven Software

FORTNITE

GDC 2013 San Francisco, CA
March 25-29, 2013

My name is Jonathan Lindquist and I'm a technical artist working on Fortnite at Epic Games.

The work I do on the project varies on a regular basis. I create art, author materials and write online and offline scripts.

I spent three years at raven as a technical artist and spent the last two at Epic working on a variety of projects.

Fortnite's Vertex Shaders

Introduction : Project

FORTNITE

- Co-operative
- Action
- Building

FORTNITE

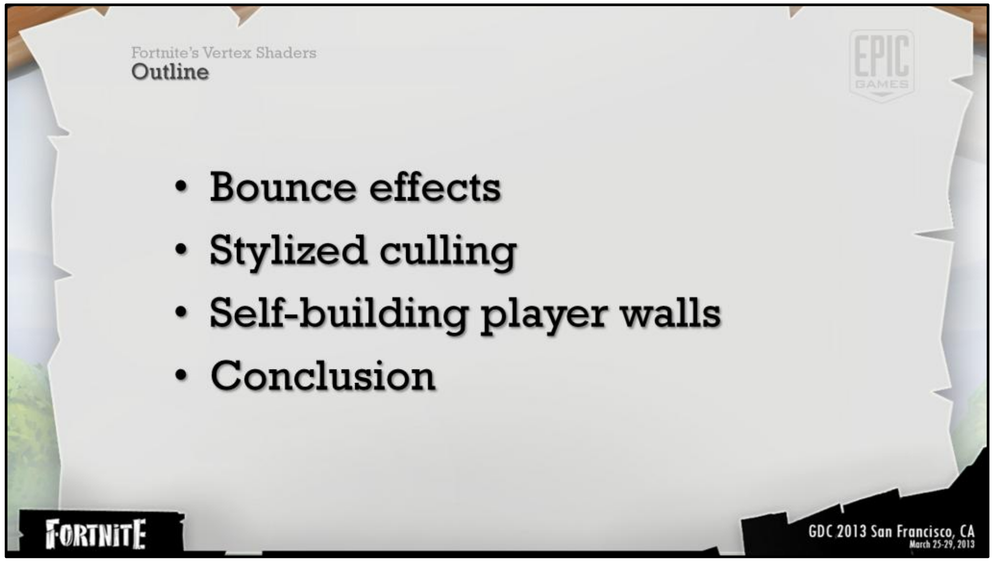
GDC 2013 San Francisco, CA
March 25-29, 2013

For anyone unfamiliar with the title, Fortnite is Epics first ue4 game. The visuals are stylized and the gameplay is mostly co-op action with a building mechanic.

Fortnite's Vertex Shaders

Introduction - Project

Lets look at some of the effects that we will be covering today.


Fortnite's Vertex Shaders

Outline

- Bounce effects
- Stylized culling
- Self-building player walls
- Conclusion

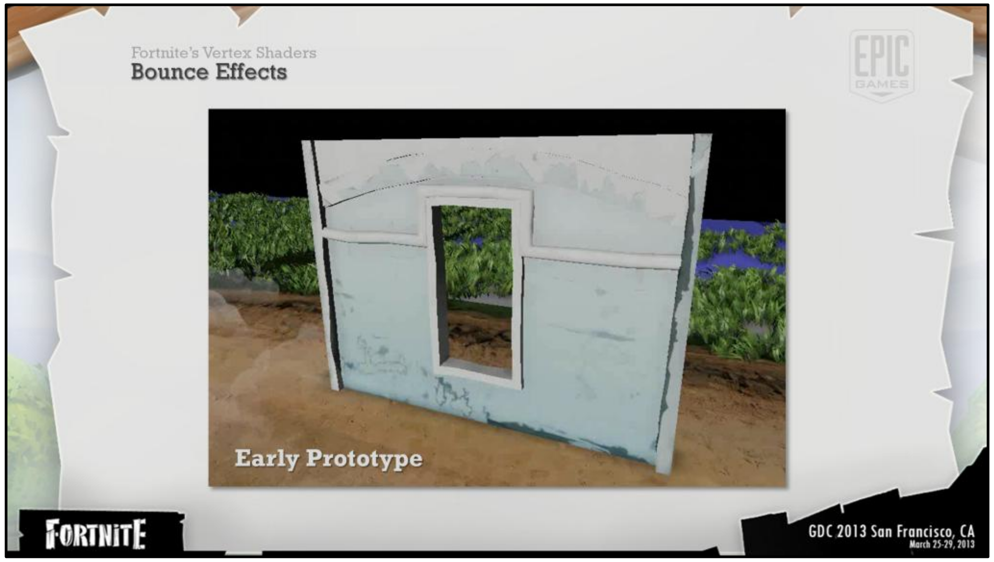
FORTNITE

GDC 2013 San Francisco, CA
March 25-29, 2013

Challenges:
Many design concerns needed to be addressed within our stylized world.

So we will start by examining the method we use to shake assets on impact when they're touched. Which is one of the first tasks that we needed to perform give the player feedback from the world.

Mention that World position offset is Unreals input for authoring vertex shaders. Additive offset to the objects world position on a per vertex basis.


Fortnite's Vertex Shaders

Bounce Effects

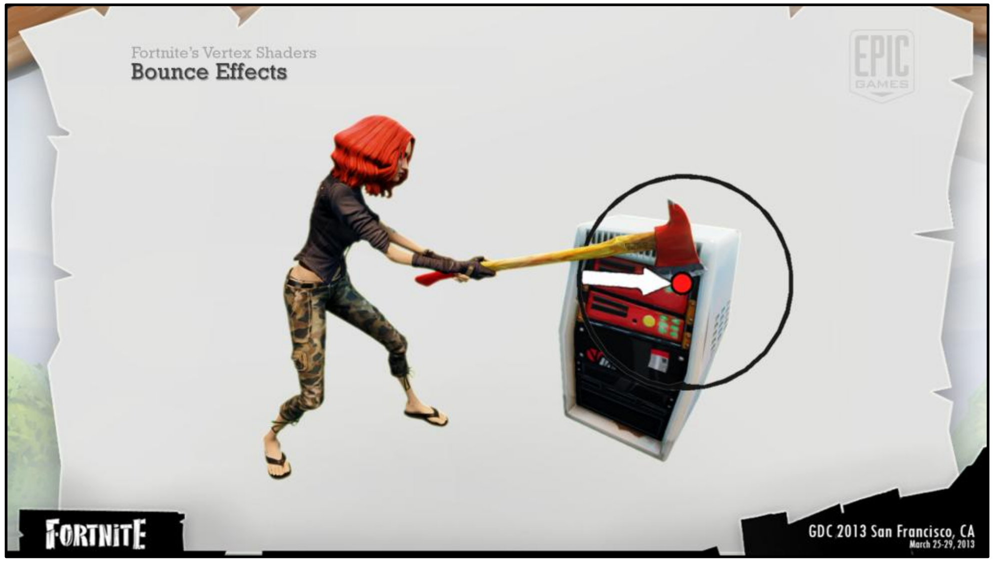
Vertex shader prevalence throughout the title started with the setting the goal to make destroying buildings fun.

During our experimentation phase we tested various methods for generically damaging structures. Automated fracturing systems, decals, particle effects etc.

None of them spoke to the visual style of the game produced noticeable visual errors as well as performance concerns.

A simple vertex shader test stood out amongst the iterations and appeared to be very promising.

We found the effect generic enough to be useful for other effects afterword.



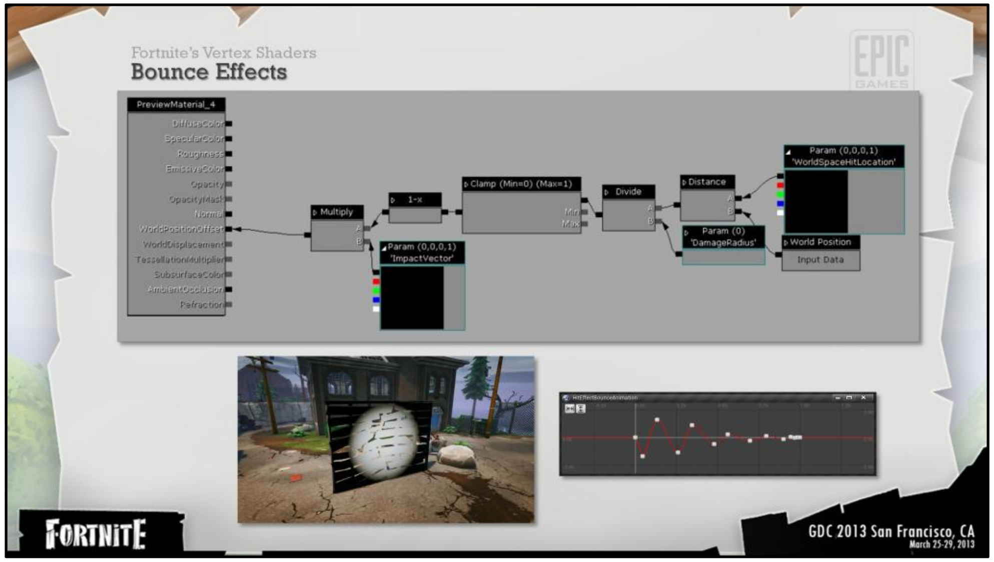
There are two parts to the effect. Gameplay code feeds several values into the object shader and the shader then processes that information to create an additive offset to vertices in the range.

Gameplay code supplies the objects shader with three values

the weapon impact location

and the impact vector

a radius for the effect of the impact


Fortnite's Vertex Shaders

Bounce Effects

GDC 2013 San Francisco, CA
March 25-29, 2013

Mention that the graph above is our material editing system. A node based visual programming tool.

As you can see the code is very simple.

To generate a falloff from the impact location we find the distance between the vertices World position and the code provided hit location. Then we divide the result by the damage radius, clamp it and invert it. Which will provide a 1 to 0 falloff like show in the picture video below.

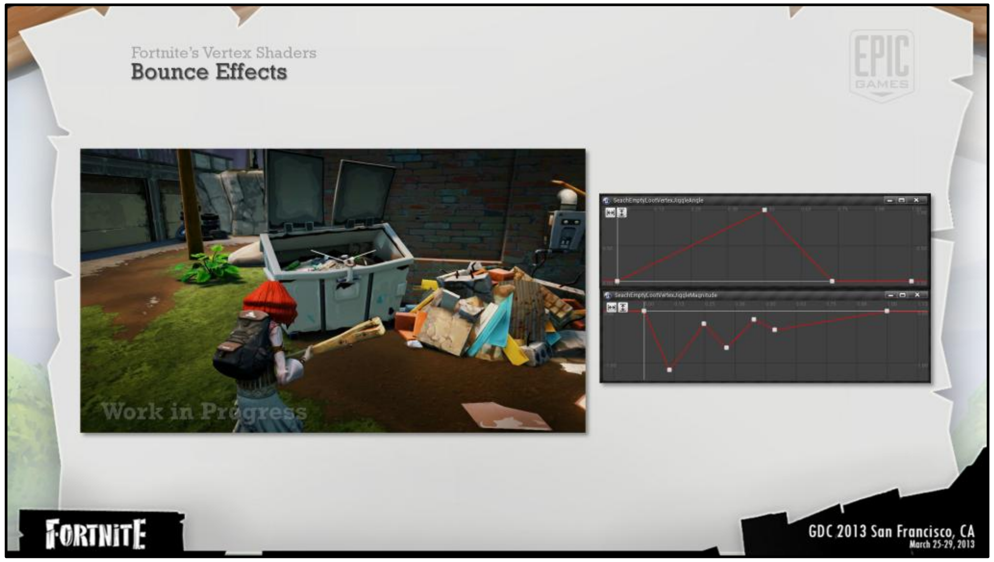
Afterword the black and white values are multiplied by the impact vector which is modulated by the artist authored animation curve seen below.

Code samples the Y value along the length of the spline at a speed of one X unit per second.

Every process that can moved off of the gpu and onto the cpu has to reduce the number of vertex shader instructions across the game.

We heavily rely on a material instancing system to ensure that the proper code is added to every material and the necessary variables are exposed on every asset.

The code was generic enough to allow us to repurpose it for other effects as well.



Fortnite's Vertex Shaders

Bounce Effects

Here you can see that objects bounce when the player interacts with an object in the world.

We reused the same vertex shader code to perform this function.
The only difference is the values being fed into the shader parameters.

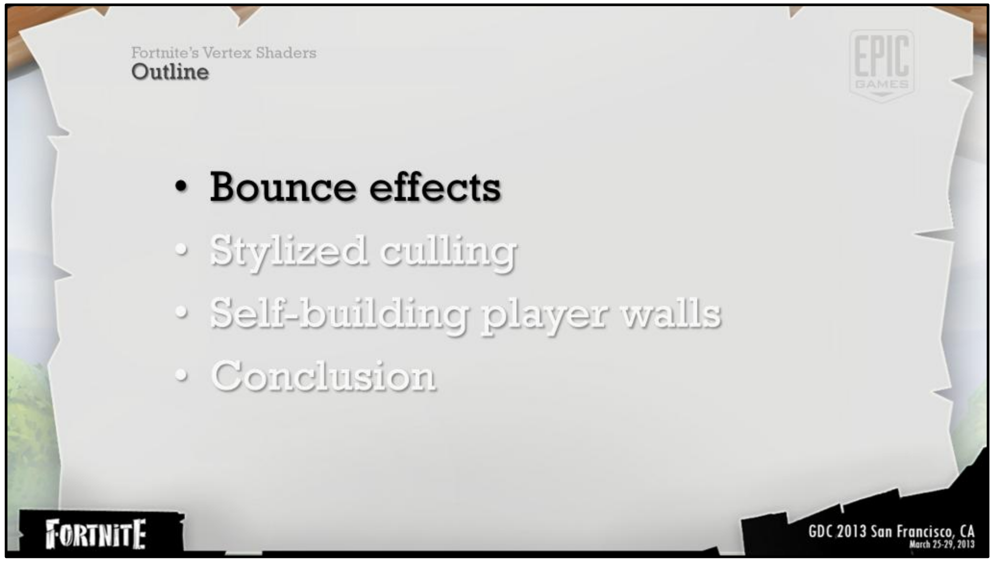
In the video you may notice that the two white lines appear when the object shakes. These are debug lines indicating the most extreme vectors that that the object can bounce along.

They are provided by code and were created by transforming specific directional vectors from the player cameras local space to world space.

The yellow line that moves around is the result of linearly interpolating between the two extreme angles using an animation curve and then modulating the magnitude of the resulting vector by another animation curve object output.

As a side note: we heavily use a material instancing system which makes propagating shared vertex shader code very simple.

The resulting vector is used to replace the “Impact vector” from the previous example.



Fortnite's Vertex Shaders

Outline

- Bounce effects
- Stylized culling
- Self-building player walls
- Conclusion

FORTNITE

GDC 2013 San Francisco, CA
March 25-29, 2013

Exploration is another key component of Fortnite. To support the lush worlds that we wanted we needed to apply distance culling.

To continue on exploring the visual style we ran a few tests to replace the common approach of masked dithering with a specialized approach.

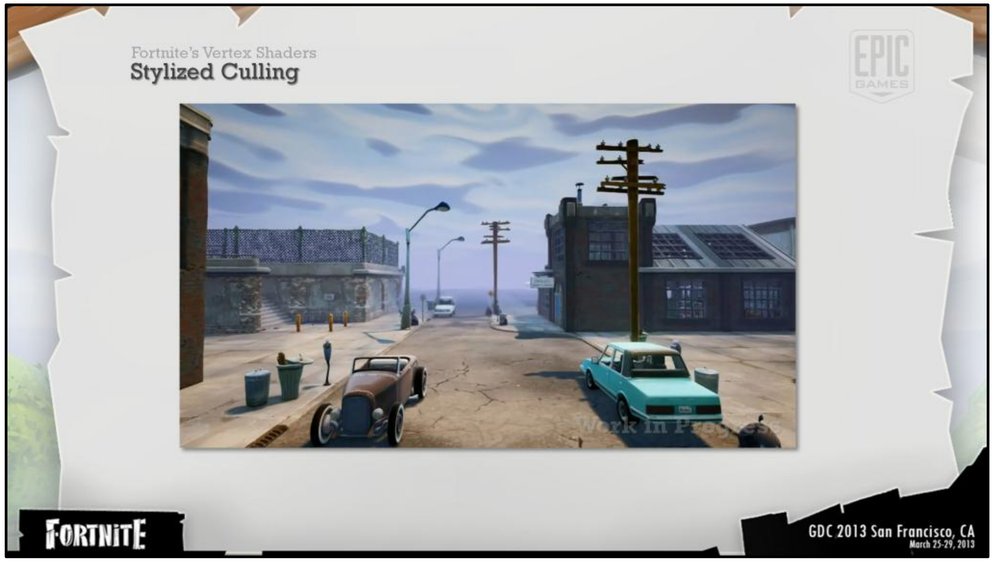
Fortnite's Vertex Shaders

Stylized Culling

FORTNITE

GDC 2013 San Francisco, CA
March 25-29, 2013

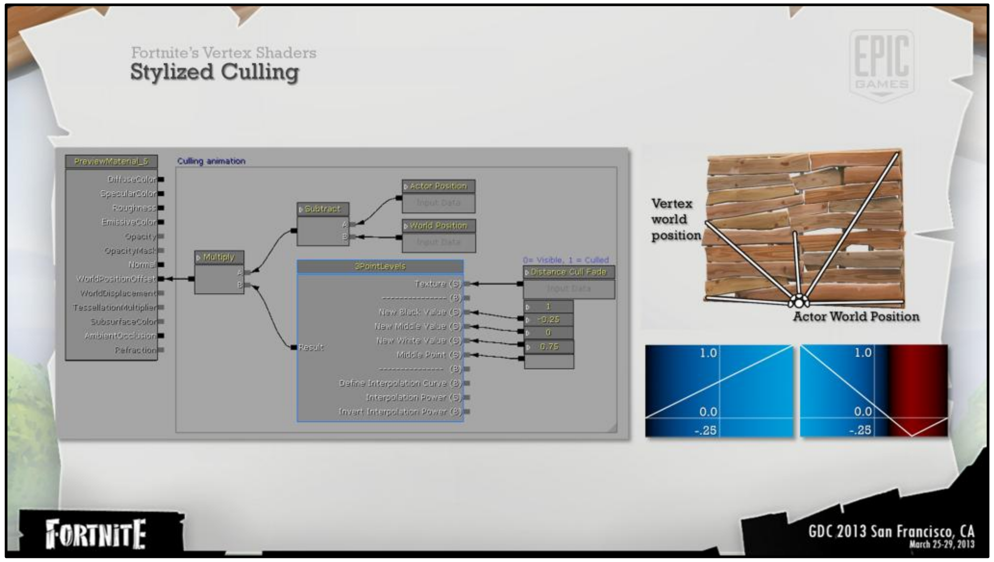
For entertainment purposes, I wanted to show some early prototypes.


Fortnite's Vertex Shaders

Stylized Culling

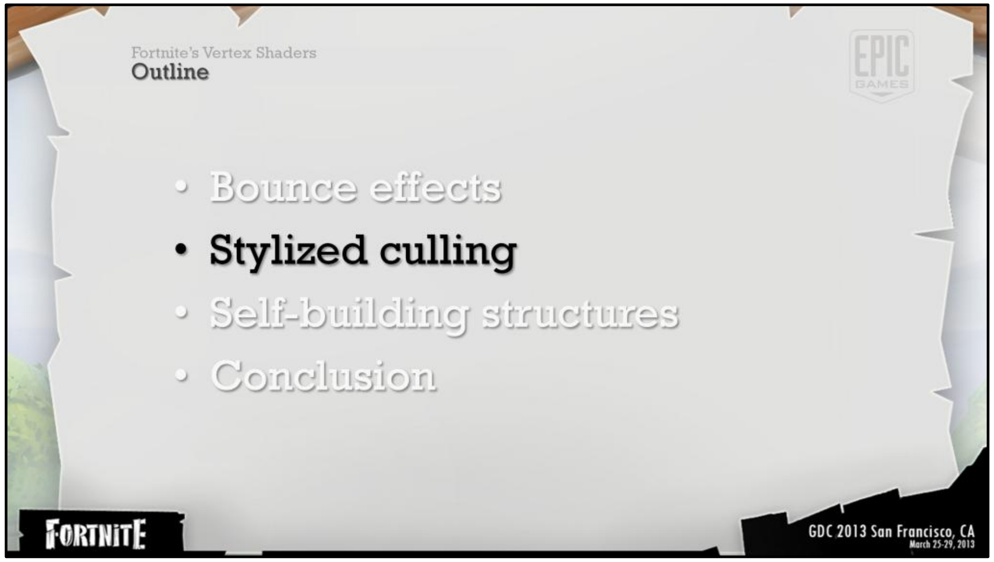
This is the final in game culling effect. The draw distance is considerably reduced to show the effect.

In the end we found that the large scale mesh stretching from below the map was too distracting whereas the scale effect was far less so.



The culling effect relies on scaling code within the vertex shader, To avoid manipulating the actors transforms which would have caused us to tick the actors.

This time we subtract the actor pivot point from the vertex world position then modulate the returning vector with a value that animates from 0-1 as objects are culled.



Fortnite's Vertex Shaders

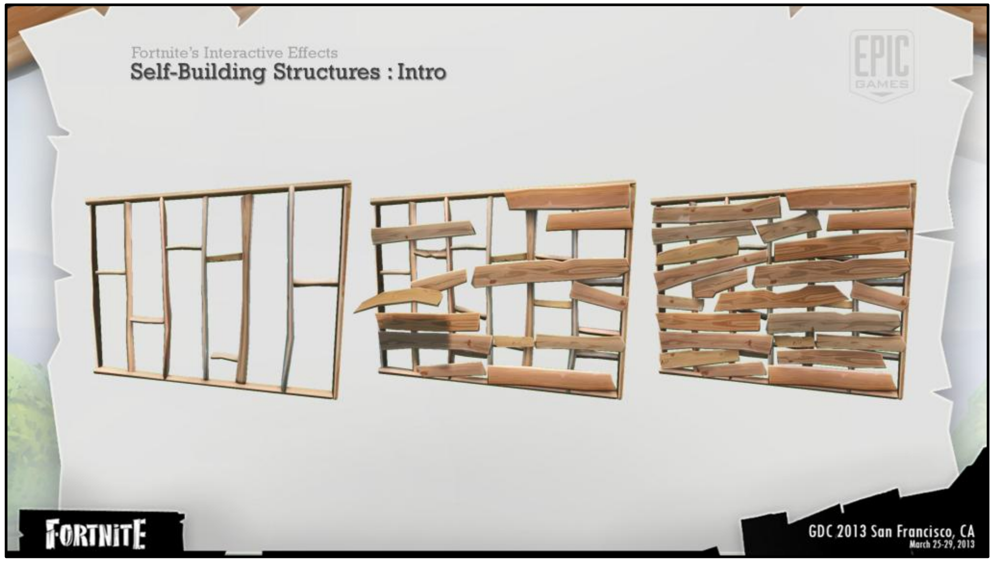
Outline

- Bounce effects
- **Stylized culling**
- Self-building structures
- Conclusion

FORTNITE

GDC 2013 San Francisco, CA
March 25-29, 2013

The final system we will look into is the tech behind the building mechanic

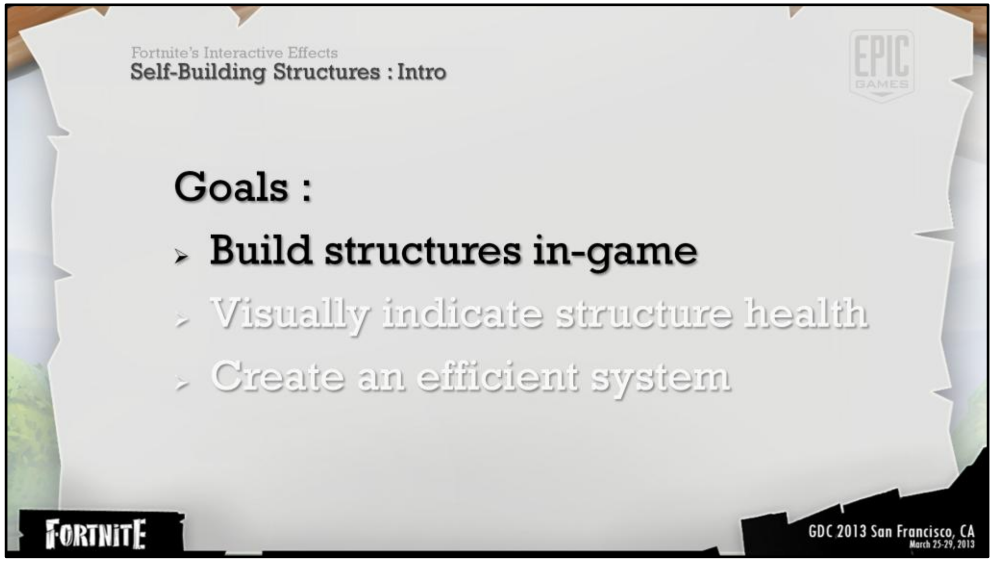

Fortnite's Interactive Effects

Self-Building Structures : Intro

The self animating player pieces represent one of the more complex effects in the game. We felt that it was key to make building a rewarding and entertaining experience for the player. The best way to do that seemed to be to animate the assets into place plank by plank.

We will cover the system at high level overview and then jump into the details

There were a lot of design requirements to be met and we can now step through the problems that were presented and the solutions that we found.



Fortnite's Interactive Effects

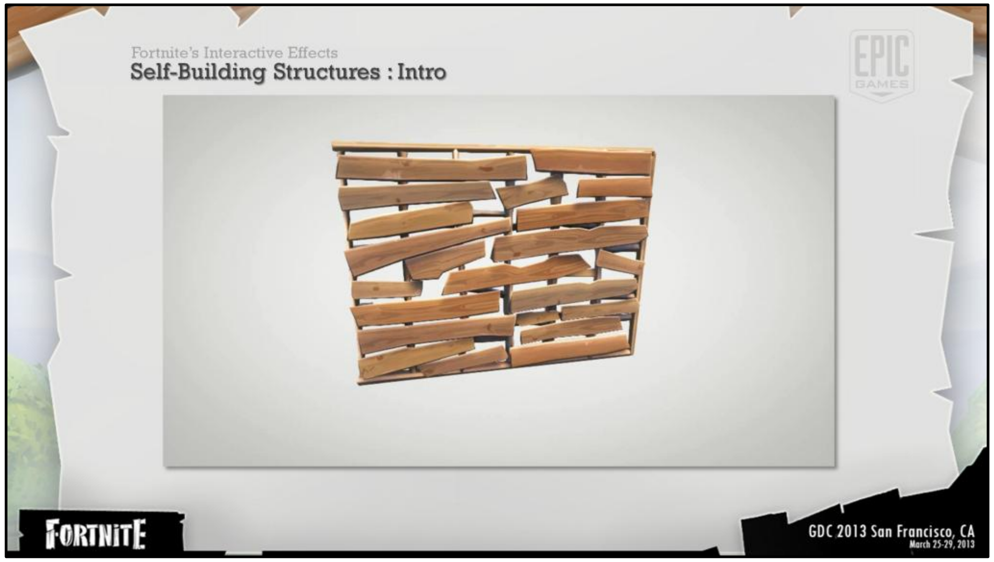
Self-Building Structures : Intro

Goals :

- Build structures in-game
- Visually indicate structure health
- Create an efficient system

FORTNITE

GDC 2013 San Francisco, CA
March 25-29, 2013


Fortnite's Interactive Effects
Self-Building Structures : Intro
FORTNITE
GDC 2013 San Francisco, CA
March 25-29, 2013

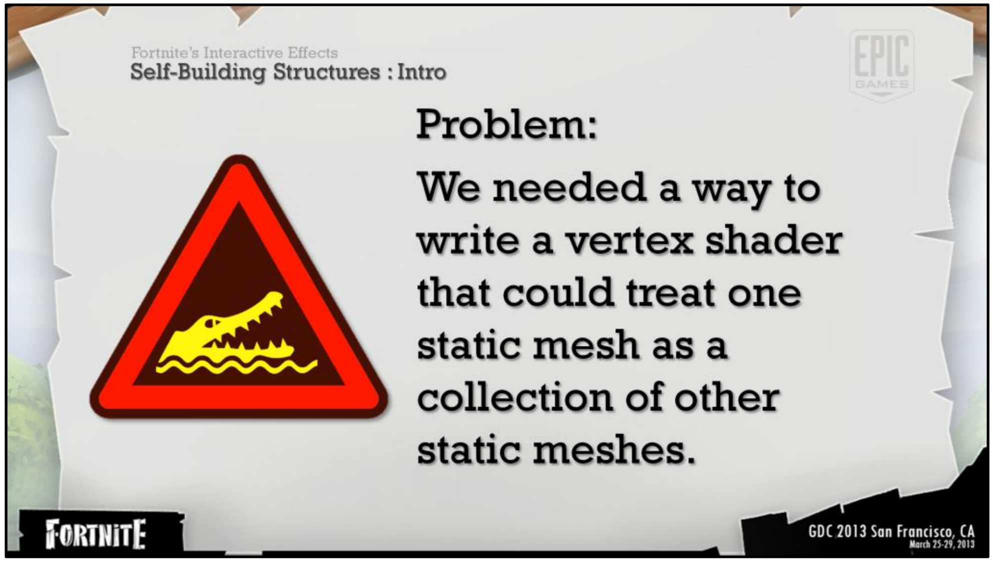
When first evaluating the problem of building walls it appeared that building is much like destroying them in reverse.

We then found that we needed to destroy and recreate structures cheaply in an ordered manner.

An artist initially attempted to animate the walls using skeletal animations but it became quite clear that skeletal animations would be far too costly, in terms of memory and performance.

The next option became obvious at this point.

Vertex shaders



Fortnite's Interactive Effects

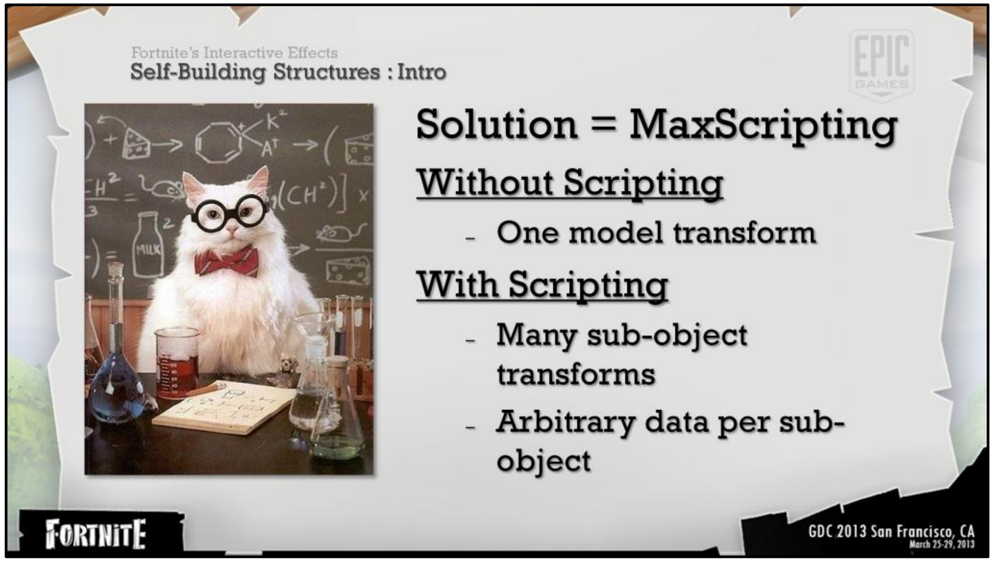
Self-Building Structures : Intro

Problem:

We needed a way to write a vertex shader that could treat one static mesh as a collection of other static meshes.

FORTNITE

GDC 2013 San Francisco, CA
March 25-29, 2013



Fortnite's Interactive Effects

Self-Building Structures : Intro

Solution = MaxScripting

Without Scripting

- One model transform

With Scripting

- Many sub-object transforms
- Arbitrary data per sub-object

FORTNITE

GDC 2013 San Francisco, CA
March 25-29, 2013

(using this as a teaser could be a little awkward)

Vertex shaders give the artist control over the position of every vertex in the model with low overhead costs. Which is exactly what we want but we don't get all of the data that we need right out of the box.

Without scripting shader authors can access the models pivot point, the vertex positions and the vertex normals.

With scripting they can treat one static mesh as many objects. UV channels and vertex colors can be used to store the pivot information among other data.

This video illustrates what a small number of static meshes can look like when model elements are animated uniquely using a vertex shader.

The script is publicly available on the udn website.

- http://udn.epicgames.com/Three/PivotPainterTool.html

The pivot painter script stores objects pivot point location and rotation information in the 3dsmax models vertices. Specifically encoded to decode correctly in Unreal

G channel 1-x

There are also a host of other tools available in the script that you could look at if interested

The stored pivot information can be interpreted as local data which can be transformed into world space in the shader

We have material functions that decode the data in the engine – which can also be explored if one is interested

The pivot painter tool will allow us to import single models while still treating them as many sub objects. This is important because rebuilding walls will require us to write shaders that deal with model elements and not individual vertices

Fortnite's Interactive Effects

Self-Building Structures : Intro

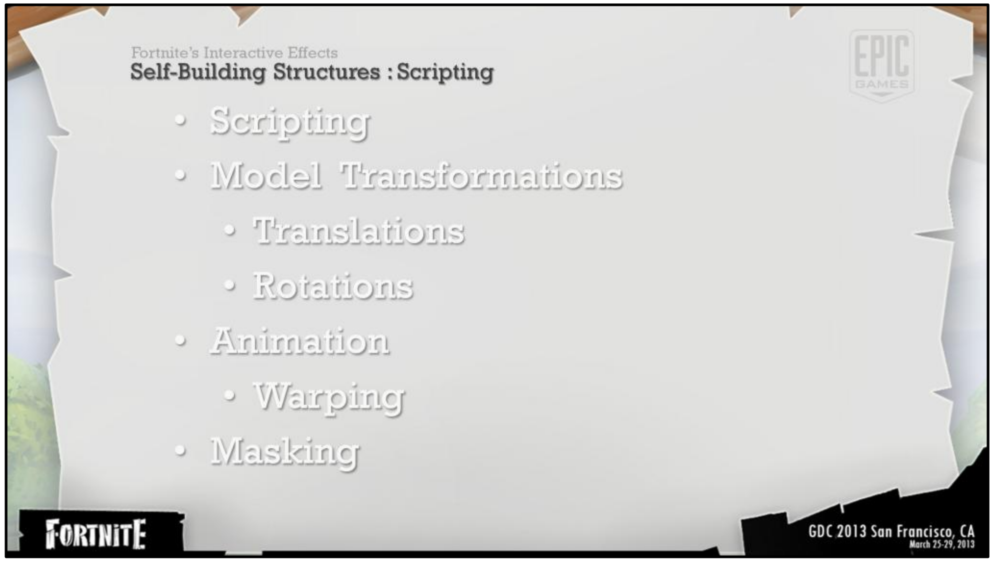
[Pivot_Painter]

http://udn.epicgames.com/Three/PivotPainterTool.html

FORTNITE

GDC 2013 San Francisco, CA
March 25-29, 2013

The script is publicly available on the udn website.



Fortnite's Interactive Effects

Self-Building Structures : Scripting

- Scripting
- Model Transformations
  - Translations
  - Rotations
- Animation
  - Warping
- Masking

FORTNITE

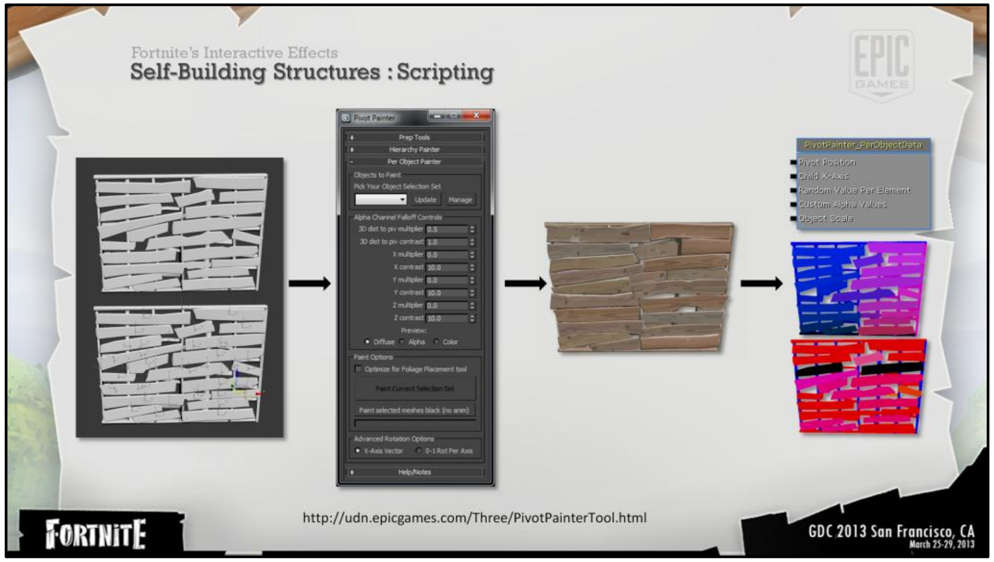
GDC 2013 San Francisco, CA
March 25-29, 2013

So now we know that we need scripting to store access all of the models sub objects transforms before applying any transformations. So we will quickly cover that before moving onto the type of offsets and animations that we will need to fully flesh out the

To cover the entire system quickly
We'll first talk a little bit about scripting
Then what we can do to models after the scripting is finished

After that well discuss the controls needed for gameplay code to control the assets

And finally how we hide models elements that are no longer supposed to exist.



Fortnite's Interactive Effects

Self-Building Structures : Scripting

Objects to Paint
Pick Your Object Selection Set

Alpha Channel Falloff Controls

```
3D dist to piv multiplier  0.5
3D dist to piv contrast    1.0
           X multiplier    0.0
             X contrast    10.0
           Y multiplier    0.0
             Y contrast    10.0
           Z multiplier    0.0
             Z contrast    10.0
```

Preview: [✓ Diffuse] Alpha Color

Paint Options
☐ Optimize for Foliage Placement tool

Advanced Rotation Options
[✓ X-Axis Vector] 0-1 Rot Per Axis

http://udn.epicgames.com/Three/PivotPainterTool.html

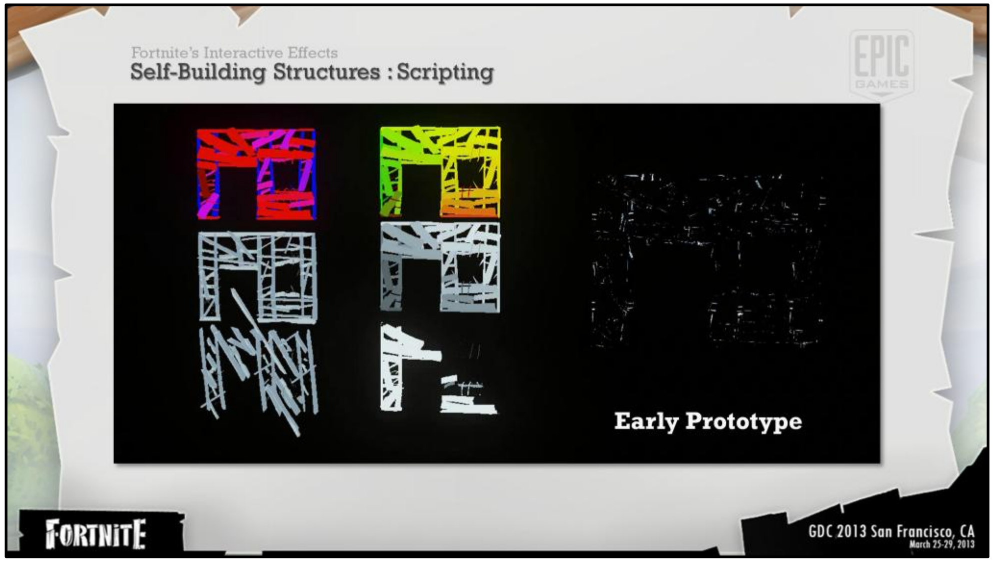
GDC 2013 San Francisco, CA
March 25-29, 2013

First lets explain how the pivot painter tool works.

Explain the material function that decodes the data and show the pivot painter script.

This allows the vertex shader to access objects at a sub object level instead of the vertex level

With this data I knew that we could add offsets to the individual sub-elements over time to the walls to either appear to be breaking or building.


Fortnite's Interactive Effects

Self-Building Structures : Scripting

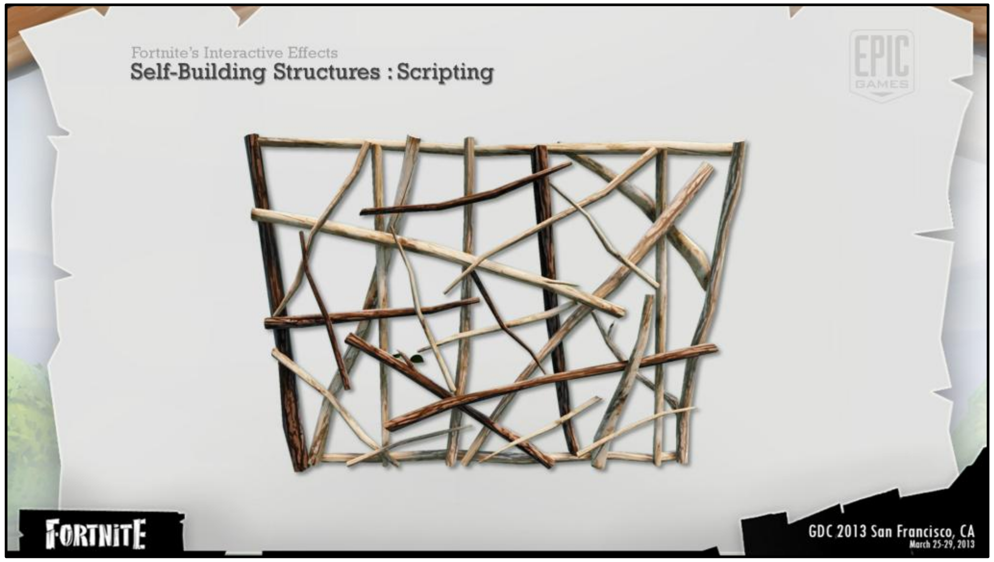
Removed

Using the transformations that we have shown you thus far (and some polish features) we were able to create the animation above. If you notice the boards on the bottom you can see them lighting up as the model on the right animates into place.

This animation is derived from offsetting and clamping the result of a distance calculation from the boards pivot point and an imaginary point in space.

The animated greyscale value is then fed into the rotation amount and the inverse modulates the board translation.

This however did not meet all of our design goals as we wanted to be able to indicate a walls health by how many boards were in place. We also changed the art direction a bit more as well.



Fortnite's Interactive Effects

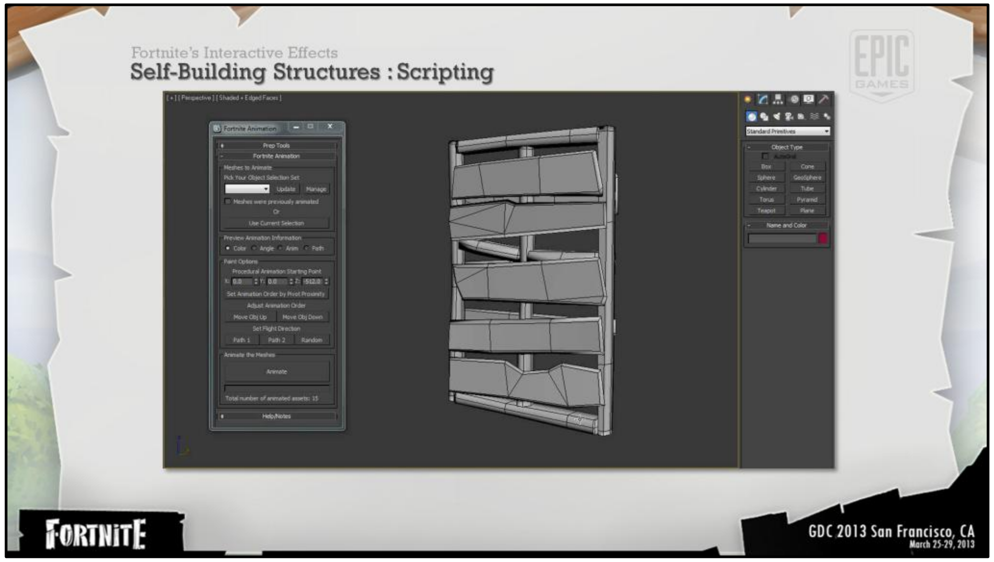
Self-Building Structures : Scripting

Using the pivot painter met most of our design goals for the self building structures but not all of them.

For instance there was no way to specify exactly which order boards flew into place or which direction they flew in from.

The design goals and art direction like shown here meant that we needed control over exactly how many sub objects were in place and the order they were built in.

That would be become an obvious shortcoming on assets like these.



Fortnite's Interactive Effects

Self-Building Structures : Scripting

[+] [Perspective] [Shaded + Edged Faces]

Fortnite Animation

X: 0.0 Y: 0.0 Z: -512.0

Total number of animated assets: 15

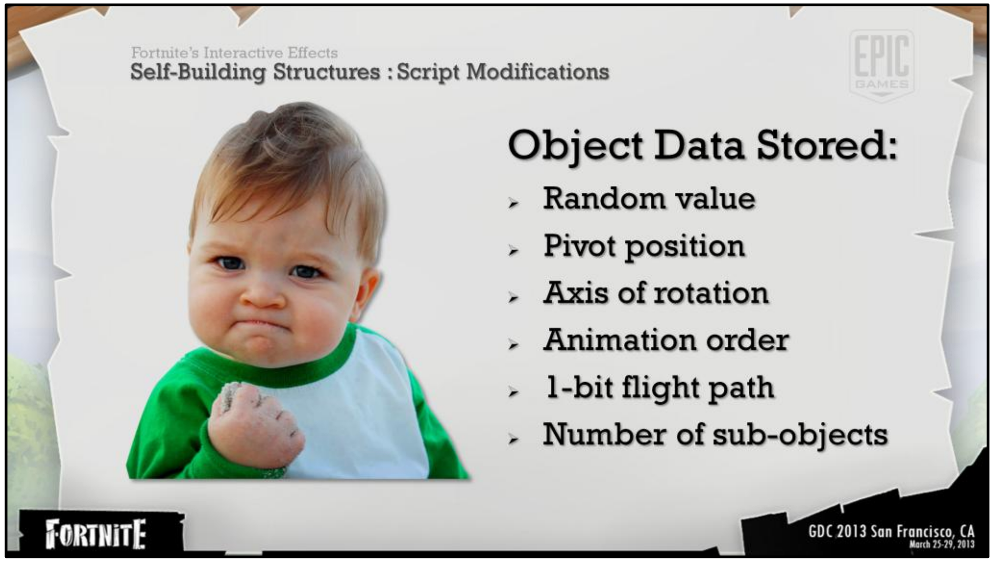
So we created a variant of the pivot painter script.

Learned from the previous tests

Added several controls to control the animation order. Surprisingly, the most used animation approach is the least automated. During the production of the script I evaluated a number of ways to automatically calculated the board order and implemented tools to adjust the stored animation order. In practice I realized that the artists animating the walls preferred to manually pick each asset in, one at a time, to create the animation order. Selecting assets one at a time automatically generates an ordered selection array. The script takes the elements index in the selection and stores the number in the models uvs to later be read by the vertex shader.

Additional features include storing one bit information to control the sub-objects flight paths and a random value per board.

So finally we have a solution to animate



Fortnite's Interactive Effects

Self-Building Structures : Script Modifications

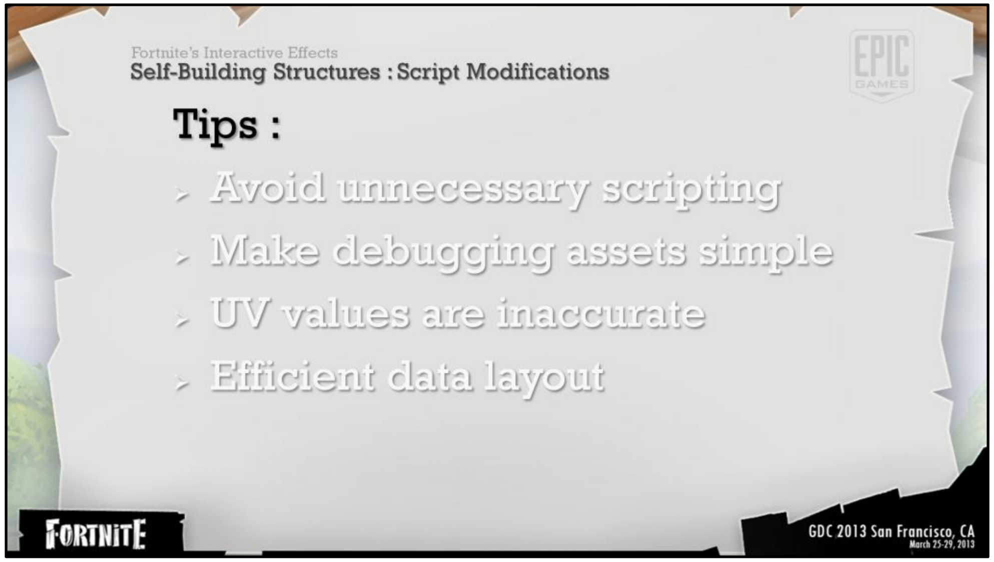
Object Data Stored:

- Random value
- Pivot position
- Axis of rotation
- Animation order
- 1-bit flight path
- Number of sub-objects

FORTNITE

GDC 2013 San Francisco, CA
March 25-29, 2013

Success! :D
We now have all of the data that we need to animate the walls. Lets look into how the actual animation is handled.



Fortnite's Interactive Effects

Self-Building Structures : Script Modifications

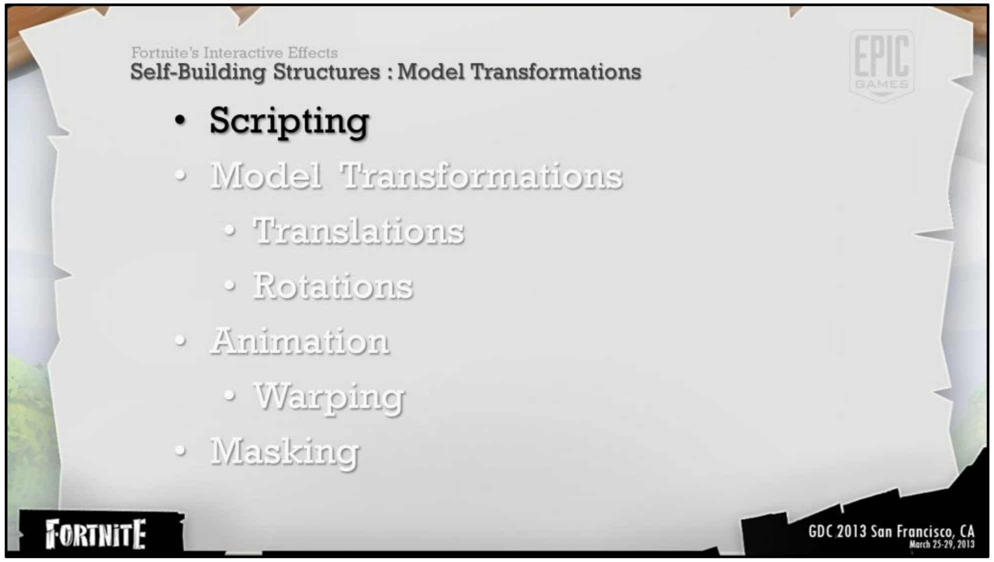
Tips :

- Avoid unnecessary scripting
- Make debugging assets simple
- UV values are inaccurate
- Efficient data layout

EPIC GAMES
FORTNITE
GDC 2013 San Francisco, CA
March 25-29, 2013

For a more in depth look at the data format please see the pivot painter script which is located online on the UDN website.

-UV values are very inaccurate
    -Store values in whole number increments and offset their value by .5
    -Use ceil function in the shader to ensure that every board is assigned an Int value when sampled
-Avoid unnecessary scripting
    -Use the ordered array automatically created by 3DS Max ($selection)
    -Speak with other artists for preferred work methods
-Debug
    -Store data in the objects user variables while writing info to the vertex colors
    -Make the data viewable with view modes
-Efficiency
    -When possible pack one bit data along with other information to save on memory size (multiply a positive value by 1 or -1 before storing it to add a piece of one bit data to the storage channel)



Fortnite's Interactive Effects

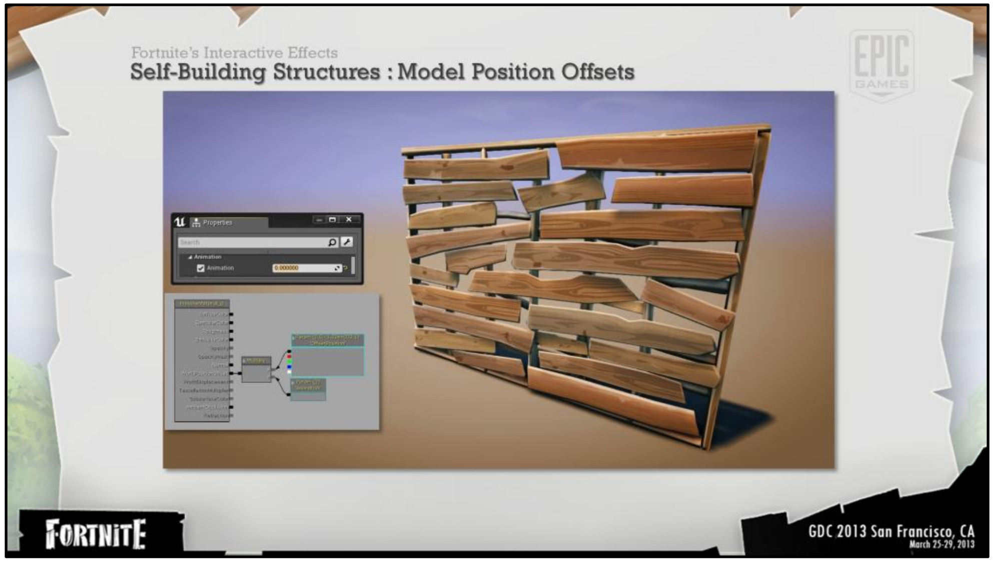
Self-Building Structures : Model Transformations

- Scripting
- Model Transformations
  - Translations
  - Rotations
- Animation
  - Warping
- Masking

FORTNITE

GDC 2013 San Francisco, CA
March 25-29, 2013

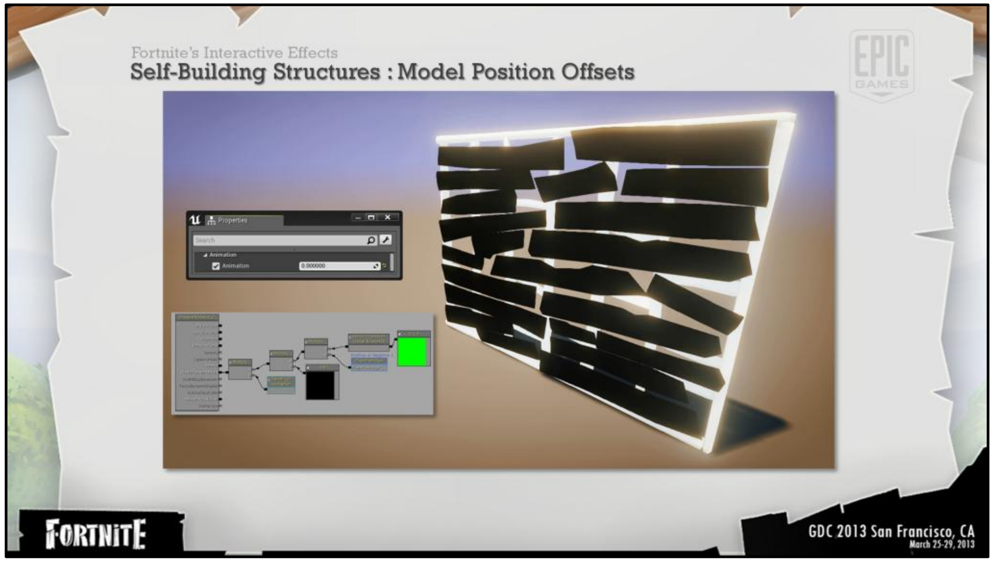
Now that the data is available lets go into what is nessesary to make the boards to appear to move.



Fortnite's Interactive Effects

Self-Building Structures : Model Position Offsets

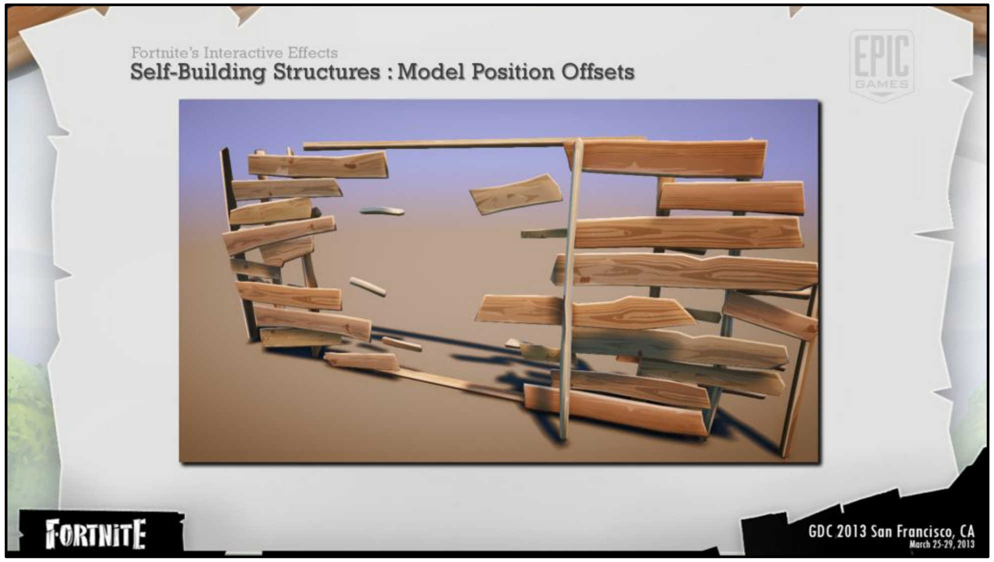
(the animation multiplier will be replaced with board specific animation)



Fortnite's Interactive Effects

Self-Building Structures : Model Position Offsets

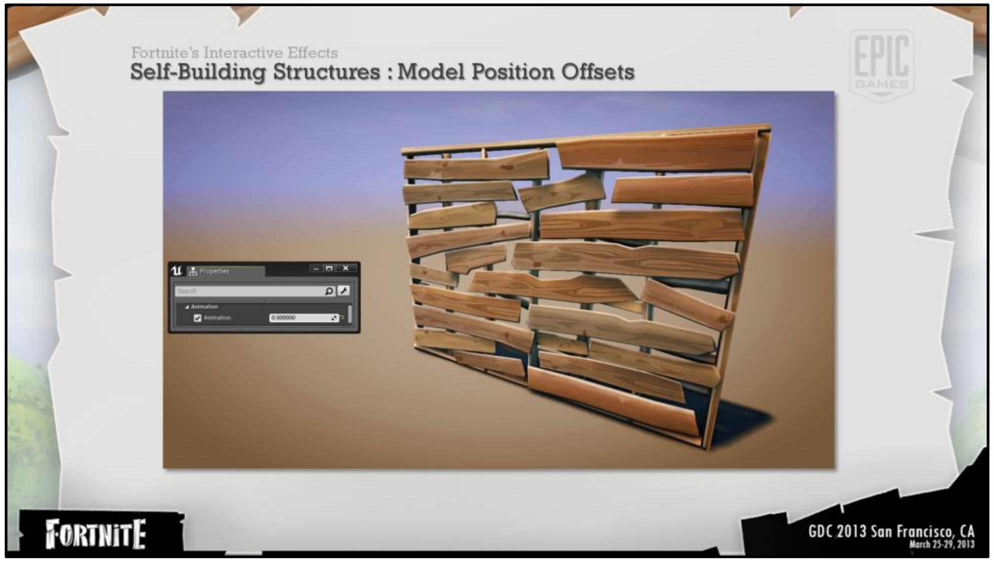
Import the binary value from the script and use it as a negative or positive sign for a directional offset multiplied by another arbitrary offset.



Fortnite's Interactive Effects

Self-Building Structures : Model Position Offsets

Subtract the board pivot point from the object center
Remove the z component
Normalize
Multiply by arbitrary value



Fortnite's Interactive Effects

Self-Building Structures : Model Position Offsets

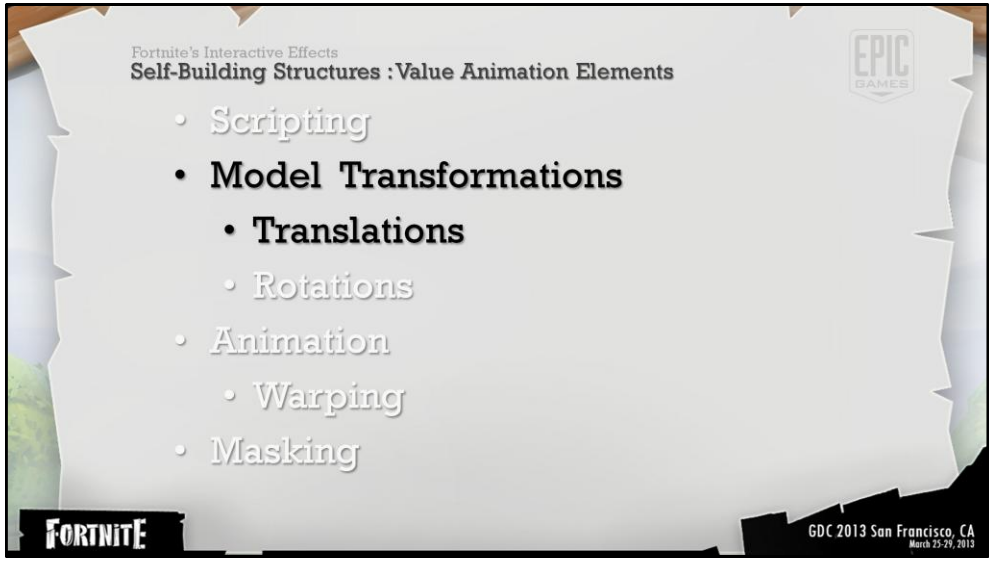
Properties

Animation

Animation 0.000000

GDC 2013 San Francisco, CA
March 25-29, 2013

Add the offsets together

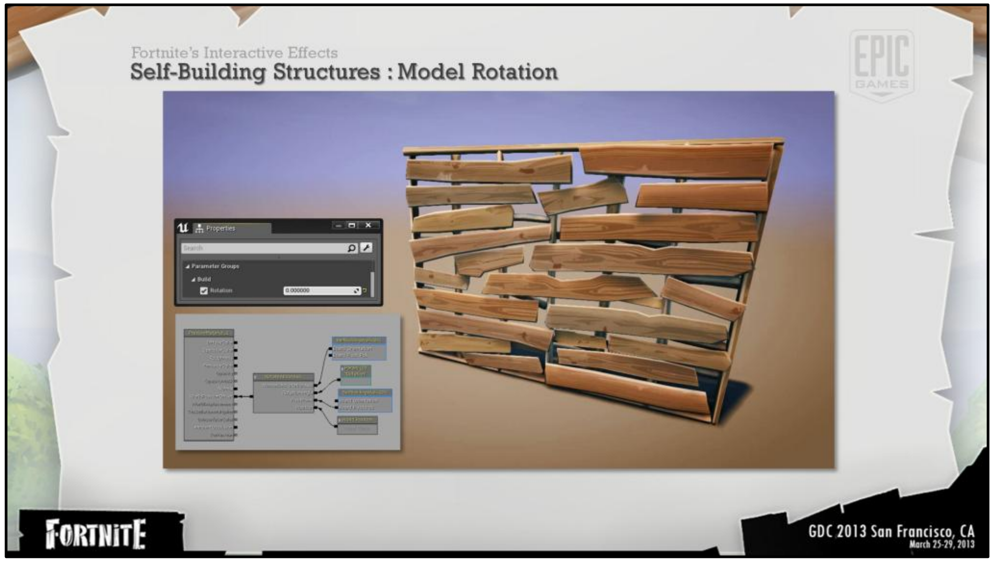


Fortnite's Interactive Effects

Self-Building Structures : Value Animation Elements

- Scripting
- Model Transformations
  - Translations
  - Rotations
- Animation
  - Warping
- Masking

EPIC GAMES
FORTNITE
GDC 2013 San Francisco, CA
March 25-29, 2013

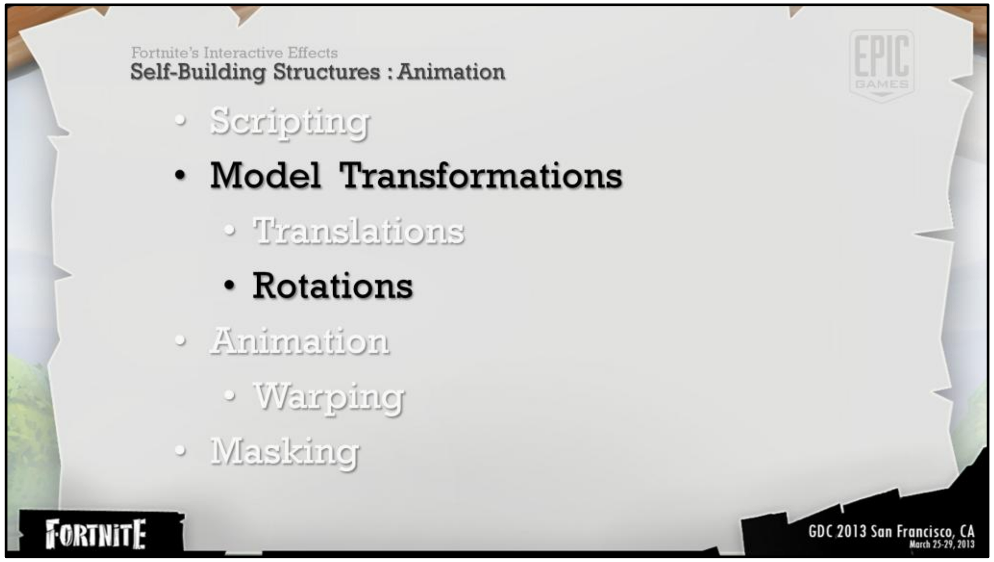

Fortnite's Interactive Effects
Self-Building Structures : Model Rotation
Properties
Parameter Groups
Build
Rotation
0.000000
FORTNITE
GDC 2013 San Francisco, CA
March 25-29, 2013

Discuss rotation and animation curves.

Ease in and ease out

Given the model data available (rotation axis and and pivot position) we can easily rotate each asset along their own axes using a native node in the material editor called rotate about axis.

We can use the animation values from our previous steps to control the number of rotations that occur as the boards fly into position.



Fortnite's Interactive Effects

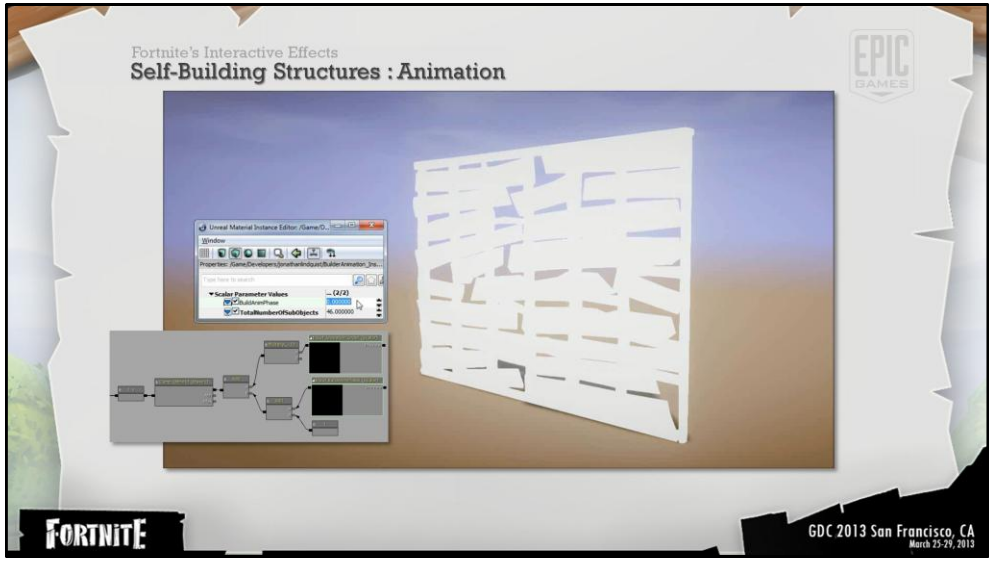
Self-Building Structures : Animation

- Scripting
- Model Transformations
  - Translations
  - Rotations
- Animation
  - Warping
- Masking

FORTNITE

GDC 2013 San Francisco, CA
March 25-29, 2013

Gameplay driven animation parameters and then
Then how to fake drag/secondary motion on the models outer vertices.



Fortnite's Interactive Effects

Self-Building Structures : Animation

Unreal Material Instance Editor: /Game/D...

Properties: /Game/Developers/jonathanlindquist/BuilderAnimation_Ins...

Scalar Parameter Values	-- (2/2)
BuildAnimPhase	0.000000
TotalNumberOfSubObjects	46.000000

FORTNITE

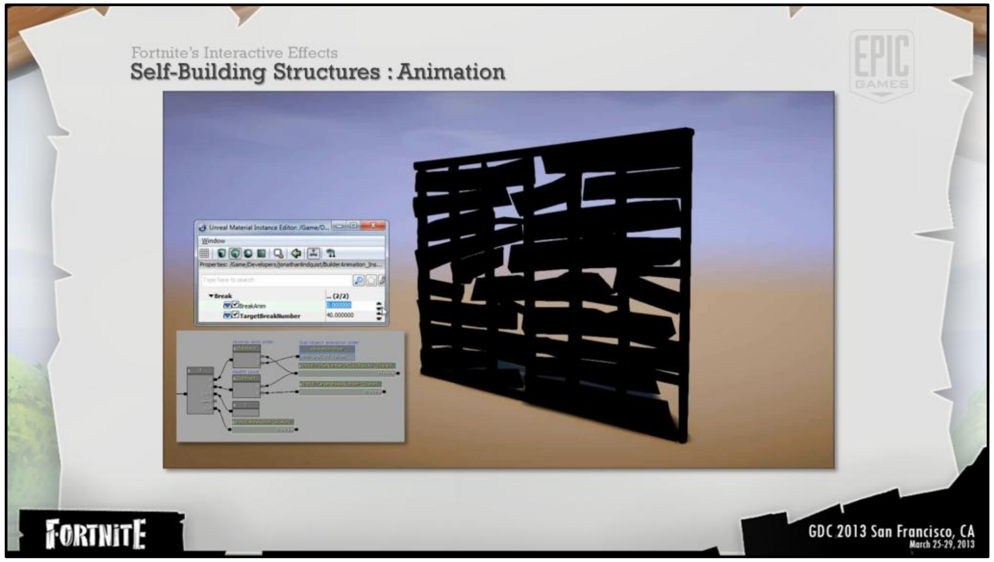
GDC 2013 San Francisco, CA
March 25-29, 2013

Taking the data packed into the model from the previous steps we can create an animation that animates one board at a time from white to black.

To do this we reference the build order number stored in each sub-object model. This wall has 46 wooden planks. Each plank of wood in this asset is then assigned a specific number between 1-46.

To animated the boards in order we simply subtract a value up to 46 and clamp the result. This will make the assets ramp from 1 to 0 one at a time.

This animation value is then multiplied against a 3d model offset.


Fortnite's Interactive Effects

Self-Building Structures : Animation

Unreal Material Instance Editor: /Game/D...

Properties: /Game/Developers/jonathanlindquist/BuilderAnimation_Ins...

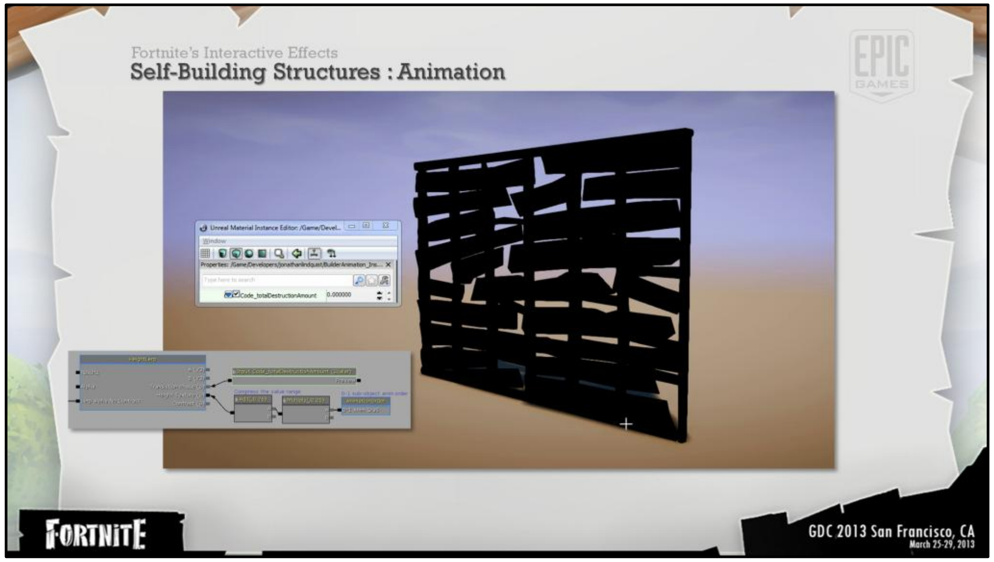
```
▼Break                  ... (2/2)
    BreakAnim           0.000000
    TargetBreakNumber   40.000000
```

GDC 2013 San Francisco, CA
March 25-29, 2013

Large monsters can break off multiple planks of wood at a time.

To do this we do a comparison between the boards animation order and the number of boards we want to remain.

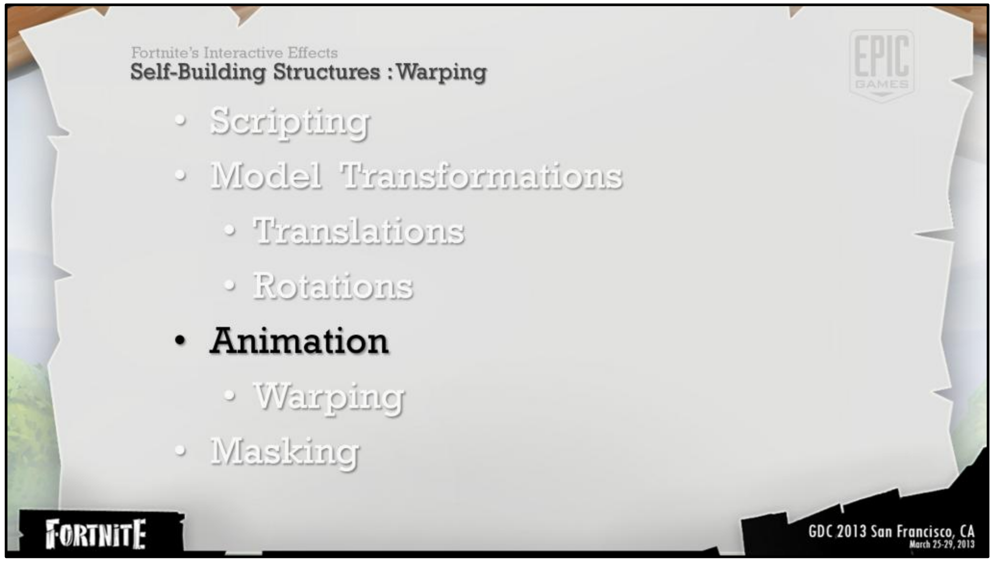
If the boards animation value is greater or equal to the target board number then add another variable will pass through on the selected boards. This variable is called the “break anim” parameter.


Fortnite's Interactive Effects
Self-Building Structures : Animation
EPIC GAMES
FORTNITE
GDC 2013 San Francisco, CA
March 25-29, 2013

When a wall receives enough damage to be completely destroyed a value is added to the entire mesh at once.

For an added amount of polish
Multiply the build list by .25 then add .75 and add in the break amount to make offset the order that the boards falloff. (25 percent variance of animation across surface)

Awesome our animation gradients work! Lets add them together and clamp the result to produce the animation that will act as a base for the effect.



Fortnite's Interactive Effects

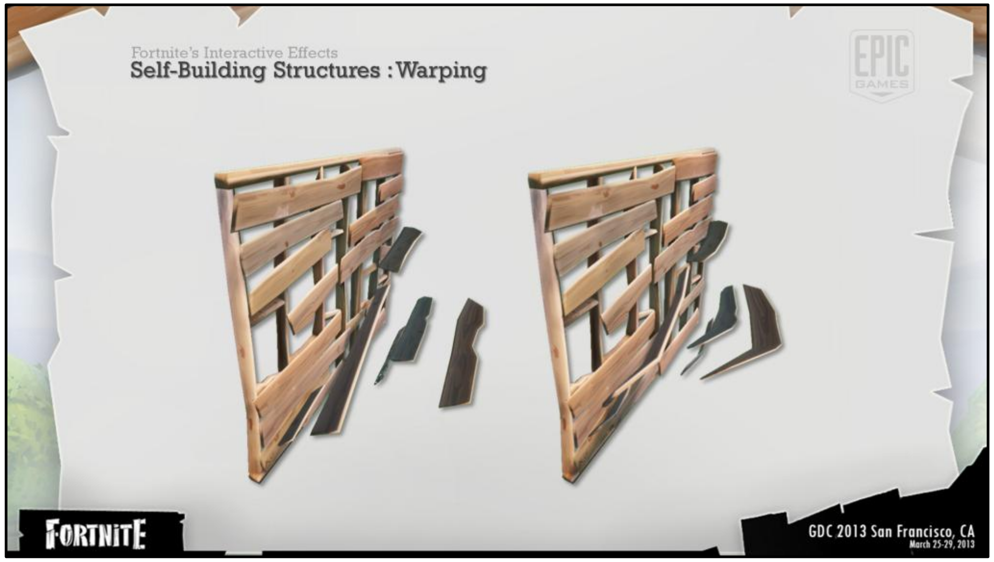
Self-Building Structures : Warping

- Scripting
- Model Transformations
  - Translations
  - Rotations
- Animation
  - Warping
- Masking

FORTNITE

GDC 2013 San Francisco, CA
March 25-29, 2013

how to fake drag/secondary motion on the models outer vertices.

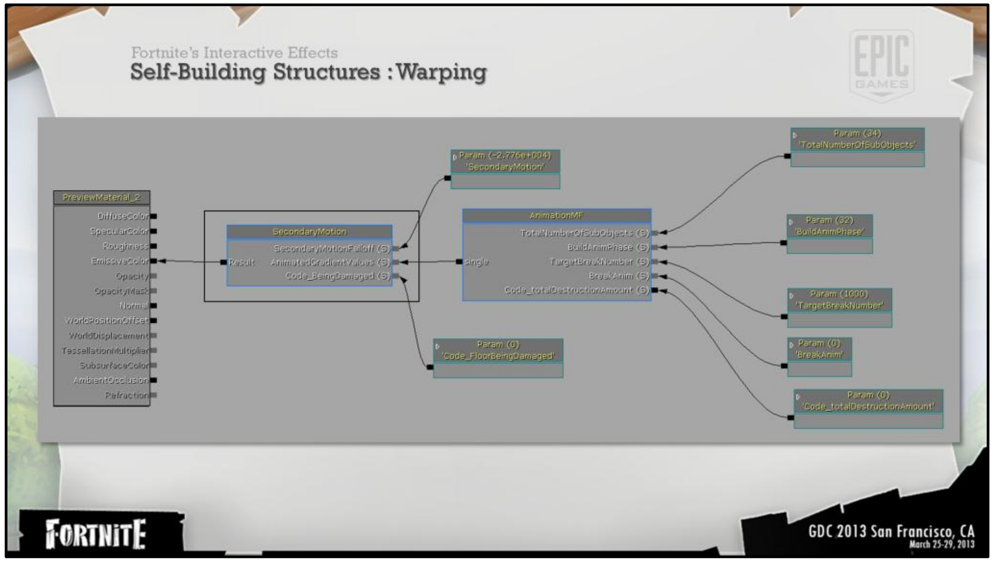


Fortnite's Interactive Effects

Self-Building Structures : Warping

FORTNITE

GDC 2013 San Francisco, CA
March 25-29, 2013



Fortnite's Interactive Effects

Self-Building Structures : Warping

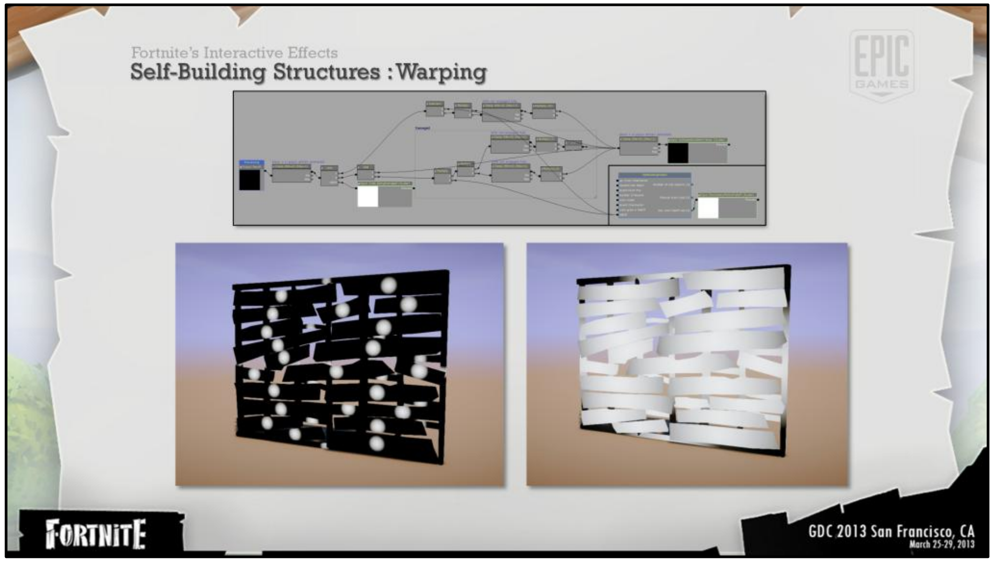
FORTNITE

GDC 2013 San Francisco, CA
March 25-29, 2013

Now that the gradient animations are complete we can offset the values to create the appearance of secondary motion.

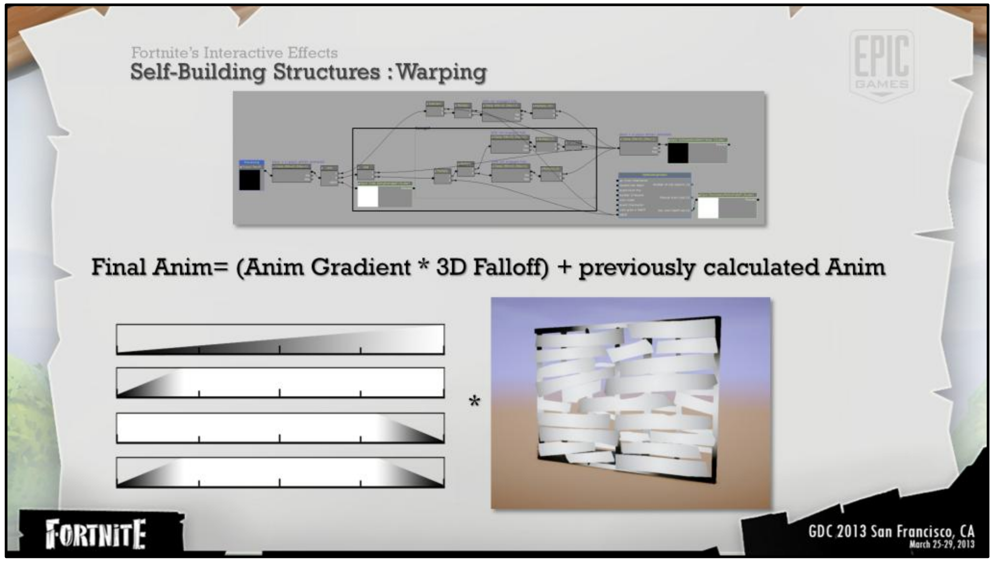
When possible make sure to break apart functionality as much as possible. Separating the secondary motion from all of the animation code makes it far simpler.

Lets jump into this node and see what goes into it.


Fortnite's Interactive Effects
Self-Building Structures : Warping
EPIC GAMES
FORTNITE
GDC 2013 San Francisco, CA
March 25-29, 2013

It's fairly similar to the other approaches we've been using.

Using a distance calculation from the sub from the sub object pivot to the vertex position we can create an outward gradient.
(this can be optimized by storing the distance information in the pivot)

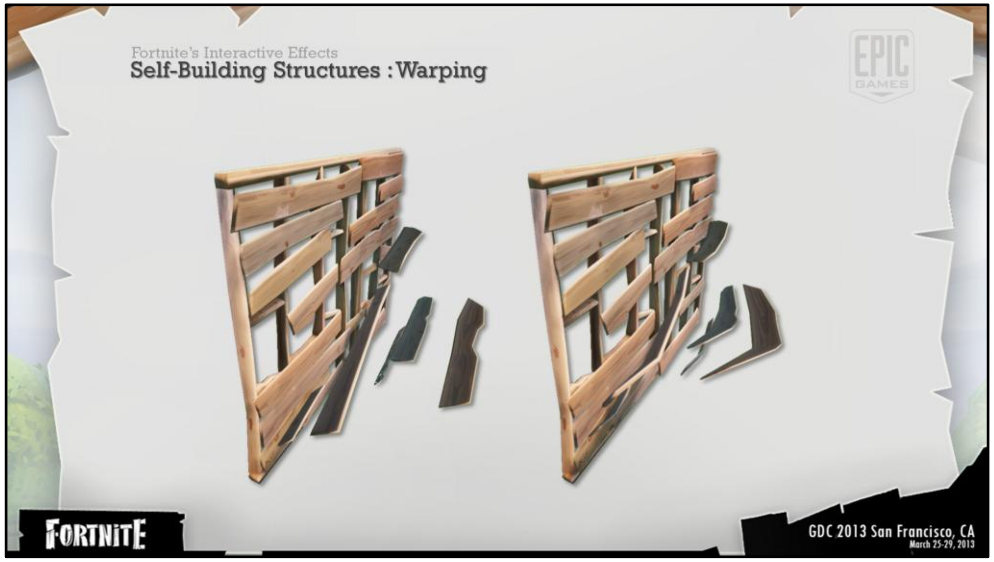

Fortnite's Interactive Effects

Self-Building Structures : Warping

Final Anim= (Anim Gradient * 3D Falloff) + previously calculated Anim

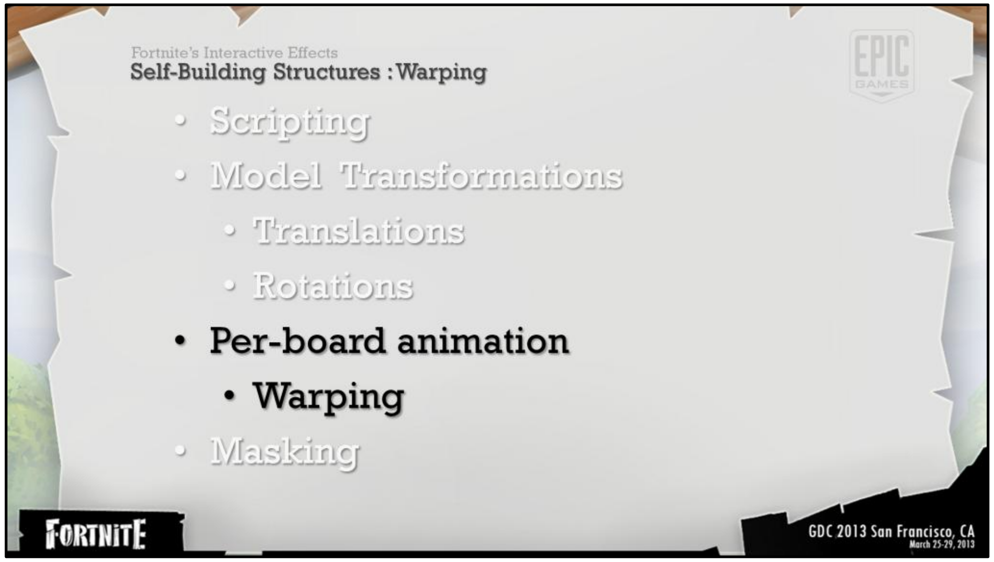
20% of front and back end of the animation is masked off using the calculations on the left.

Drag is only present at the beginning and the end of the animation timeline.


Fortnite's Interactive Effects

Self-Building Structures : Warping

End result of drag



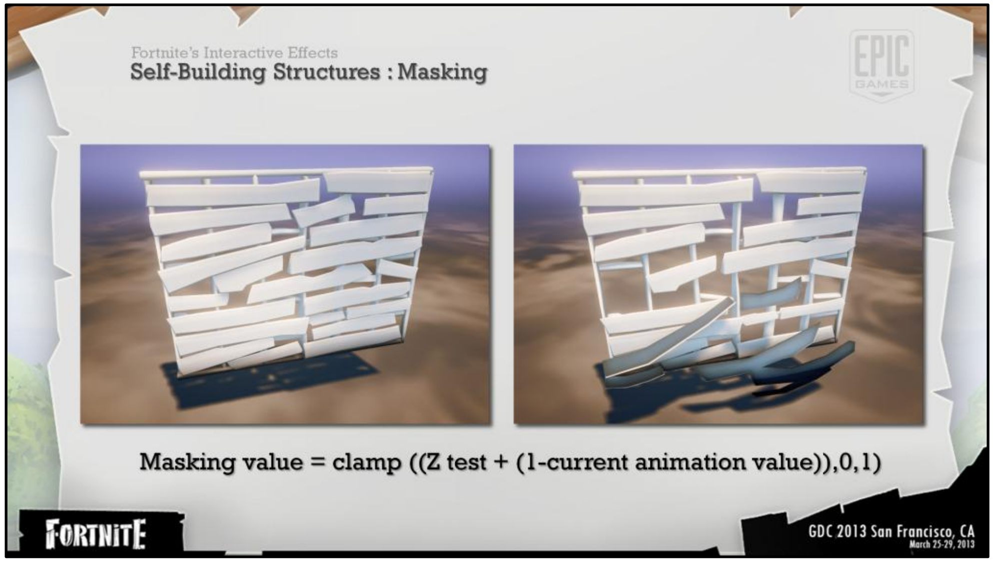
Fortnite's Interactive Effects

Self-Building Structures : Warping

- Scripting
- Model Transformations
  - Translations
  - Rotations
- Per-board animation
  - Warping
- Masking

FORTNITE
GDC 2013 San Francisco, CA
March 25-29, 2013

What happens to mesh elements that should not exist.



Fortnite's Interactive Effects

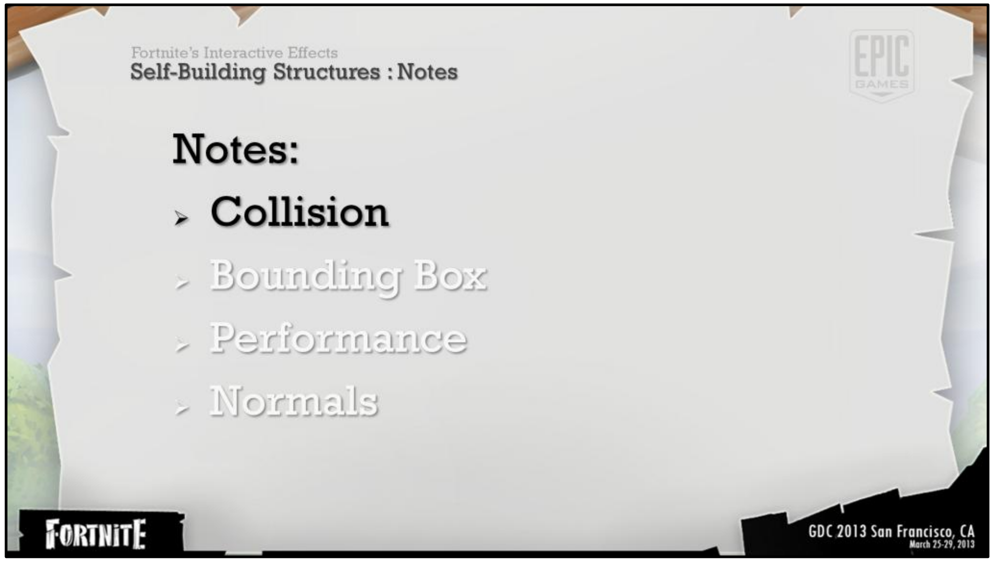
Self-Building Structures : Masking

Masking value = clamp ((Z test + (1-current animation value)),0,1)

They stay around ! But are masked and scaled down to reduce overdraw.

The masking value is derived from the math above. The mask value is run through a thresholding process offset by a grayscale texture.

To reduce overdraw we scale down the boards that are fully animated with the vertex shader using some of the techniques we discussed earlier.


Fortnite's Interactive Effects
Self-Building Structures : Notes

Notes:
- Collision
- Bounding Box
- Performance
- Normals

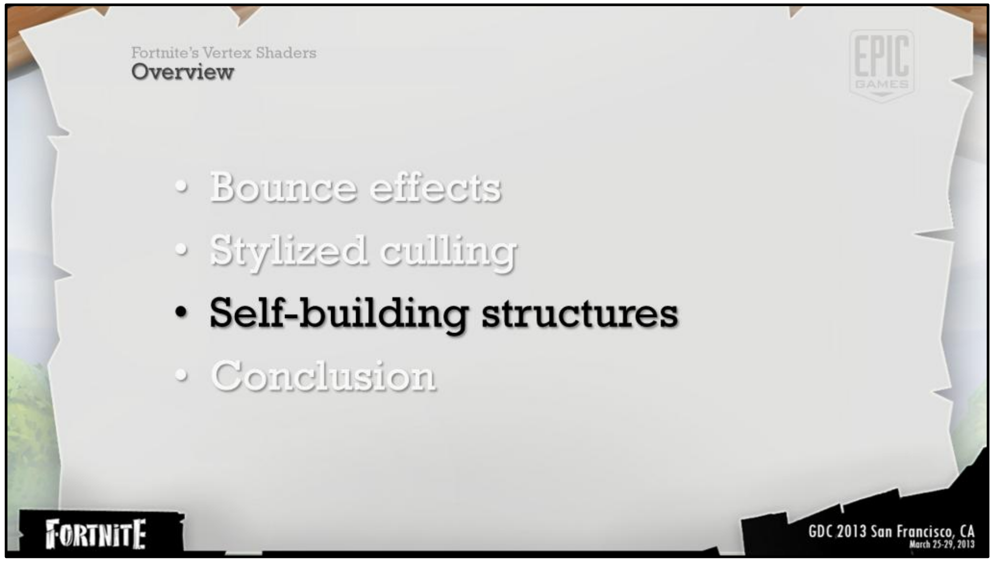
FORTNITE
GDC 2013 San Francisco, CA
March 25-29, 2013

There are a few potential shortcomings of the techniques

-Per poly collision still occur with the static mesh as if it hasn't been animated at all. The deformation occurs on the gpu and is therefore purely visual. This is not a problem if the objects do not need to collide with bullets or if the visuals should not effect the hit surfaces.

- Bounding box
  -There is a problem with moving the elements far outside of their bounding boxes. If the bounding box is no longer in view the object will be culled from view. That includes polygons that have been moved outside of their bounding box. There are several solutions to the problem
    - Don't move polygons outside of the bounding box
    - Create verts at the extents of the animation boundaries to set the bounding box size
    - Programmatically scale the bounding box size.
    - Create animations that move quickly enough to make the problem unnoticeable

- Performance
    - Processed a model adds 18% of file size to the asset.
    - Animations do not require any memory
    - The shader math to animate the boards only needs to run when the animation is taking place.
- Normals
    - The assets normals do not automatically update but it is possible. It is possible to modify the normals manually to match the animation but certain animations are slightly expensive to replicate.

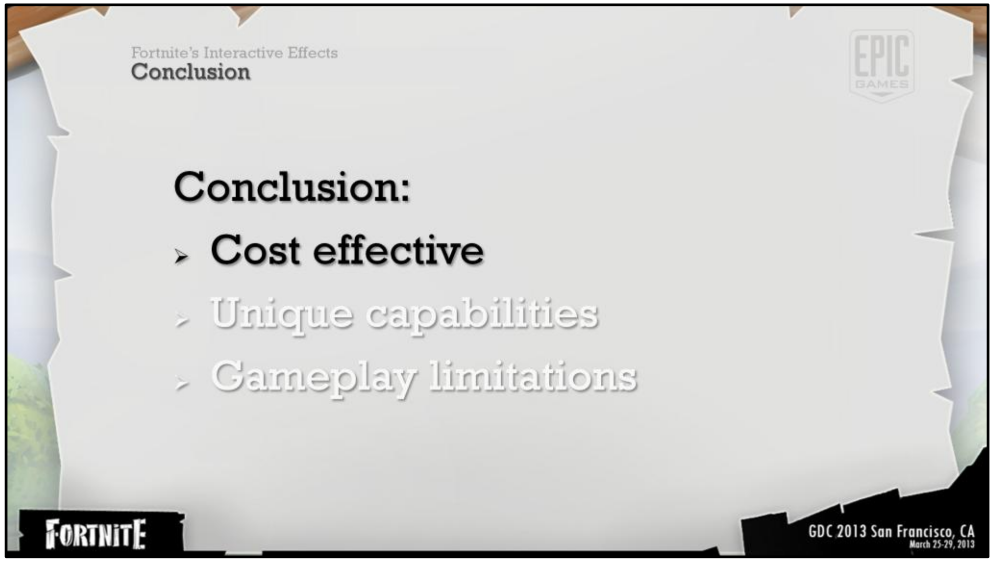


Fortnite's Vertex Shaders

Overview

- Bounce effects
- Stylized culling
- **Self-building structures**
- Conclusion

FORTNITE
GDC 2013 San Francisco, CA
March 25-29, 2013

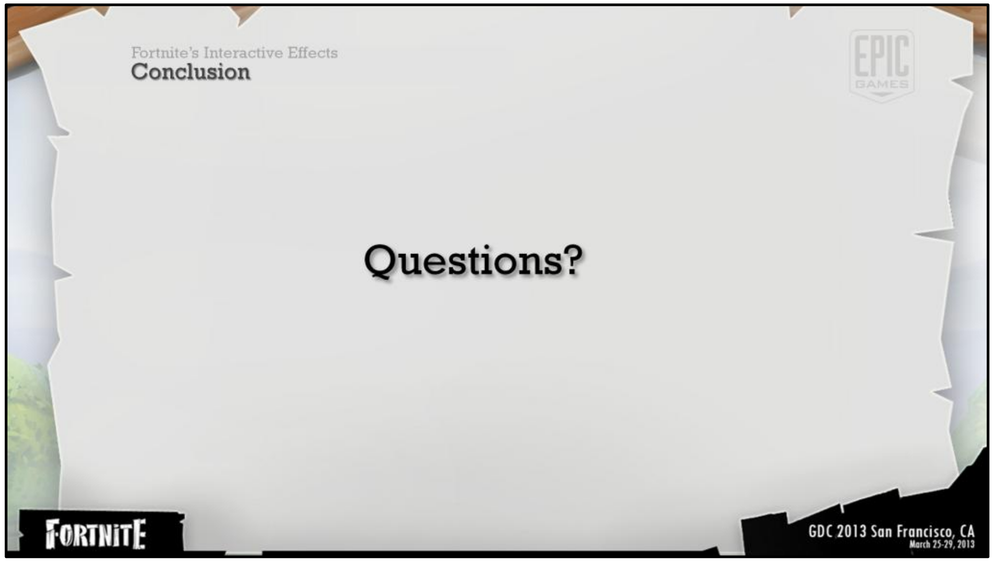


Fortnite's Interactive Effects
Conclusion

Conclusion:
- Cost effective
- Unique capabilities
- Gameplay limitations

FORTNITE
GDC 2013 San Francisco, CA
March 25-29, 2013

- Vertex shaders provide a very cost efficient method for creating procedural animation
- They can achieve effects that would be otherwise impossible
- All of the effects that we've seen have been reused across many assets
- These techniques offer unprecedented levels of control over mesh sub-objects. Without involving the cpu we can dramatically increase the granularity of our effects.



Fortnite's Interactive Effects

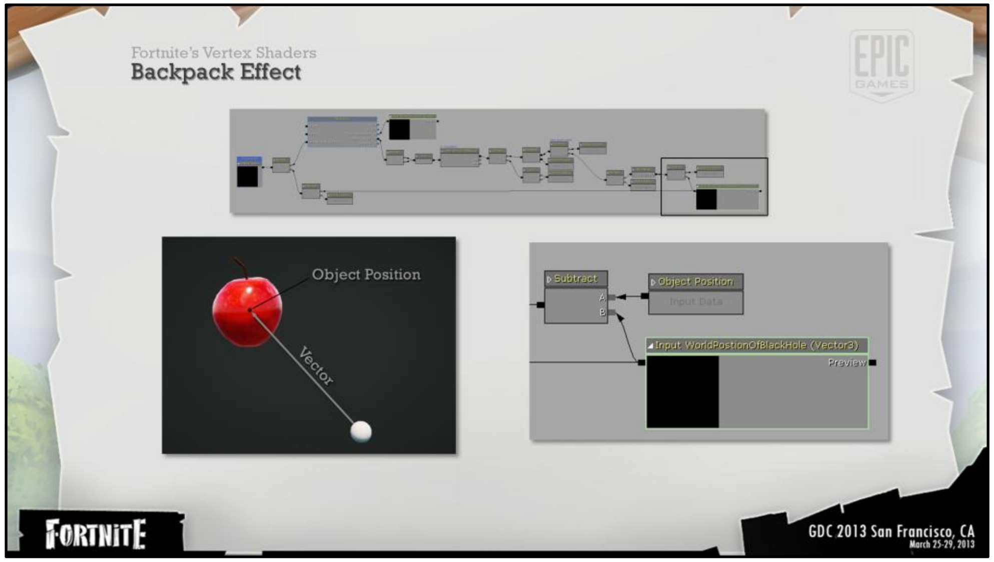
Conclusion

Questions?

FORTNITE

GDC 2013 San Francisco, CA
March 25-29, 2013

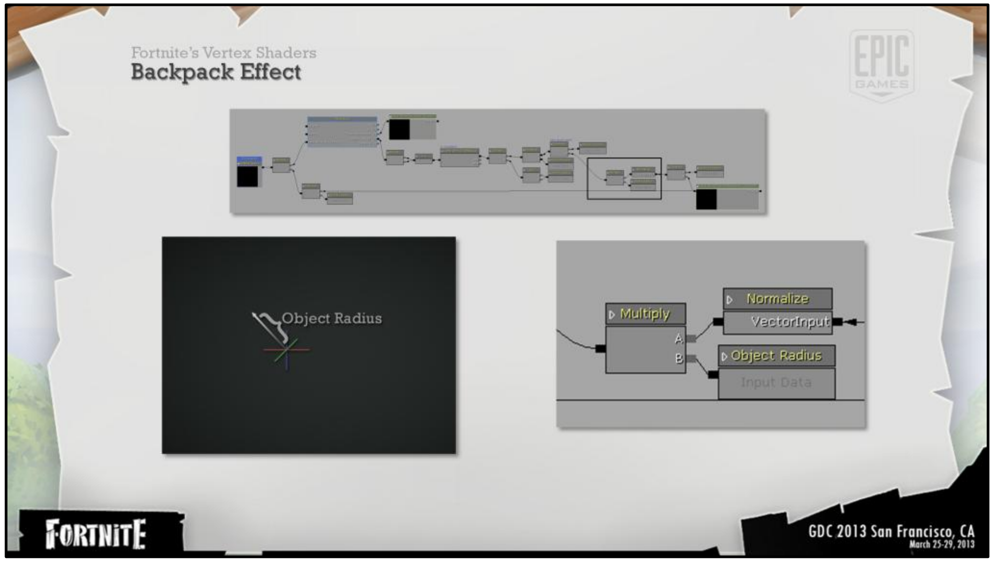
JONES
FORTNITE
GDC 2013 San Francisco, CA
March 25-29, 2013



To continue on exploring vertex shaders throughout the world we tried a few more things

To show items entering the players inventory we went with a stylized shrinking effect. The approach chose works on props of any size and doesn't require custom data.

The white sphere represents the opening in the backpack.



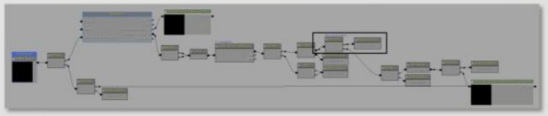
Fortnite's Vertex Shaders

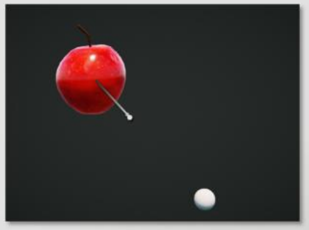
Backpack Effect

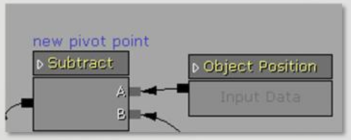
GDC 2013 San Francisco, CA
March 25-29, 2013

Fortnite's Vertex Shaders

Backpack Effect

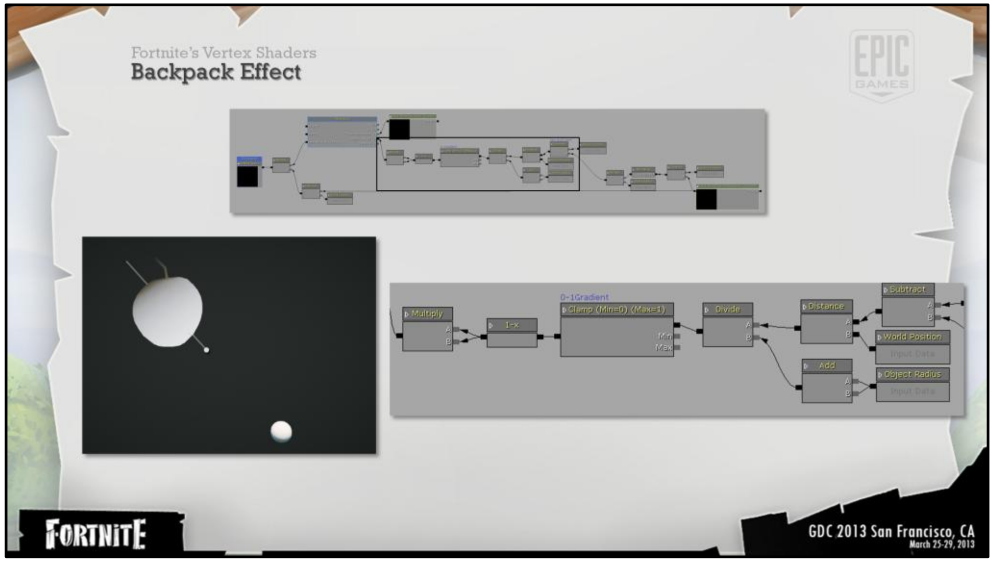






FORTNITE

GDC 2013 San Francisco, CA
March 25-29, 2013


Fortnite's Vertex Shaders

Backpack Effect

GDC 2013 San Francisco, CA
March 25-29, 2013

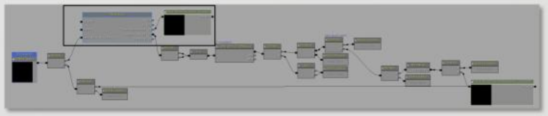
We then find the distance between that point in space and each vertices on the model.

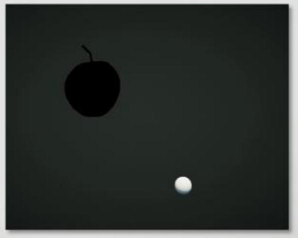
And normalize the value between 0-1 by dividing the distance by the diameter.

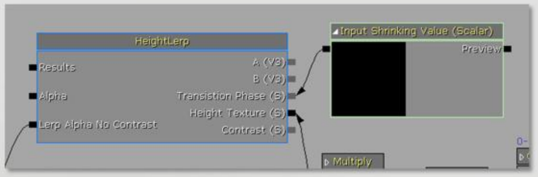
We now have a 0-1 3D gradient that spans the diameter of the object. It’s also oriented toward the target destination.

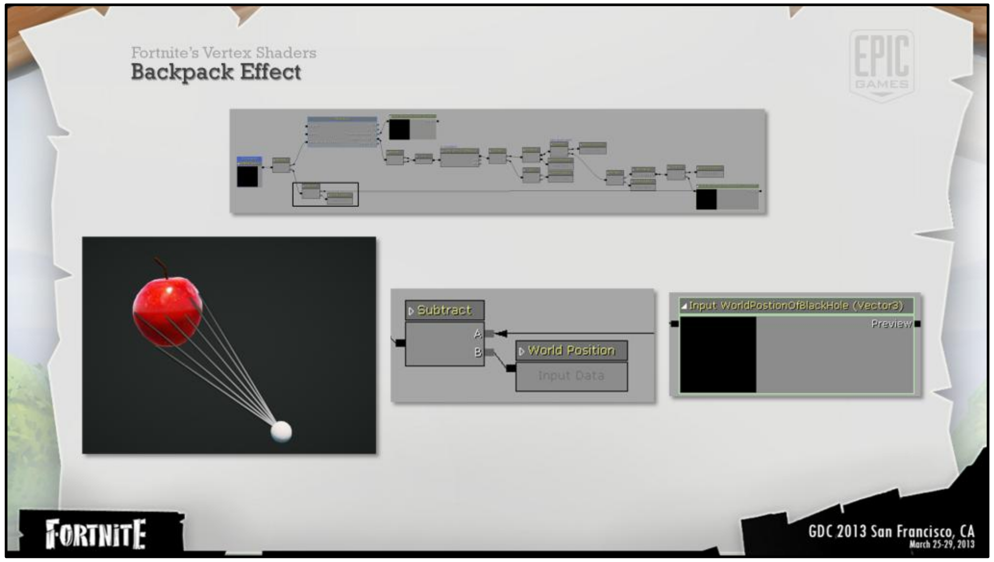
Fortnite's Vertex Shaders

Backpack Effect

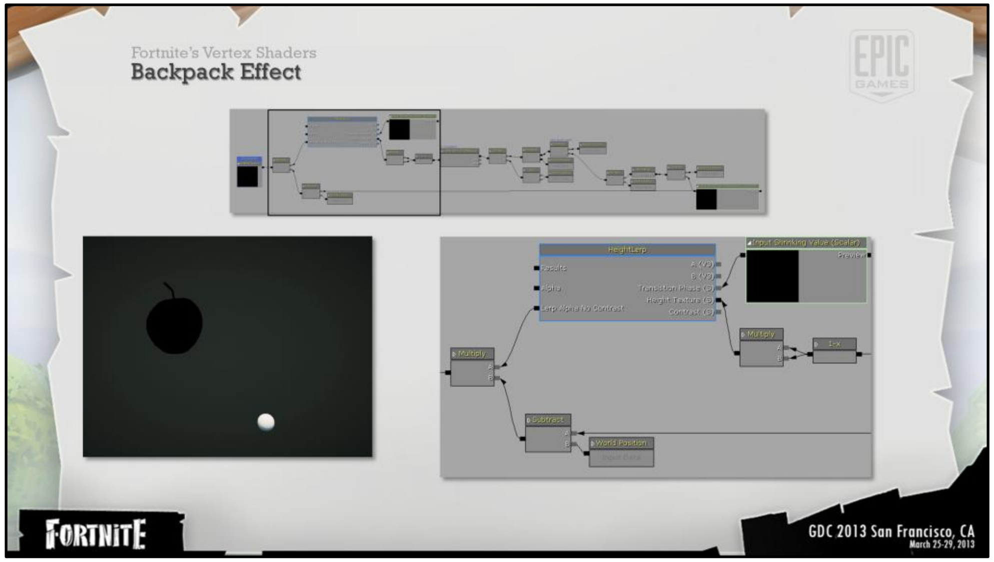







Fortnite's Vertex Shaders
Backpack Effect
Subtract
A
B
World Position
Input Data
Input WorldPostionOfBlackHole (Vector3)
Preview
FORTNITE
GDC 2013 San Francisco, CA
March 25-29, 2013

Per vertex vectors
Length=distance to the target.


Fortnite's Vertex Shaders

Backpack Effect

GDC 2013 San Francisco, CA
March 25-29, 2013

The shrinking value is controlled via a curve asset.