GDC

Autonomy in Realtime Effects: Artist Driven Tools and Techniques for Large Scale Production

Bill Kladis
Senior VFX Artist, Epic Games (Fortnite)

EPIC GAMES

GAME DEVELOPERS CONFERENCE® | MARCH 19–23, 2018 | EXPO: MARCH 21–23, 2018 #GDC18

UBM

Automation & Tools in VFX

- Motivations and recurring themes for doing this talk:
  - Ambitious project.
  - Small team size.
  - Lack of desired tech and features.
  - Autonomy.
  - **Math and tech concepts are critical for the future of FX in games.**

Automation & Tools in VFX

- FX Artists embracing more technical & mathematical concepts.
- Use scripting languages to:
  - Deal with ways to automate processes for large-scale problems.
  - Write your own tools to fill void of technical deficits.
- All with simple to follow concepts and supporting visual examples using Fortnite as a backdrop..

What is Fortnite?

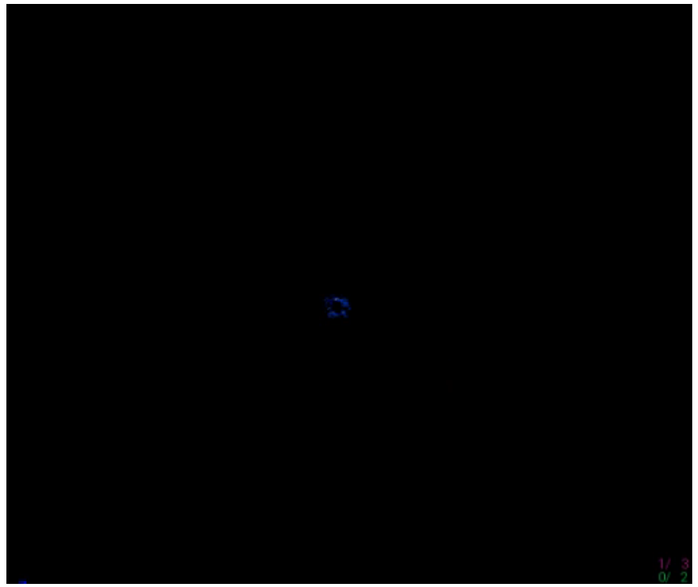
- In development for over 7 years (started in UE3).
- Released to Early Access in July 2017 (PC / Mac / PS4 / Xbox).
- Battle Royale (PvP).
- (Almost) worldwide.
- Fortnite is a success for Epic and Unreal Engine.

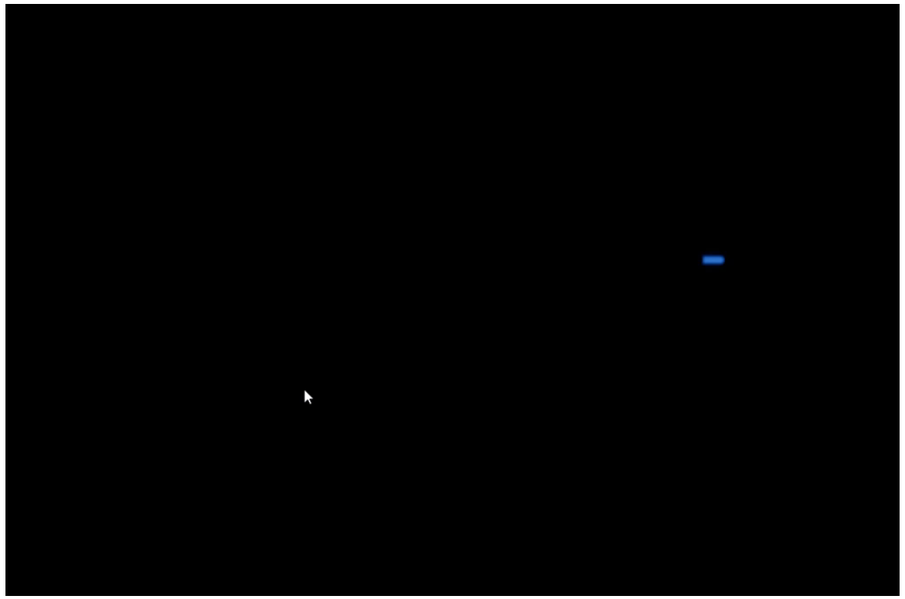
Weak Points

- PvE (Original Campaign) relies heavily on looting.
- Weak points were introduced as a mini-game to break up the monotony of harvesting for resources.
  - The original effects themselves were incredibly bright & strobed.
  - Many players did not understand the connection or purpose of the visuals before them.
- Very non-traditional problem for FX.
- How do teach players to use weak points in an organic fashion?


1 Go
832 832
1,840 1,840
100 100
22 50
817 891 800 0 999
+25
S
W
E
N
Object Destroyed
+1
Fight the Storm
Search for the ATLAS target
[3:01] Bonus: Place the ATLAS fast!
Speed Run: 2 days left
F1 F2 F3 F4 F5
Q
15 15 15
3108 1005
3 0
Z X C
1 2 3 4
Global
Joining general chat. Type "/" for
Ver-3942568-INVALID-fpsigoen
Fortnite DevLatest
MCP REGION

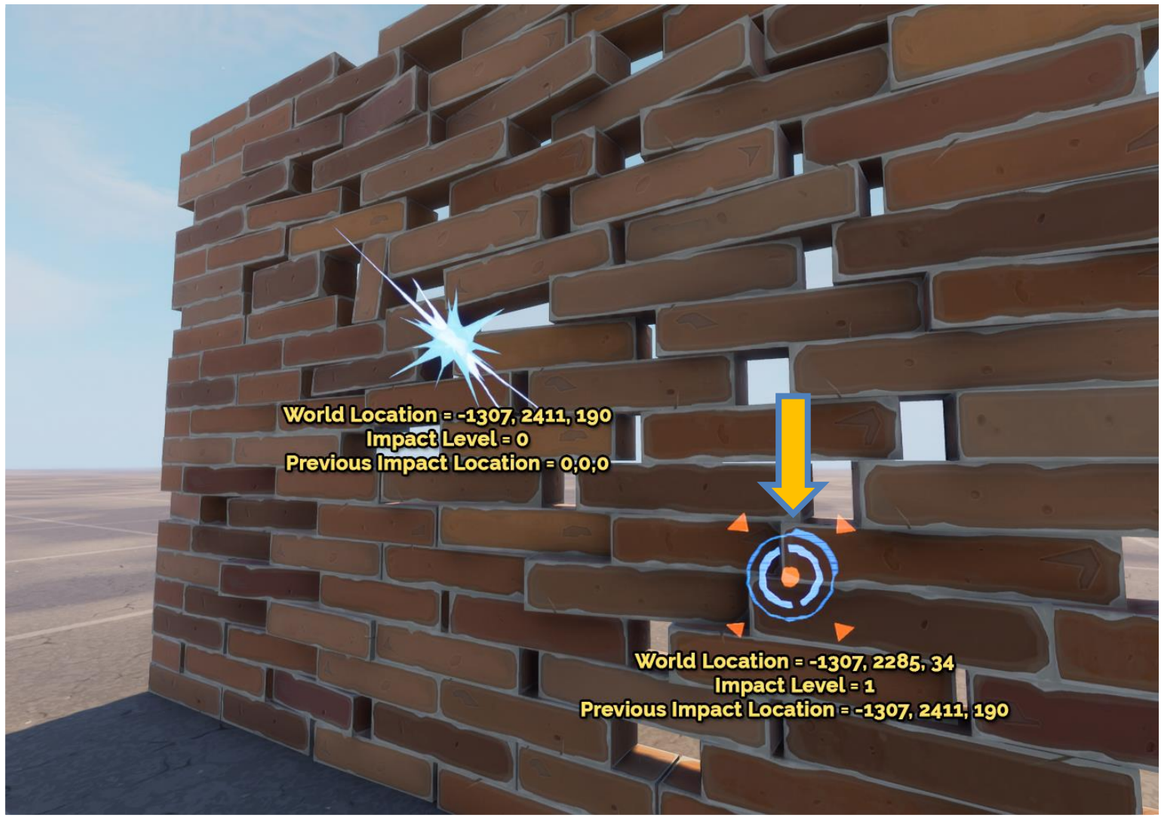
Weak Points





Weak Points

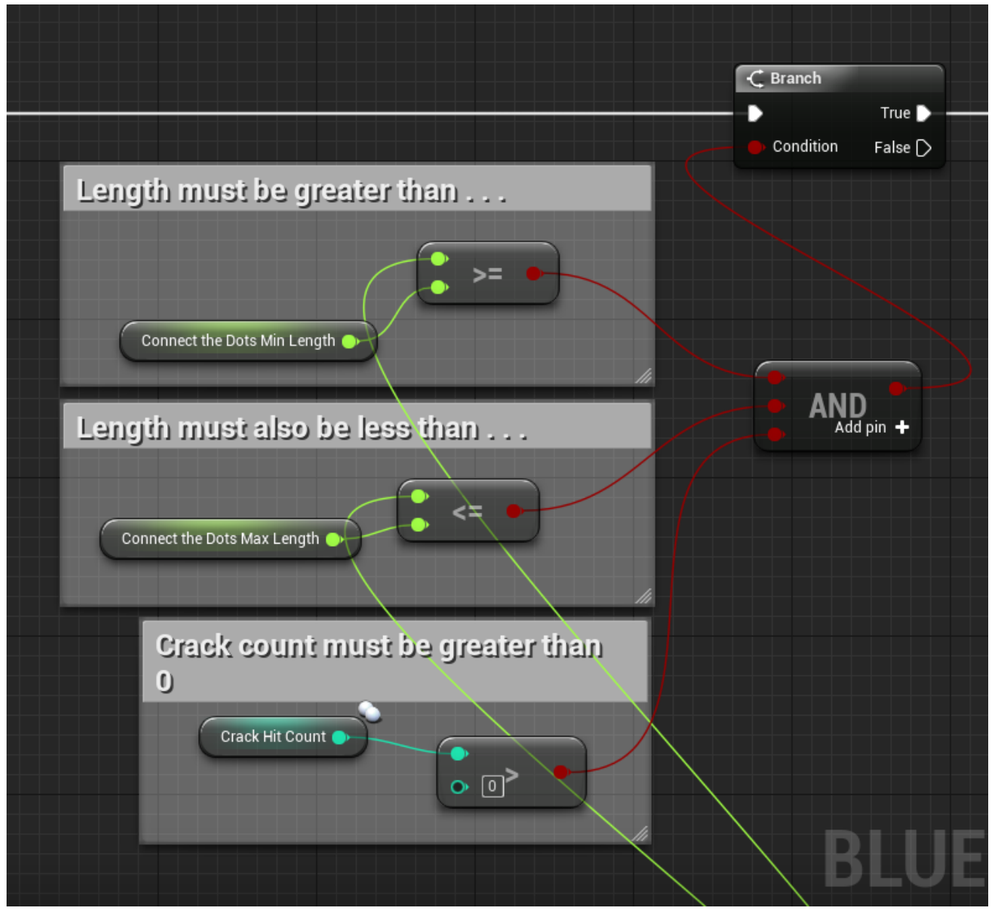
- When a player impacts a destructible actor, a weak point blueprint is spawned.
  - Has our base particle system as a component.
- If this is the first weak point in a series
  - Impact Level (int)= 0
  - Previous impact location (v3) = 0, 0, 0
  - **Integer** = Whole numbers
    i.e. - 22
  - **Vector3** = Collection of floating values
  - i.e. - (7855.113, -284.134, -0.942)
- If we're part of a series, we instead get a valid previous impact location in world space.



Weak Points

- Now we have simple rules we can test against to connect them:
  - If our impact level is > 1
  - If our distance between weak points is <= 512 units
  - Or if our distance is >= 50 units
- If any of these = false, then don't add a mesh.

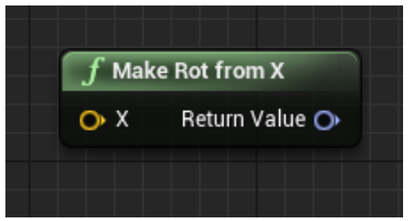
Or else, what do we need to do to figure out how to connect them?

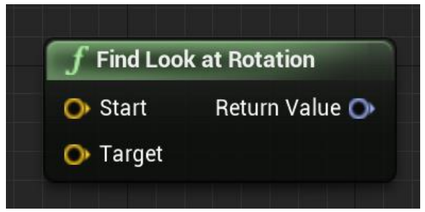


Weak Points

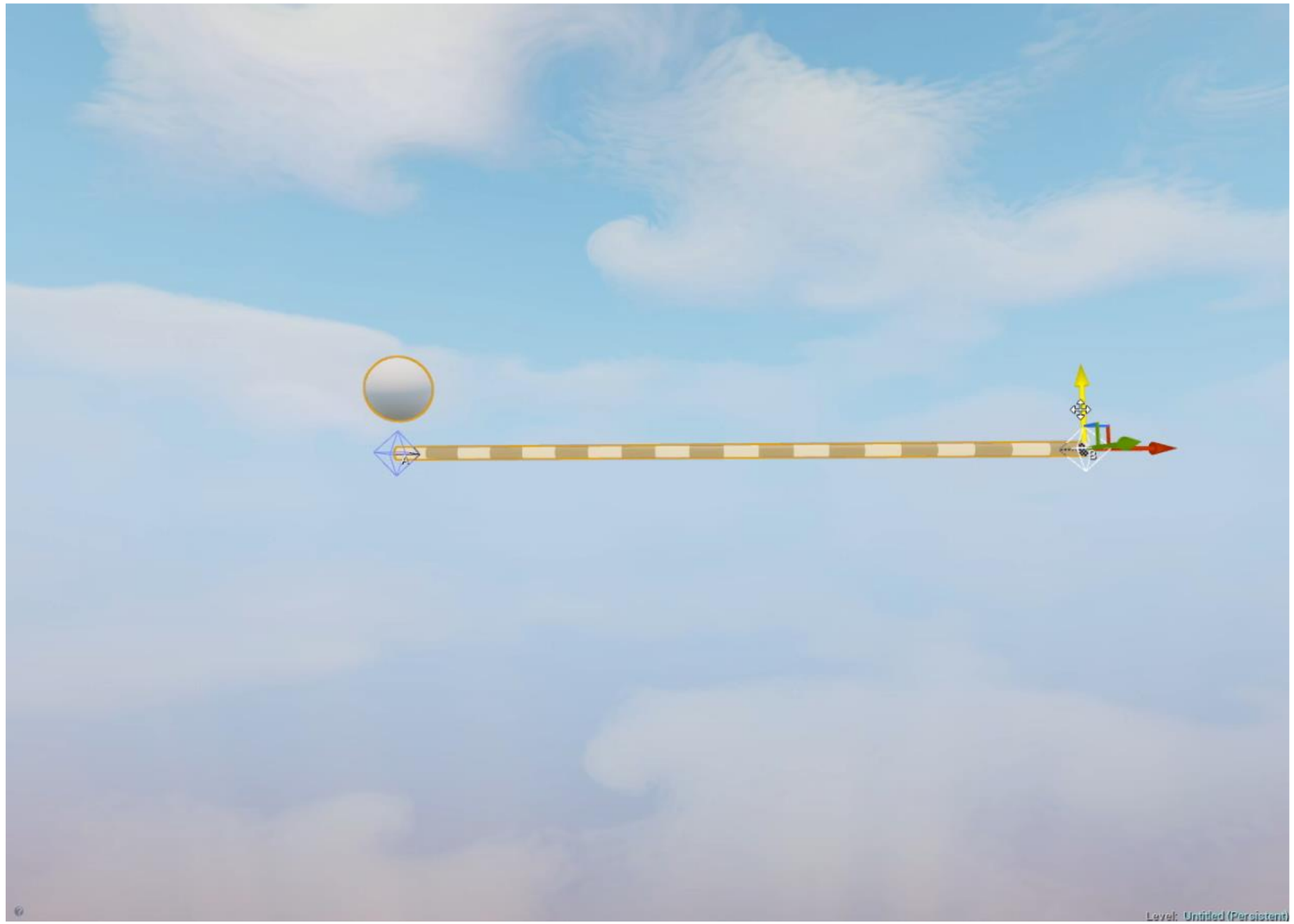
- A simple but incredibly useful equation in vector algebra:
  - If you want a vector that points from Point A to Point B, simply subtract B from A.
  - The result gives you a vector that not only can give you rotation values, but distance as well

B – A = 0, -126, -156



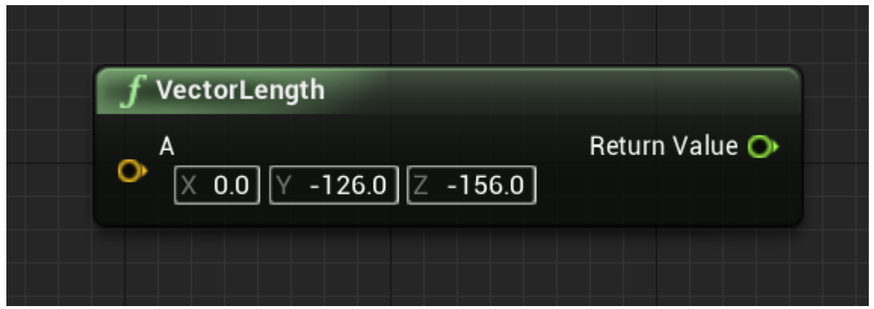


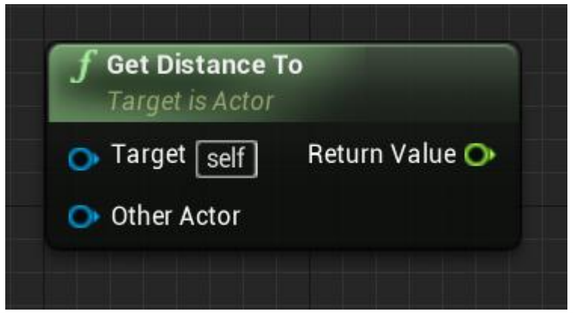
Rotation = -51.07, -90, 0

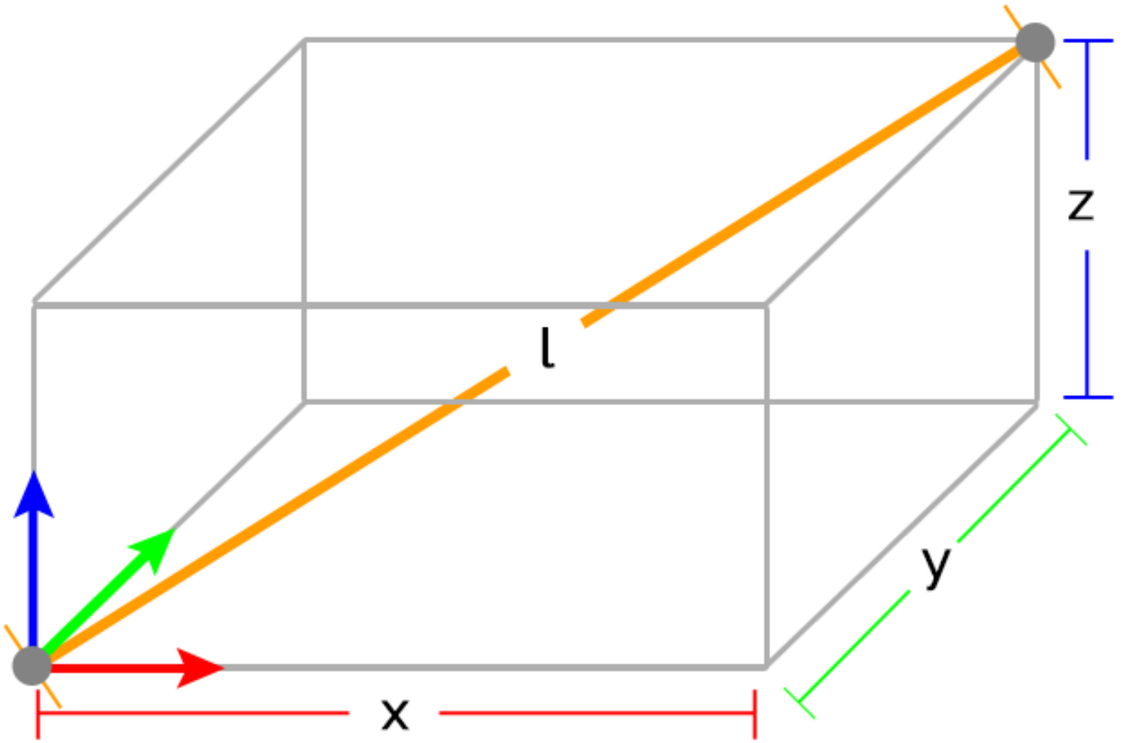

Level: Untitled (Persistent)

Weak Points

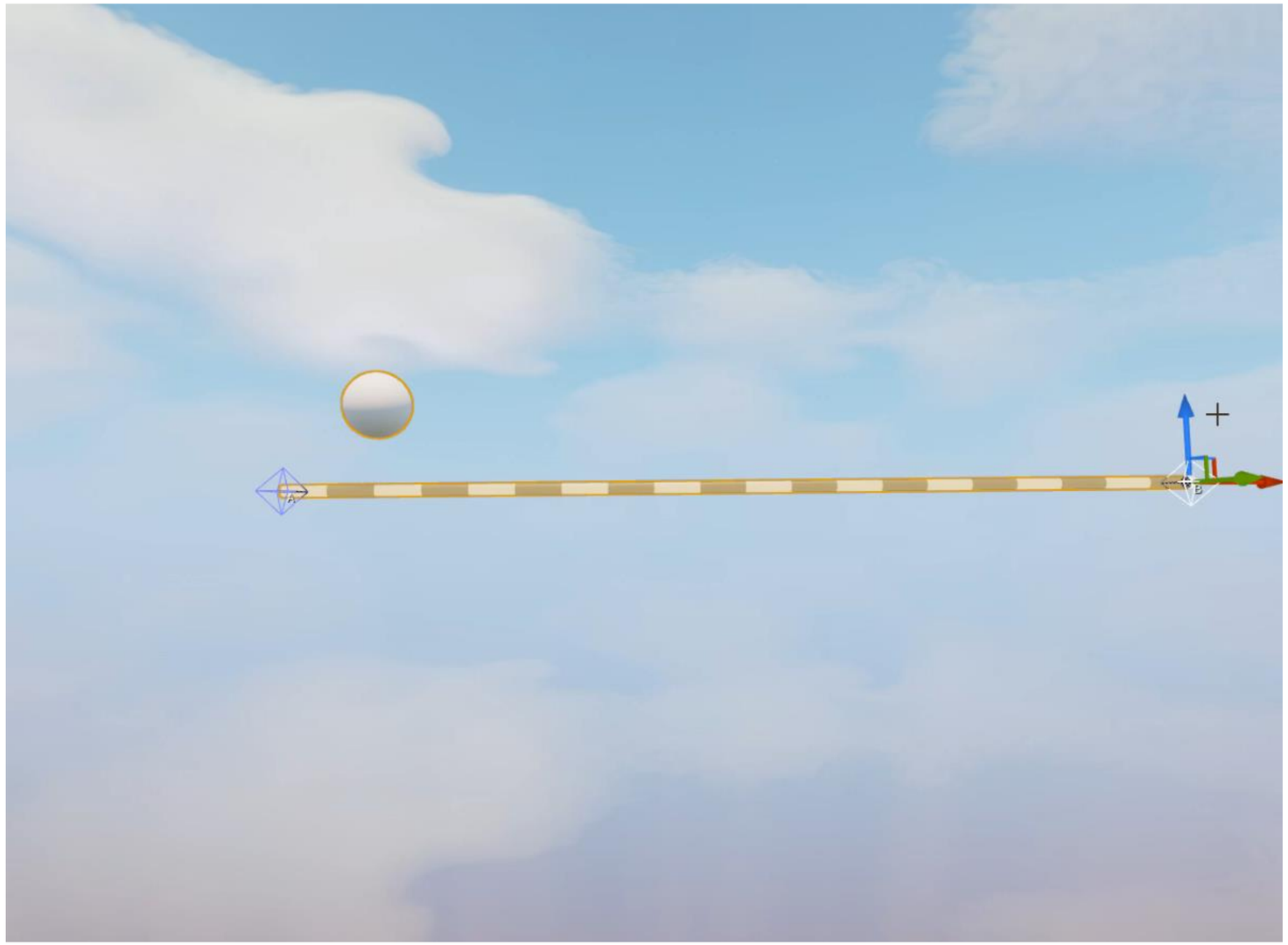
- Our rotation works perfectly, but our mesh scale is constant, no matter the length.
- We can take our result from B – A in our previous slide to determine its length (magnitude).
- Pythagorean's Theorem to the rescue!





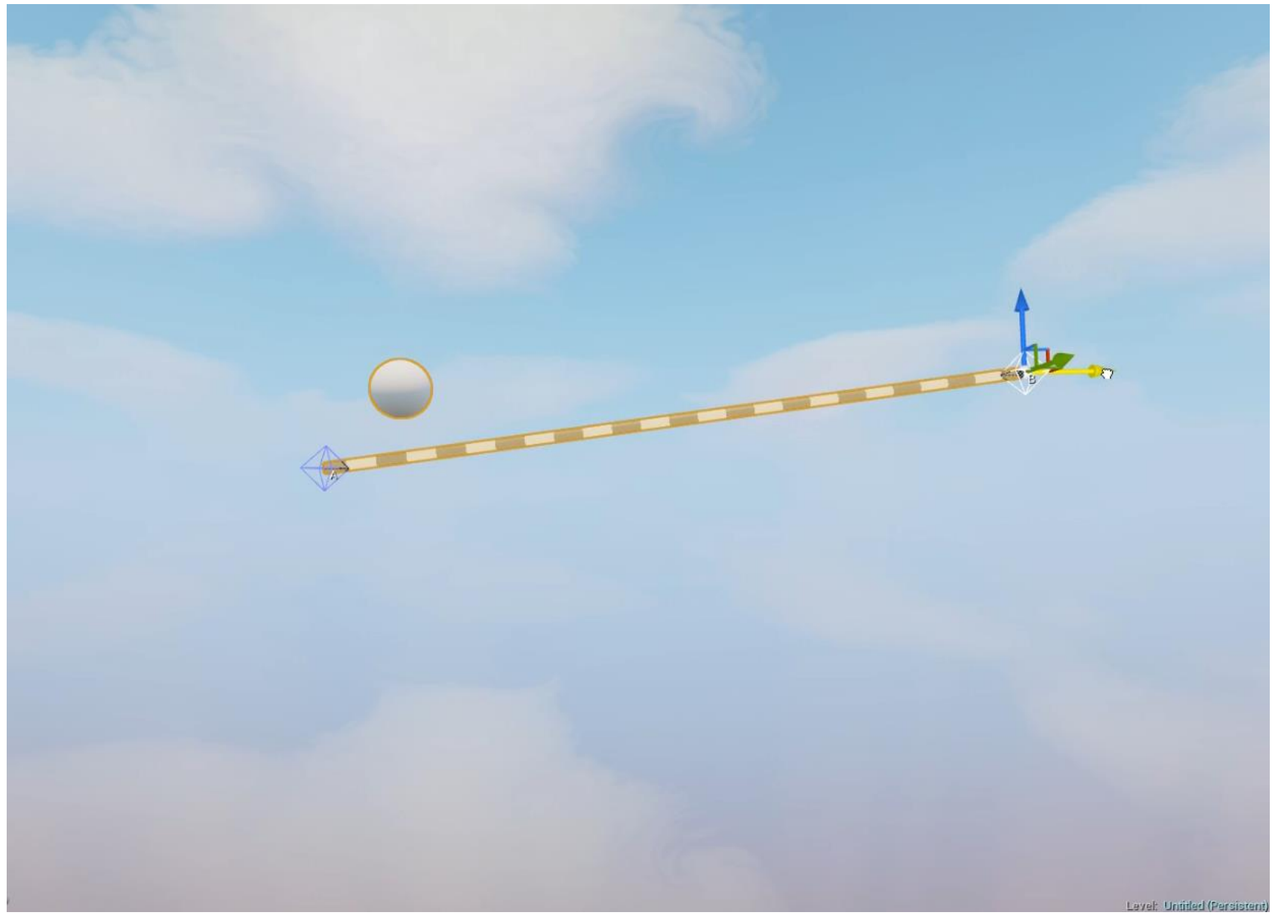


$$x^2 + y^2 + z^2 = l^2$$


A
B

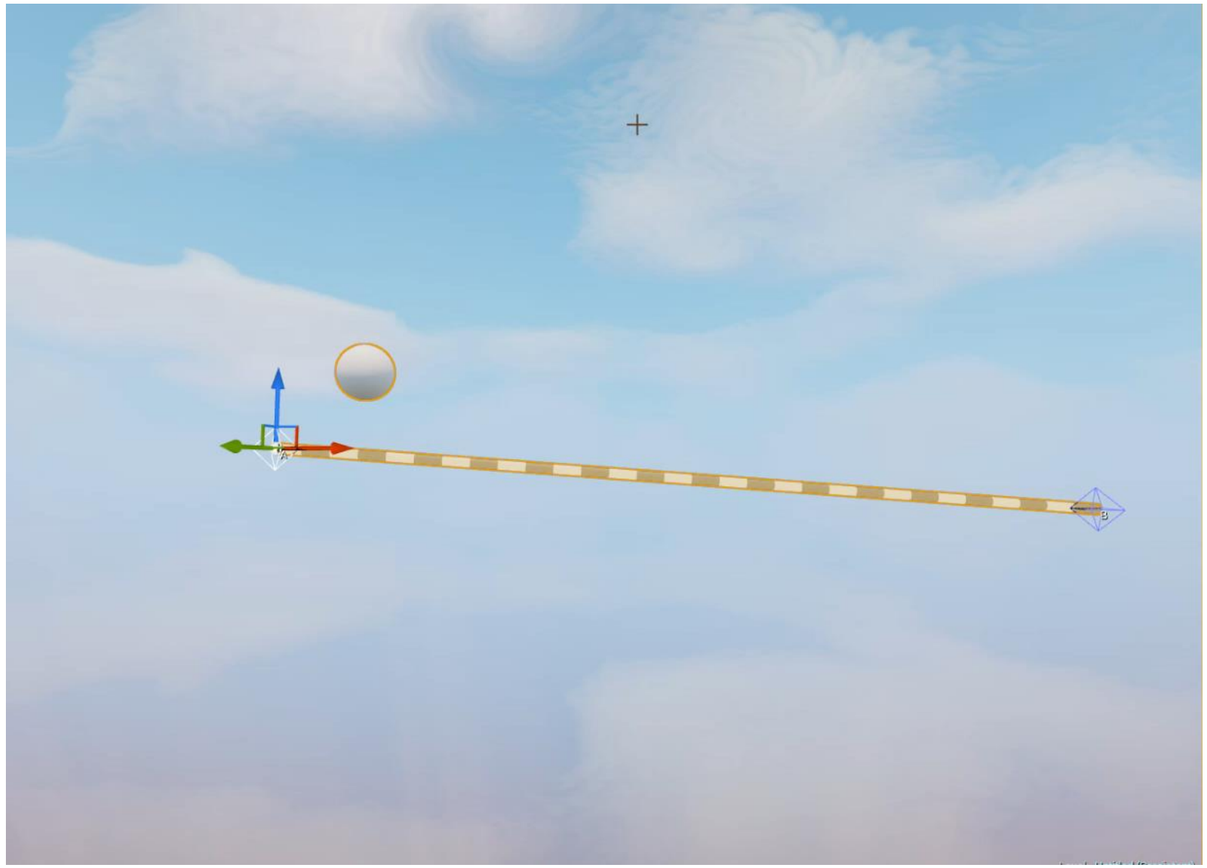
Weak Points

- We can take our vector length, and use that drive parameters within our material.
- UV tiling based on distance.
  - Use a base length (i.e. tile every 30 units)
  - Distance / 30 = New UV Scale



Weak Points

- We can ceil (up) the resulting value to avoid fractional tiling.
  - Ceil(137.15) = 138
  - Floor(137.15) = 137




1 Go
832 832
1,840 1,840
100 100
22 50
817 891 800 0 999
+25
S
W
E
N
Object Destroyed
+1
Fight the Storm
Search for the ATLAS target
[3:01] Bonus: Place the ATLAS fast!
Speed Run: 2 days left
F1 F2 F3 F4 F5
Q
15 15 15
3108 1005
3 0
Z X C
1 2 3 4
Global
Joining general chat. Type "/" for
Ver-3942568-INVALID-fpsigoen
Fortnite DevLatest
MCP REGION

Blaster Husk

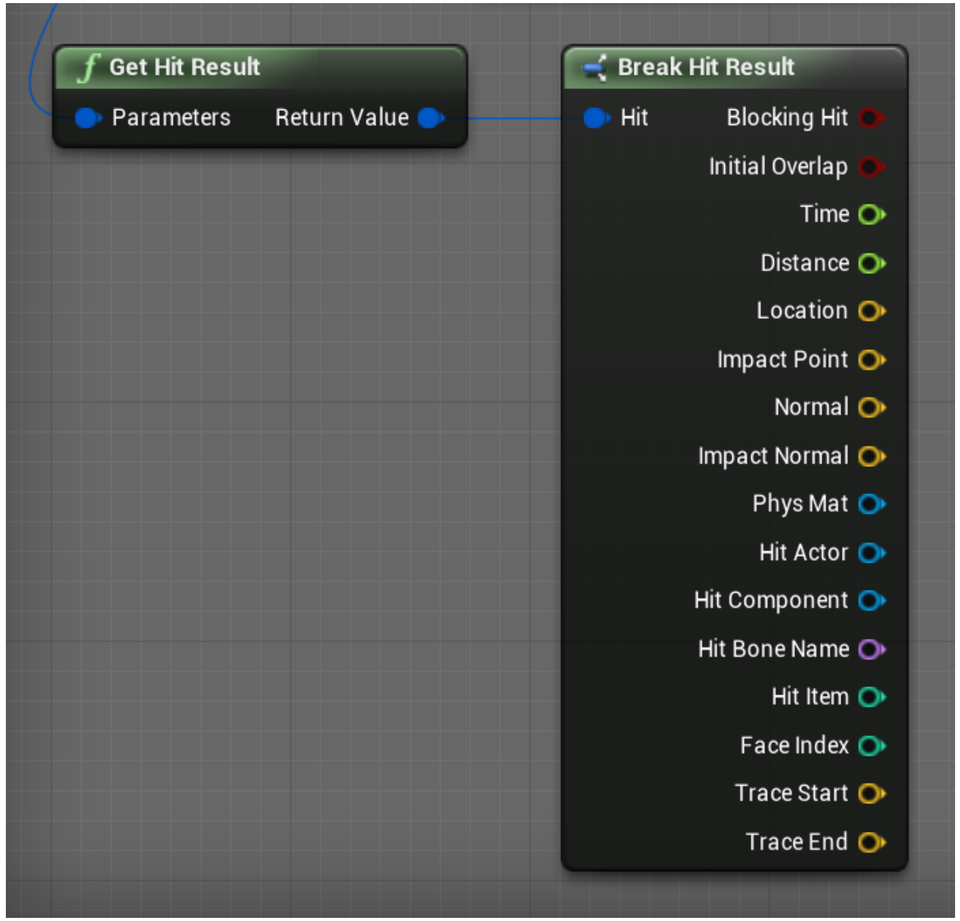
- We can take the exact same principals from weak points and begin to apply them in other instances.
- The Blaster Husk is an incredibly powerful and damaging enemy, and his FX needs to visually convey the level of damage he can inflict.
- The original catalyst to a custom blueprint setup relied on frustrations with beams in Cascade.

Blaster Husk

- The core elements of a laser beam can be broken down into a cylinder with supporting particle elements.
- Defining a multi-layered visual laser that's stylized with only one mesh (cylinder).
- Rely on Ryan Brucks' "Axis Aligned Fresnel" material function for consistent falloff regardless of camera angle.
- Establish a base length. Define what the laser looks like at a specified number of units (500 units).

Blaster Husk

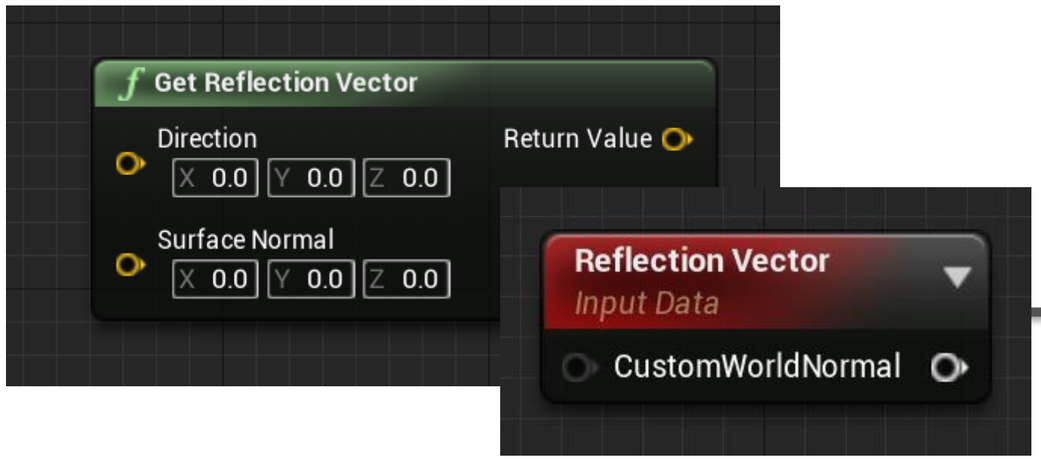
- The attack uses a hit-scan (line trace)
- Since we're using a trace, we can break the hit results and get all sorts of useful data we can use.
  - How can we use this data?
  - **Struct** = Group of data / variables

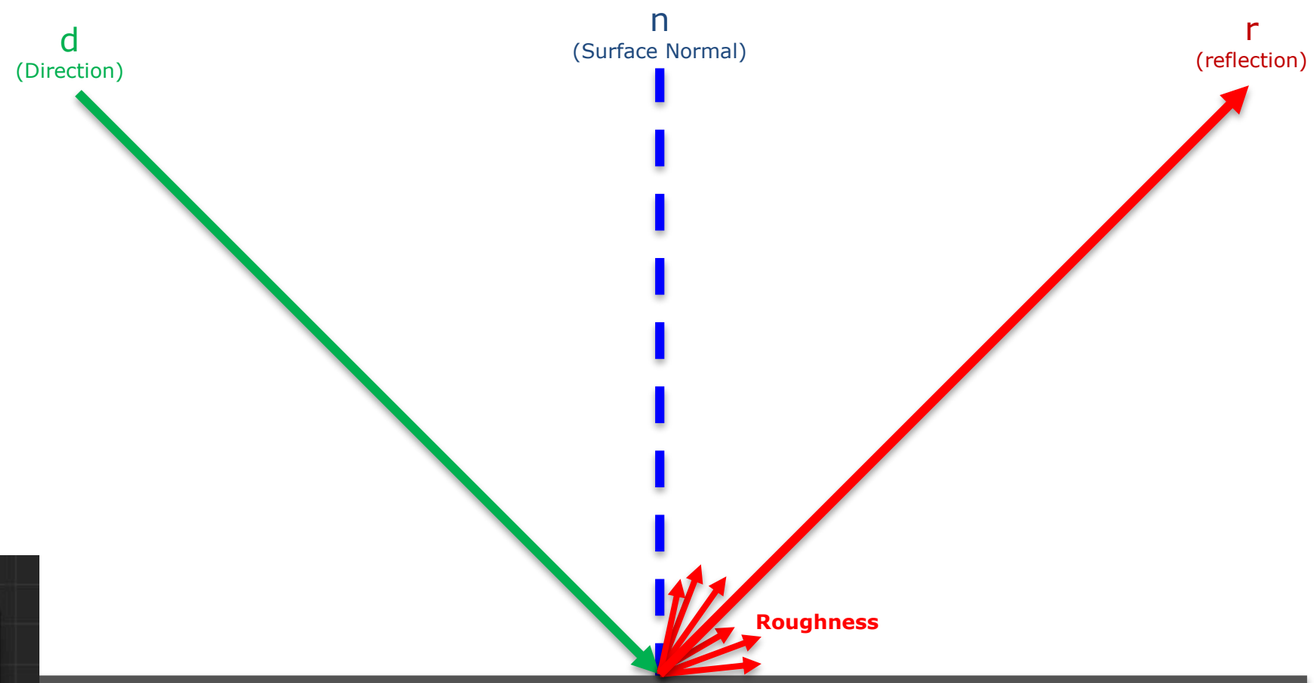


Blaster Husk

- Reflection Vector
  - As the name implies, it's the simple way to calculate the angle and which anything will bounce off a surface.

$$r = d - 2(d \cdot n)\, n$$



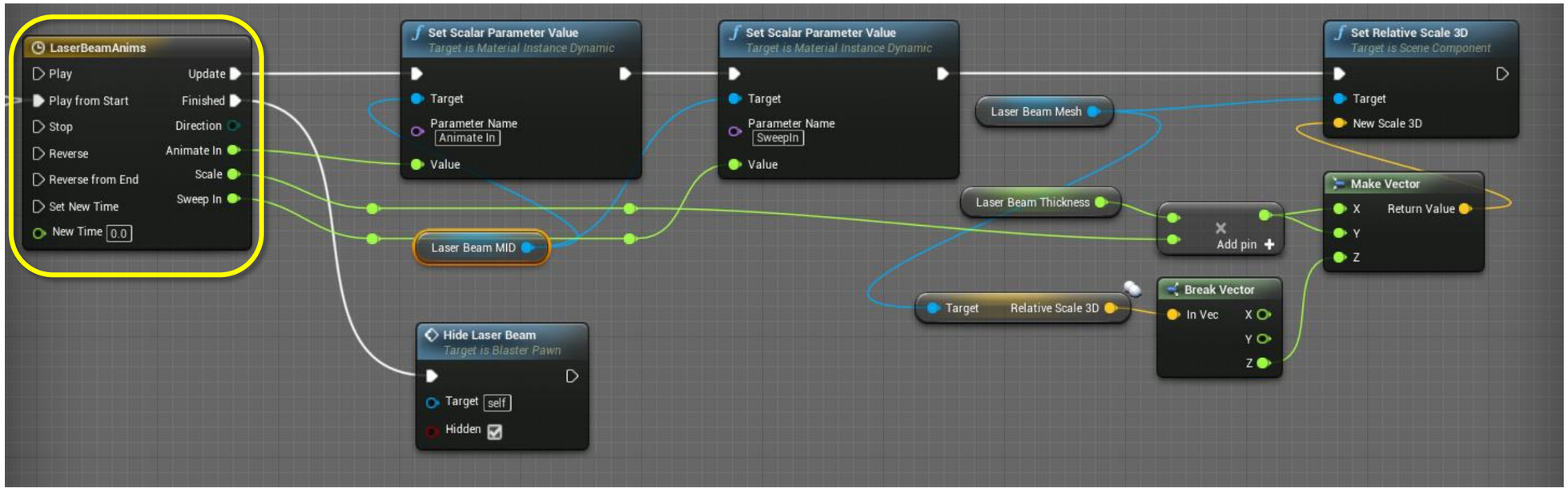



Perspective
Lit
Show
10°
0.25


Perspective
Lit
Show
Normalize to Range
Value 0.0
Range Min 0.0
Range Max 0.0
Return Value
Clamp (float)
Value
Min 0.0
Max 1.0
Return Value

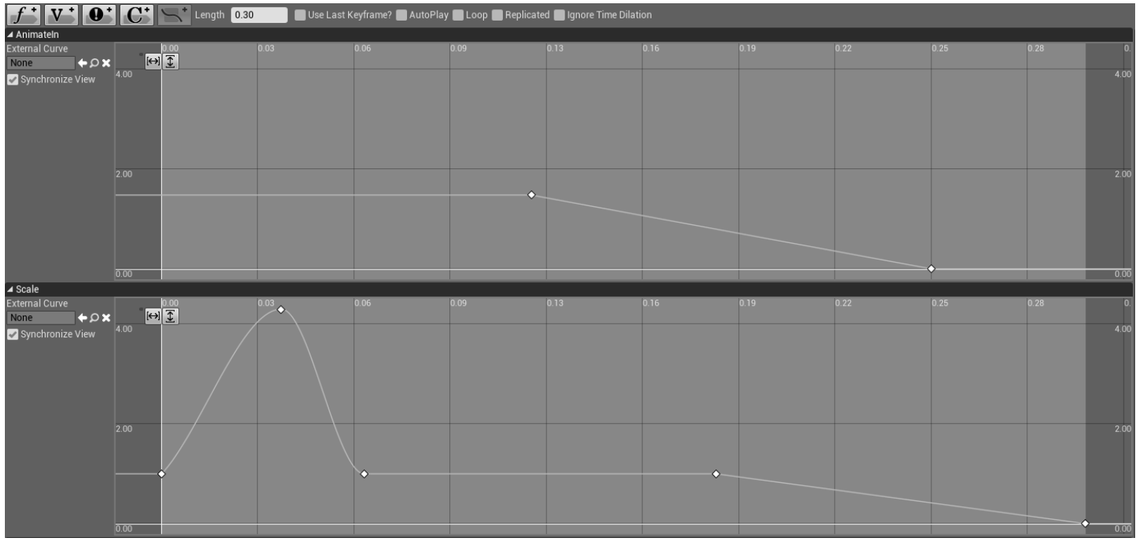
Blaster Husk

- Timelines
  - Arbitrary latent functions similar to something like sequencer/matinee.
  - Outputs can directly control anything, but commonly used things:



Blaster Husk

- We can use multiple tracks in a timeline to:
  - Sweep the laser in.
  - Erode away the material's emissive.
  - Scale the mesh down on X & Y.




1 Go
0 172
484
100 100
49 50
500
500
500
0
0
E
F1
F2
F3
Q
Global
Joining general chat. Type "/" for options
1
2
X

Weapon Alterations

- Fortnite currently has 400+ weapons (ranged and melee).
- Each weapon must have a unique "elemental alteration" based off 4 types (Fire, Ice, Electric, & Energy).
  - Now we're 1,600+ weapon permutations needing FX.
- Each elemental type uses the same particle system across all weapons in the game.
- Using Blueprints and Cascade, we can easily customize the particle system procedurally to fit upon the weapon.

Weapon Alterations

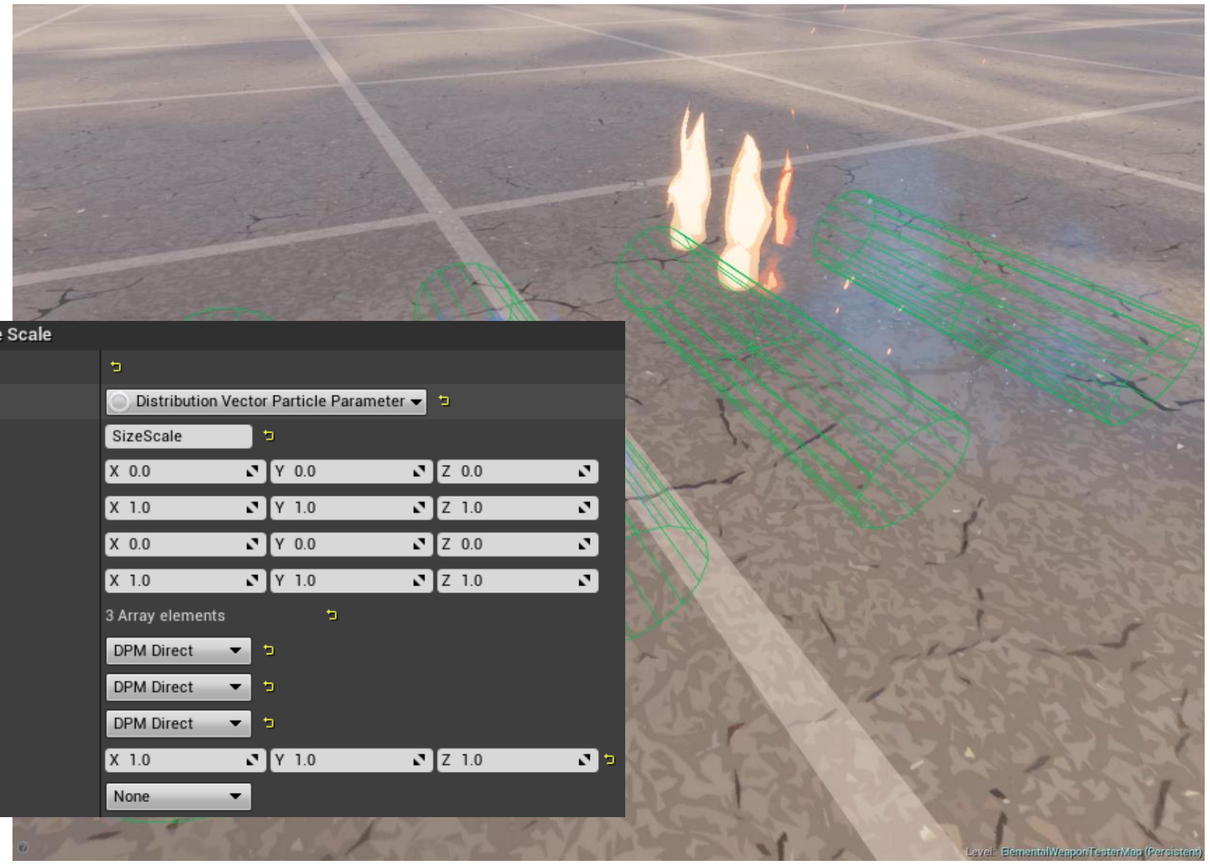
- Each weapon is a skeletal mesh . . .
- We can add sockets to our parent skeleton for things like the muzzle flash, shell ejects, etc.
- But then we can do something really cool and convert these to mesh sockets.
  - A mesh socket is simply a transform override on a per-mesh level.

Weapon Alterations

- 1st part of altering the weapon is modifying the material.
  - Originally developed by Jon Lindquist and helped inspired my subsequent ideas on how to approach this.
- We already have a “muzzle” socket where the particle system for muzzle flashes are attached to.
- A 2nd socket called “muzzle falloff” is added and moved as a mesh socket on a per weapon basis.
- The distance between the 2 sockets is fed into the dynamic material instance to give artistic control on a per-weapon basis.

Weapon Alterations

- Everything emits from a cylinder to roughly fit the shape of the weapon body and muzzle.
- All 4 have the same layout and set of parameters.
  - Cylinder Height (s)
  - Cylinder Radius (s)
  - Particle Size Scale (v3)
  - Spawn Rate Scale (s)
  - **Scalar / Float** = Floating point value
    i.e. - 87.4139



Weapon Alterations

- We can apply a similar math principals for our particle systems.
- We'll create 2 new sockets (fx_start and fx_end).
- Attach the particle system to "fx_start".
- We can go back again to our rotational and vector magnitude principals.
  - Get a **rotation** value between the "fx_start" and "fx_end" sockets and use this to get the particle system to rotate automatically.
  - Measure the distance from "fx_start" to "fx_end" and use this drive the **cylinder height.**

Weapon Alterations

- But when we start modifying parameters of the cylinder, we can easily start to get problems.
- But how can control radius uniquely and quickly as well?
- The mesh sockets have a full set of transforms (location, rotation, & scale).
- Rotation and scale aren't currently being used.
- We can hijack these values at our discretion.
  - Scale XYZ = Cylinder Radius & Size Scale

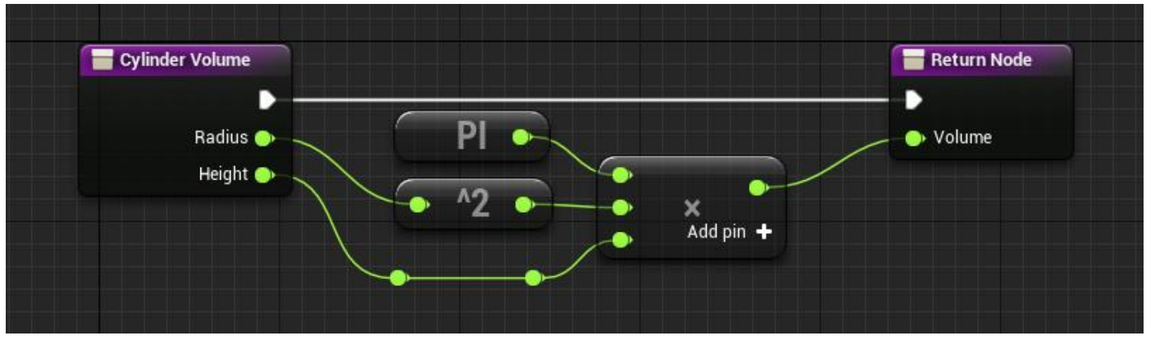
Weapon Alterations

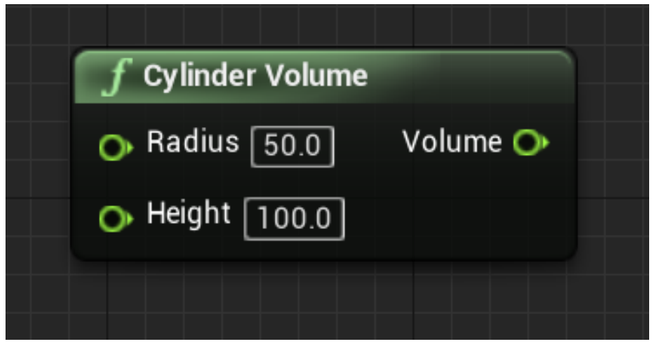
- But what about spawn rate scaling?
- We can calculate the volume and compare that against a base volume.

  Volume = π r^2 h

- Simply divide our current volume against our base volume, and set this as a spawn rate scale.
  - Base volume = 1600
  - New volume = 2249

2249/1600 = **1.405**

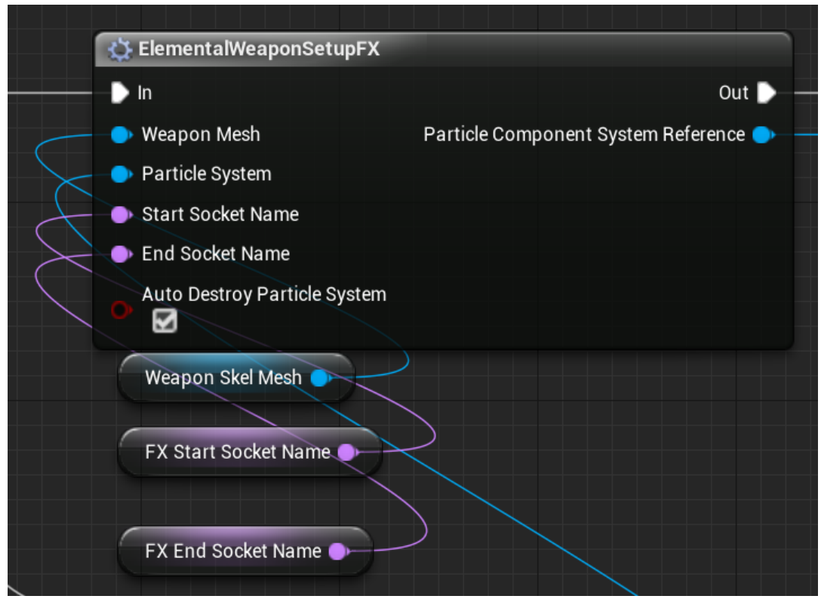




Weapon Alterations

Weapon Alterations

- How do we iterate & test in such large volumes?
  - Developing tools for testing is critical to efficiency when dealing with large volumes!



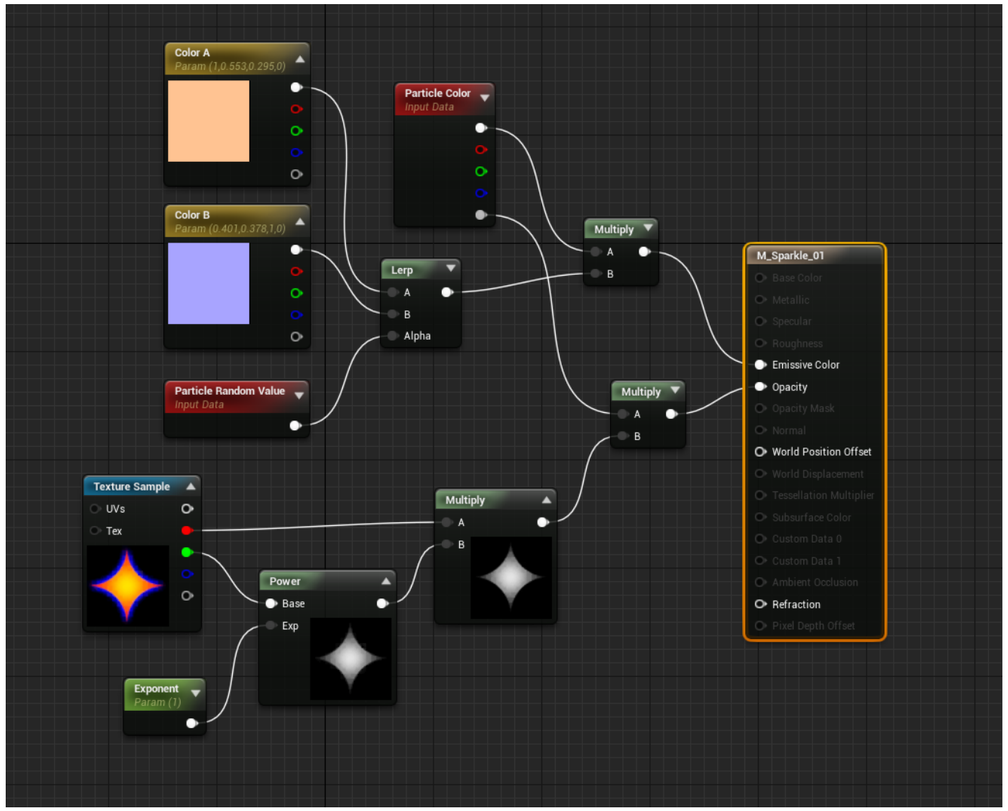
Automated Material Systems

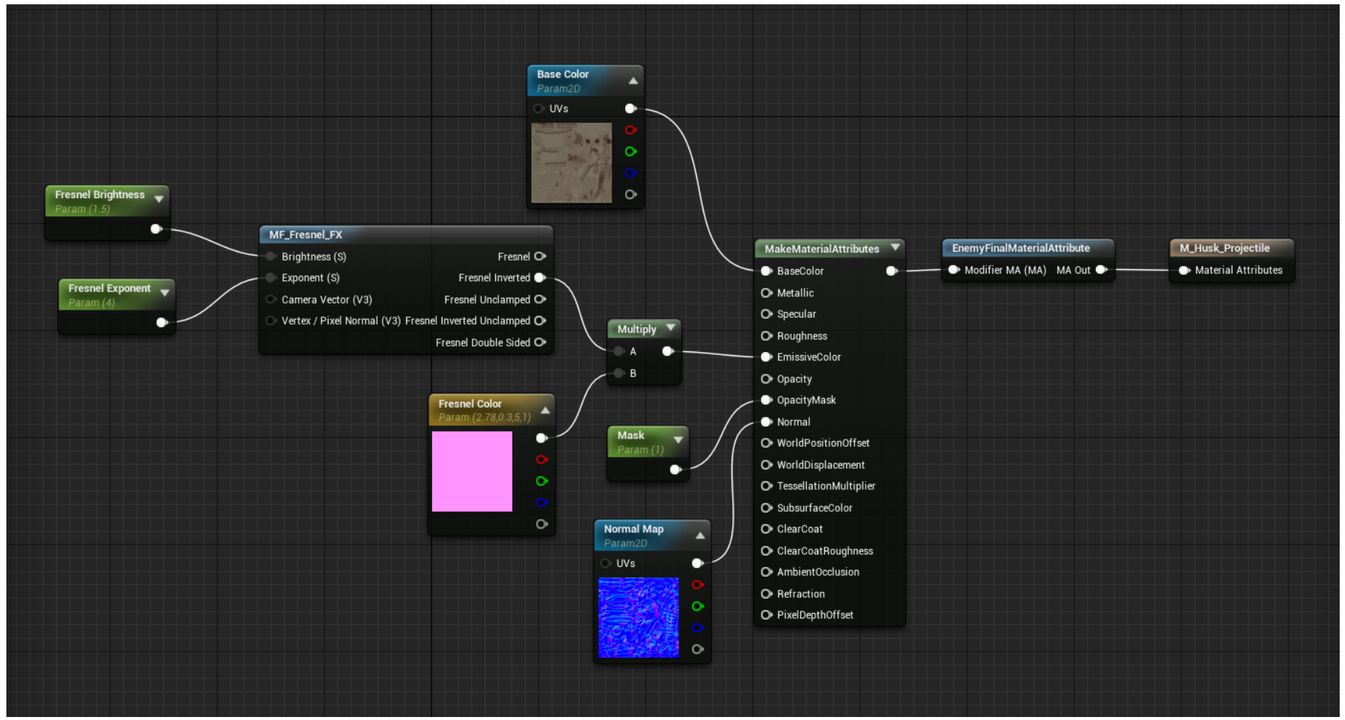
- Fortnite (PvE) currently has over 25 different enemies (most show below).
- Some of these are simple "elemental alterations" of base types (ie husk, fire husk, electric husk, etc).
- All 4 elements must be conveyed clearly in all states.
- There needs to be a system of organization to manage all of these visuals and components, and do it automatically.

Automated Material Systems

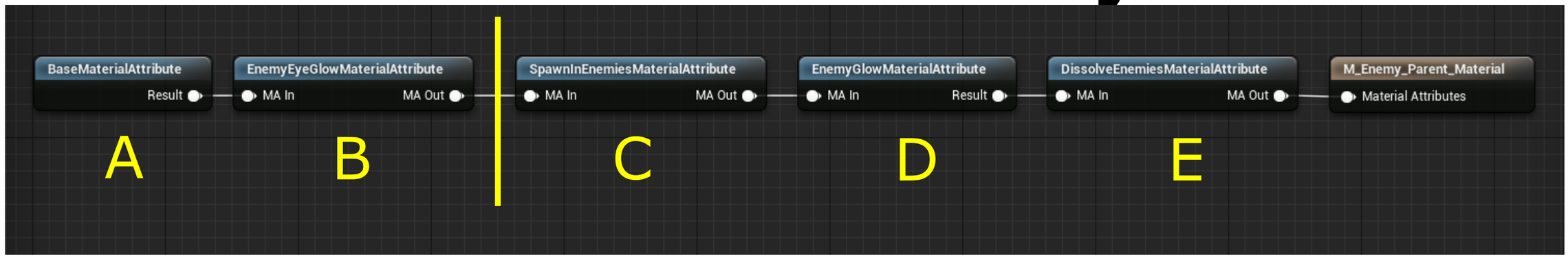
- We can apply specific damage alterations to enemies at runtime (a fire husk spawns in, becomes frozen, then dies).
- The enemy parent blueprint and material was restructured to support this.
- Material attributes were used as a "pipeline" to be able to establish a visual order of operations, as well as interchangeability between materials.

Automated Material Systems

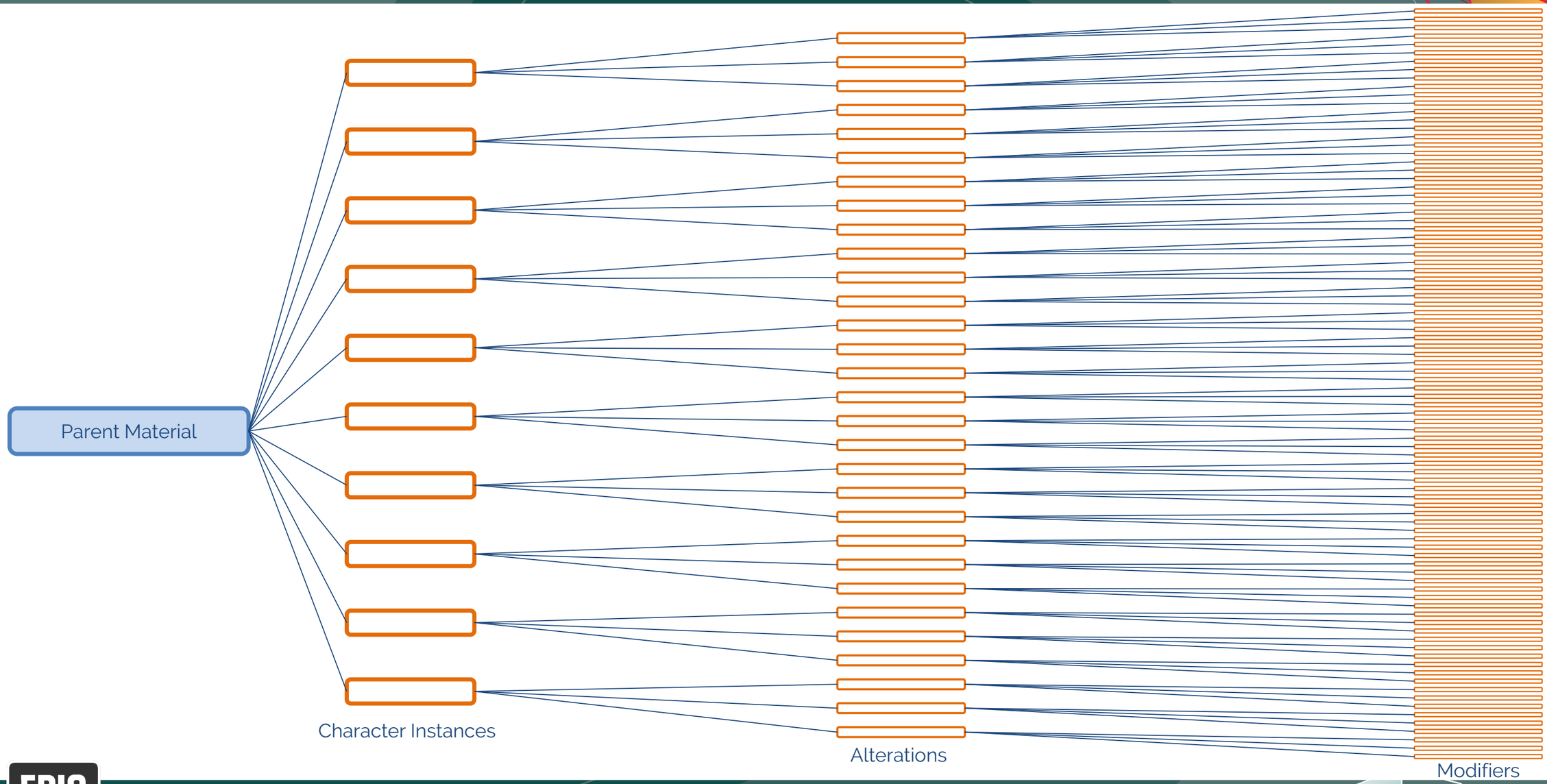


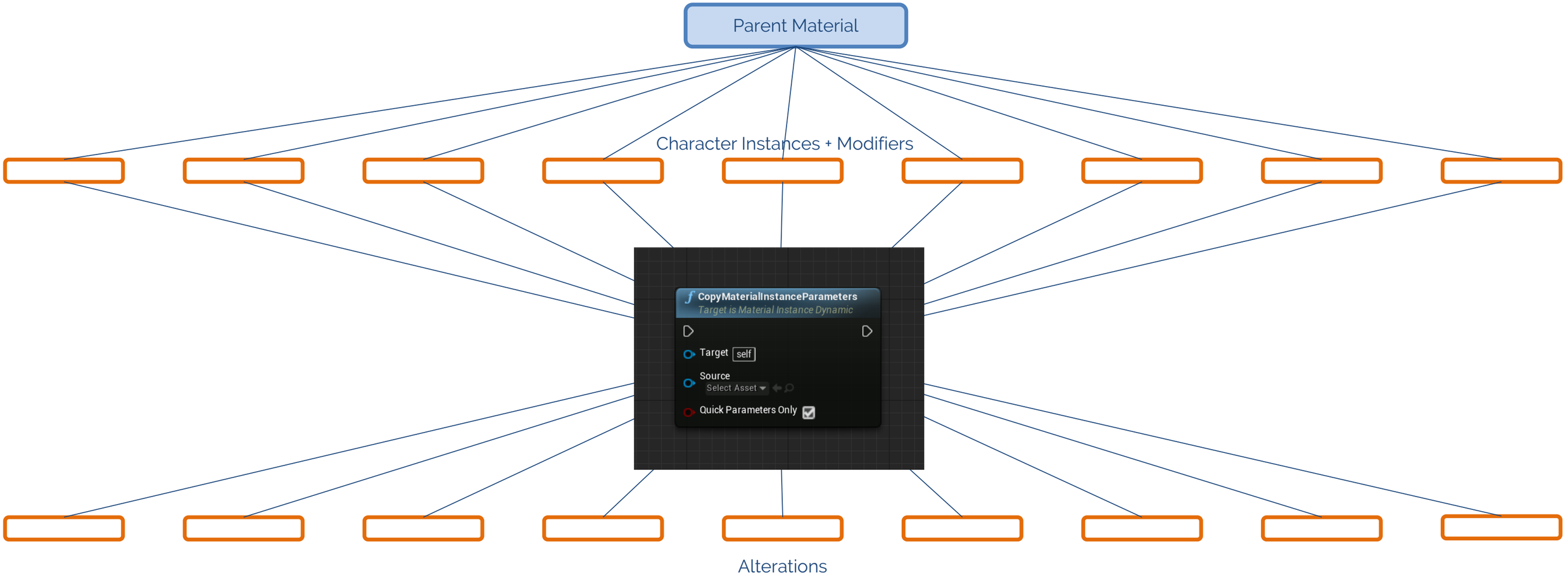


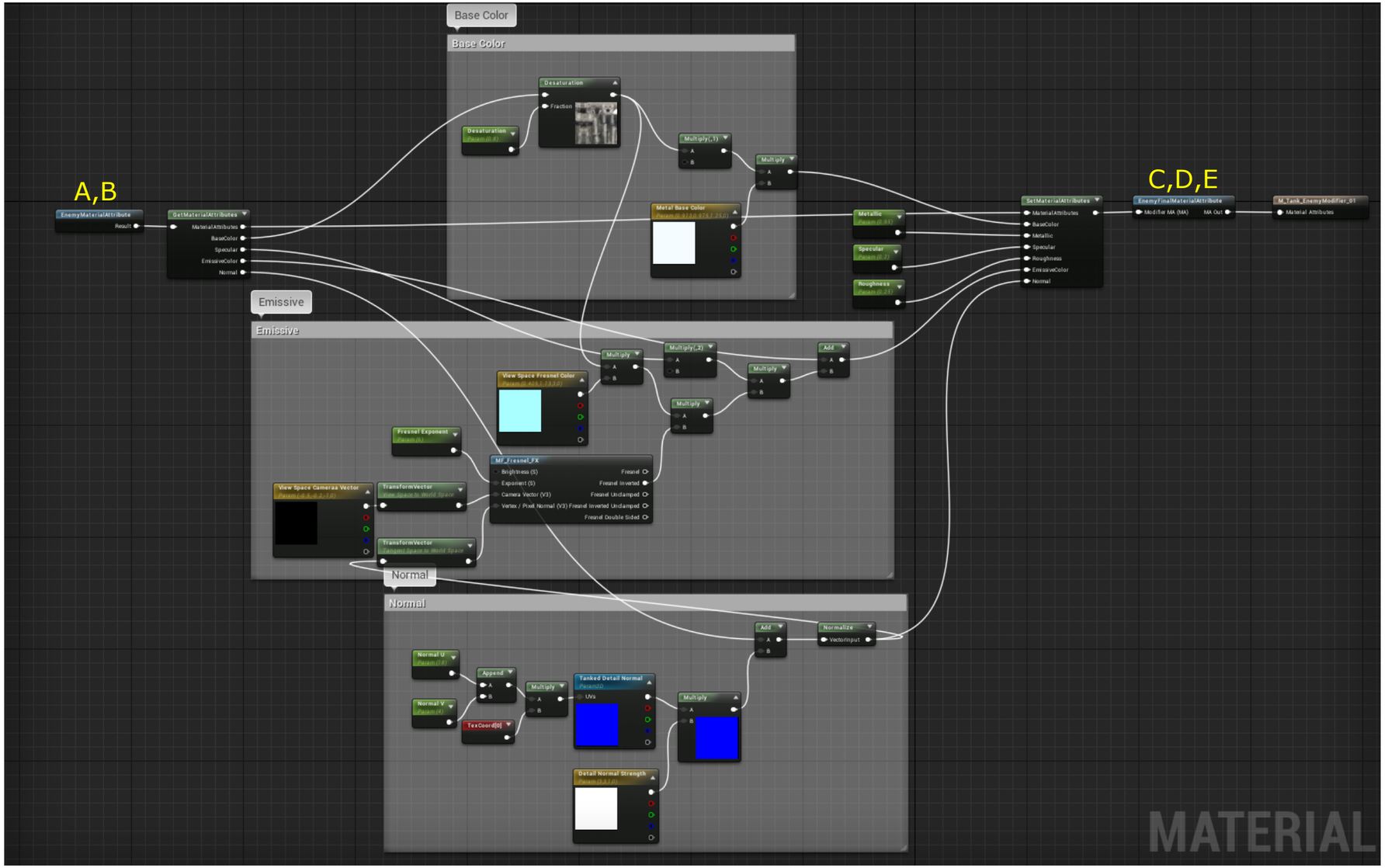
Automated Material Systems



- 5 material attributes make up the enemy parent material.
  - A: Base material - Things you'd typically find in a character material.
  - B: Eye Glow – Uses vertex colors & fresnel to control husk eye emissive values.
  - C: Spawn-In – Enemies are typically spawned from a small rift from the storm.
  - D: Glow – Primarily used to flash the enemy when being hit for visual feedback.
  - E: Dissolve – How enemies erode away while dying.
- We intentionally leave a split after eye glow. We can have separate materials that use the same material attributes order, with unique modifications located within this split.


Parent Material
Character Instances
Alterations
Modifiers

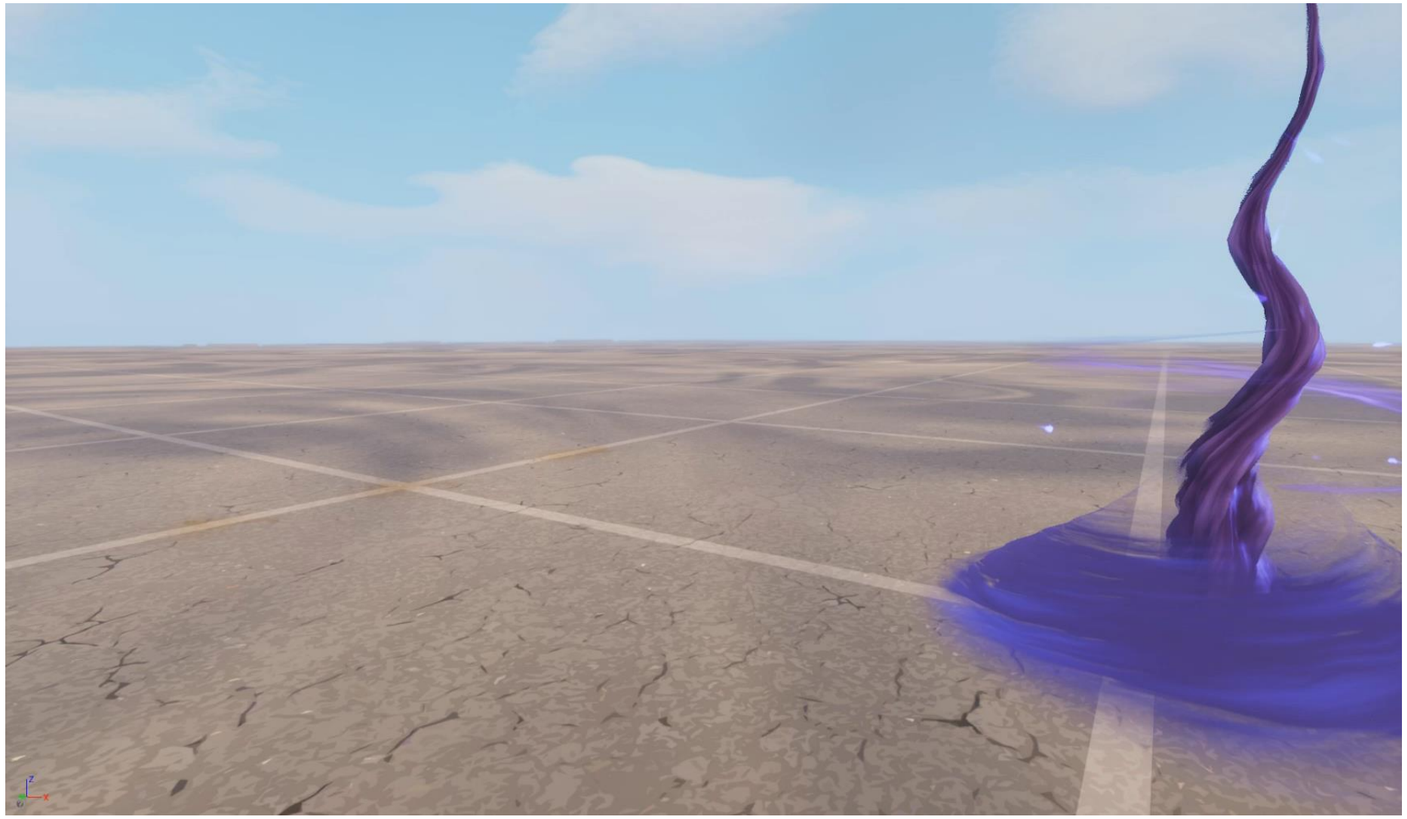

Parent Material
Character Instances + Modifiers
CopyMaterialInstanceParameters
Target is Material Instance Dynamic
Target
self
Source
Select Asset
Quick Parameters Only
Alterations


A,B
C,D,E
Base Color
Base Color
Desaturation
Fraction
Multiply
Metal Base Color
Metallic
Specular
Roughness
SetMaterialAttributes
EnemyFinalMaterialAttribute
M_Tank_EnemyModifier_01
EnemyMaterialAttribute
GetMaterialAttributes
Emissive
Emissive
View Space Fresnel Color
Fresnel Exponent
MF_FresnelFX
View Space Cameraa Vector
TransformVector
Normal
Normal
Normal U
Normal V
Append
TexCoord[0]
Tanked Detail Normal
Detail Normal Strength
Add
Normalize
MATERIAL


1
0

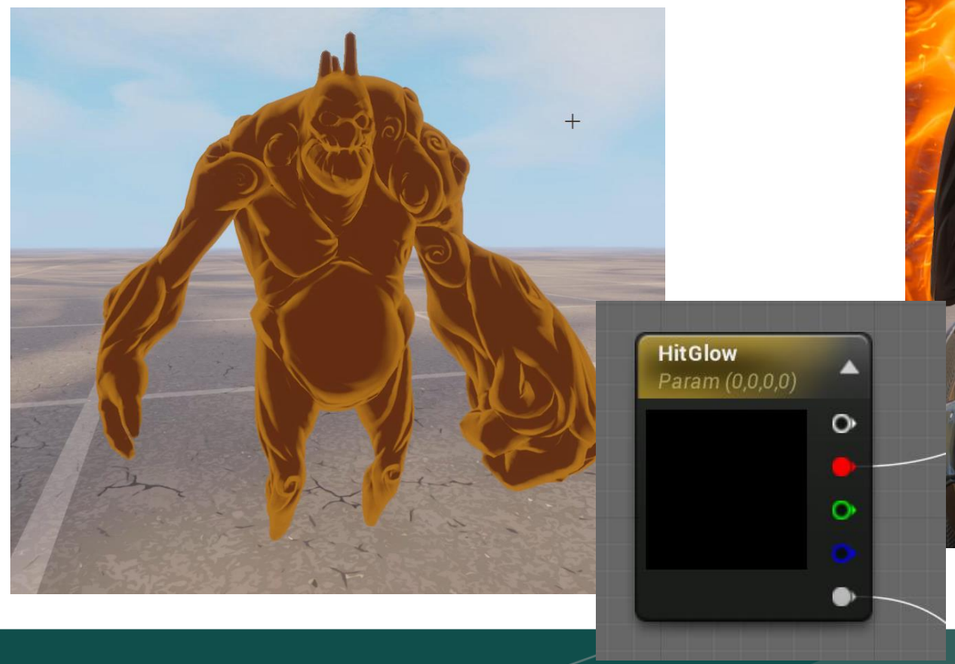
Automated Material Systems

- The spawn-in attribute relates specifically to how (most) husks enter the world during an encounter.
- Created by Jon Lindquist, the husk's verts are smeared in space using world position offset from the nearest rift in the world.



Automated Material Systems

- Hit-Glow
  - Used to give visual feedback for enemies that are struck by player impacts.
  - 2-tone additive emissive driven by fresnel.
  - Slight “jiggle” in WPO.



Automated Material Systems

- Dissolve
  - The dissolve attribute relates specifically to death.
  - Enemy death cannot be gory.
  - All husks in Fortnite die by sampling the last impact point in world space, converting this to pre-skinned local space, and driving a sphere mask across the entire mesh.

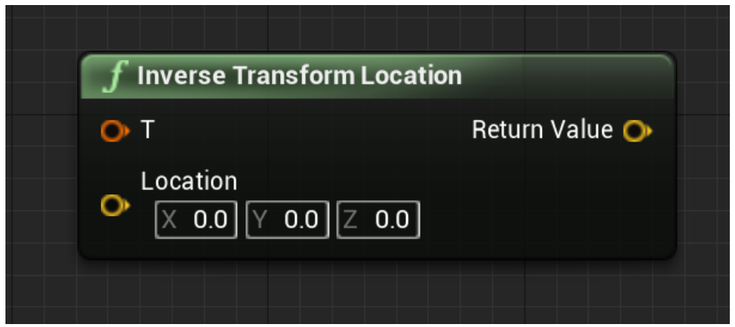
Automated Material Systems

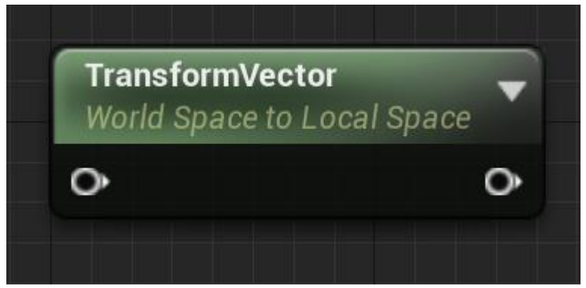
- Unreal now supports “Pre-skinned Local Position” & “Pre-skinned Normals” in the material editor.
  - Currently only done through the vertex shader.
  - We can retain the local position and surface normal before any sort of deformation.
  - This opens many doors for different types of material effects that were either expensive or impossible to try and track in a blueprint.


Perspective
Lit
Show
Death Anim
Local Space
(Normalized)
Preskinned Local Position
(Normalized)

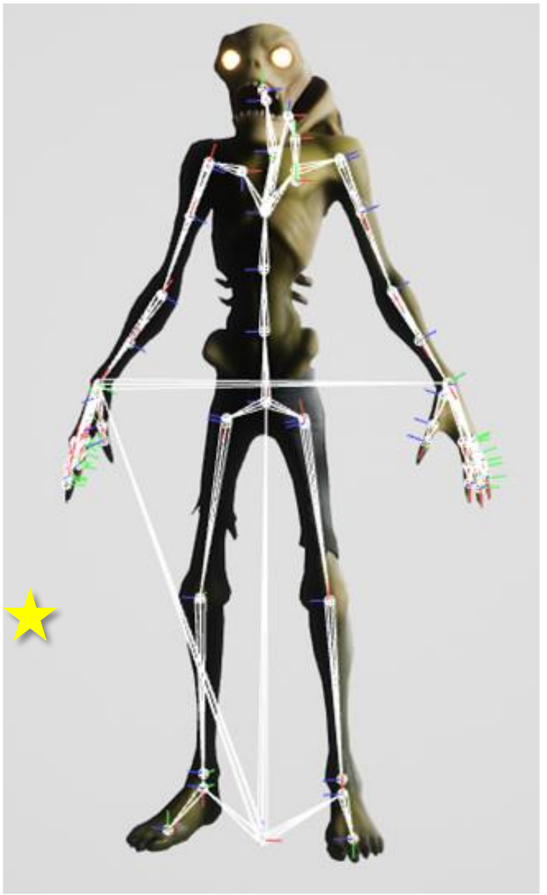
Enemy Death

- Converting to local space is super easy. In fact, we've already done it in this presentation.
- If we want to convert a vector into a relative offset of another vector, simply subtract the vector from our source.
  - Point A is my husk: (4300, -9540, 20)
  - Point B is my impact: (4325, -9535, 50)
  - Take B - A = **(25, 5, 30)**

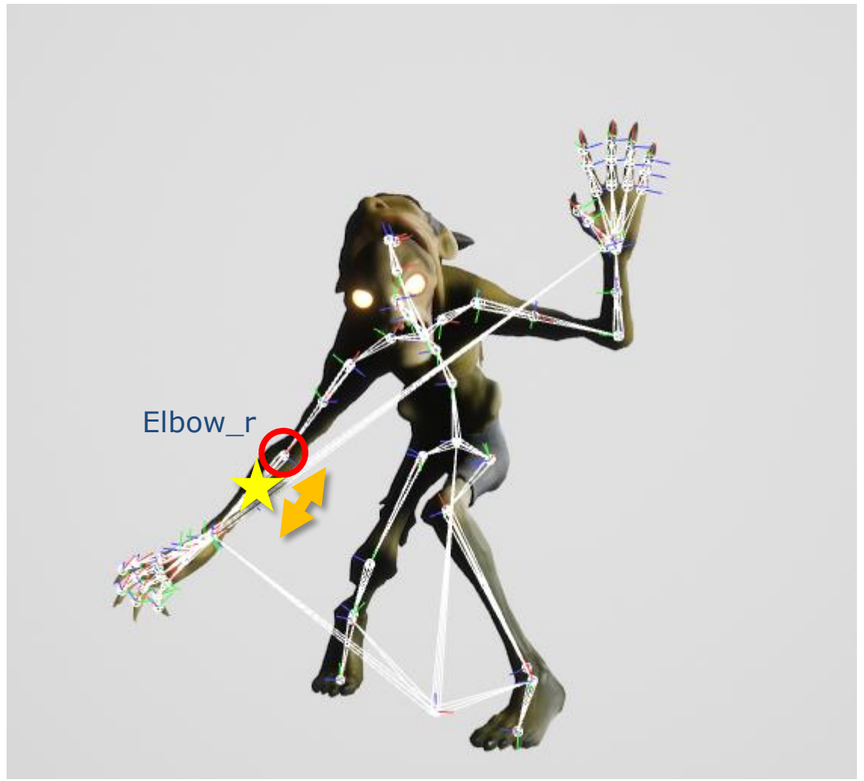


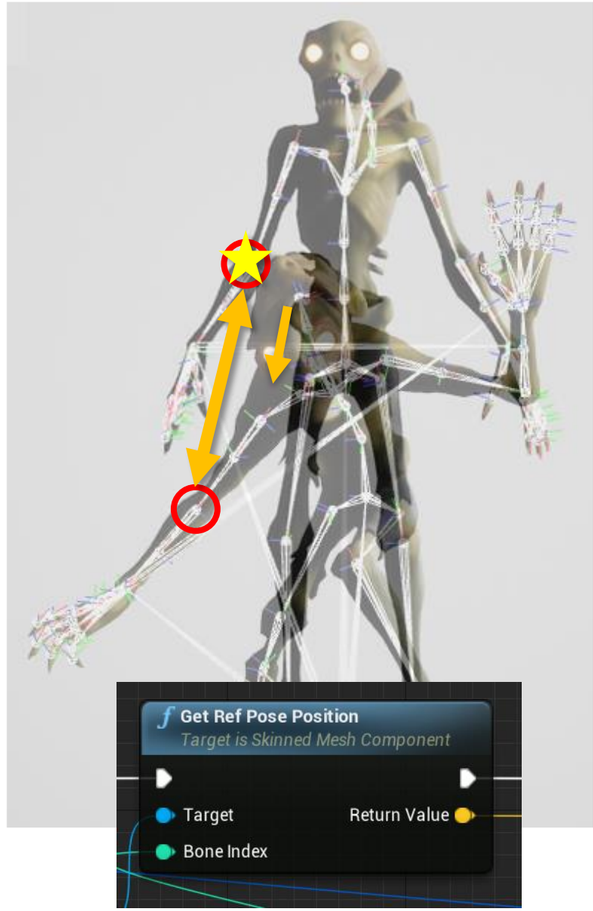


Automated Material Systems



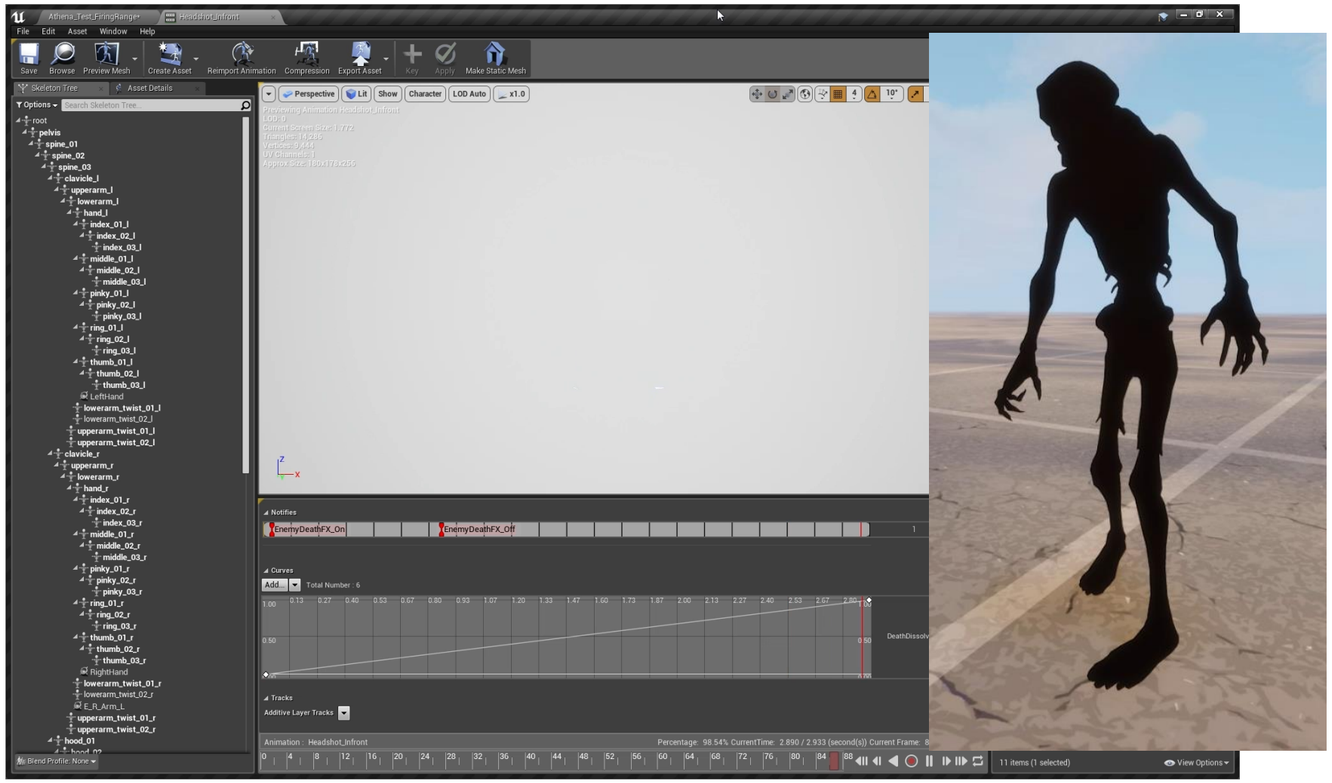
Automated Material Systems





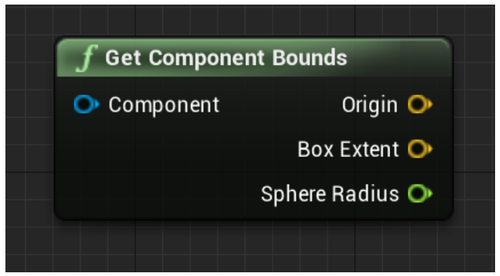
Enemy Death

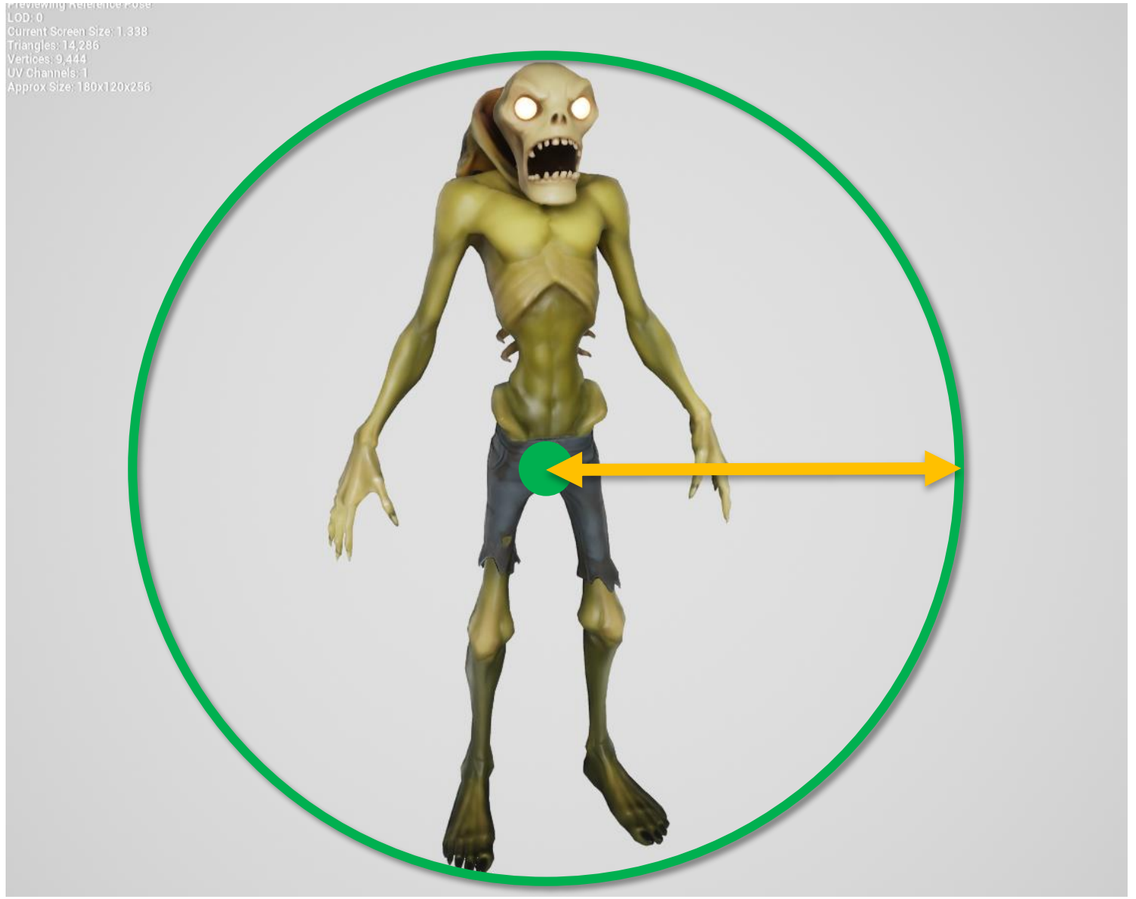
- We can use a “material curve” on any death animation to drive a 0 to 1 scalar parameter in our enemy’s material.
  - But how specifically is a 0 to 1 value controlling the dissolve?
  - *Sphere mask!* We just want to start eroding from the point of impact until the enemy is completely dissolved.
  - But going 0 to 1 gives 1 cm . . .



Enemy Death

- We can query the skeletal mesh to get its object radius.
- Multiply radius against our 0 to 1 curve.






Perspective
Lit
Show
Shaders Compiling (3065)
'DisableAllScreenMessages' to suppress
Level: Untitled (Persistent)

Enemy Death

- To fix this, we simply need to calculate how far away the impact is from the center of our mesh, then add this to our radius.
- Radius = 80
- Distance to Center = 51

- **r + l = 131**


Perspective
Lit
Show
Shaders Compiling (2355)
'DisableAllScreenMessages' to suppress
Level: Untitled (Persistent)

Caveats

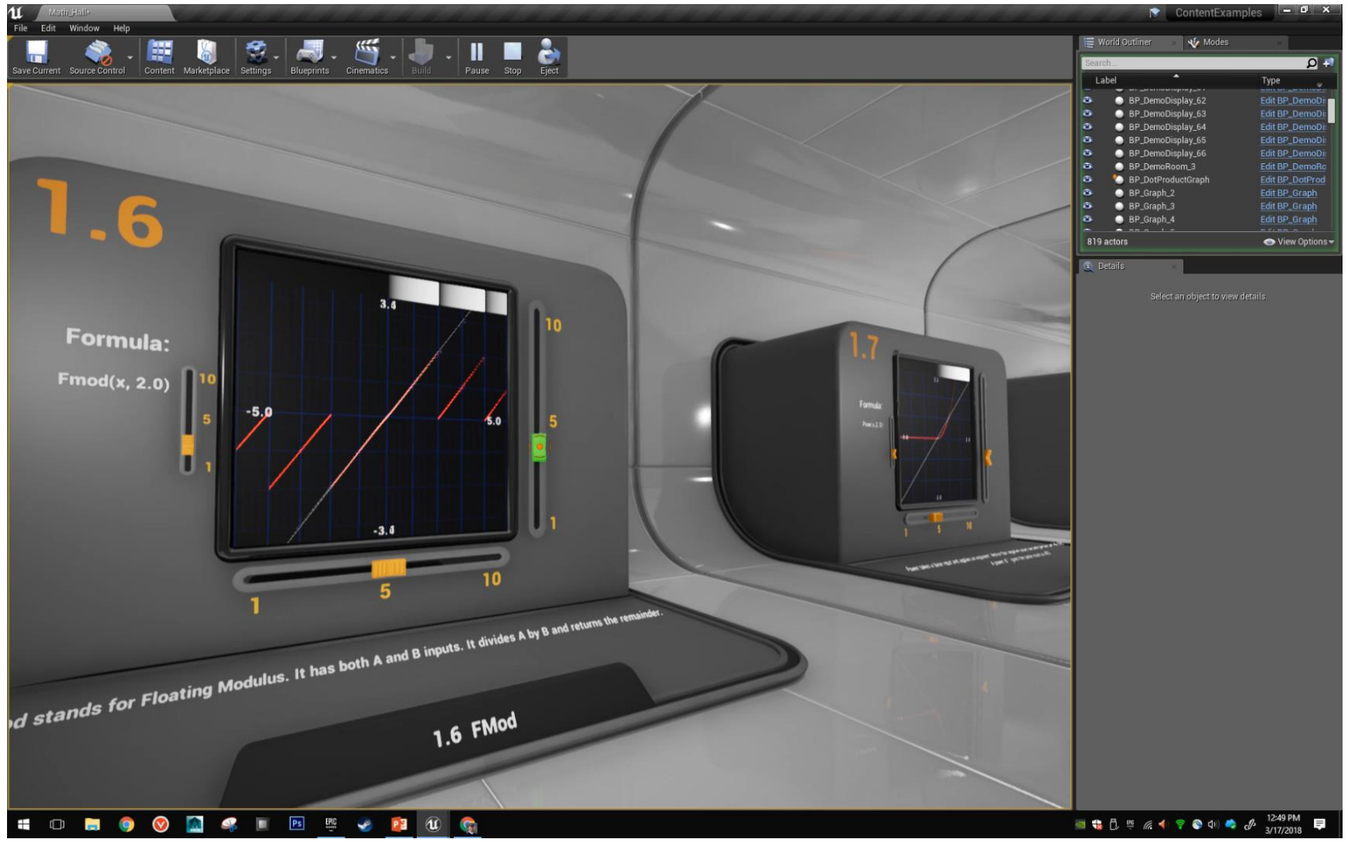
- With all this power comes the ability to break the game even easier!
- It's important to understand the difference in how a CPU works vs a GPU.
- Tick gets expensive! (calculating something every frame).
- Do keep any node based graphs clean, organized, and well commented.
- Some caveats such as keeping a reference in a blueprint, even if unused, will load the entire asset into memory.
- Beware of casts!
- Sometimes we have to get in to the world of network replication. Understand the nature of client vs. server.

Takeaways

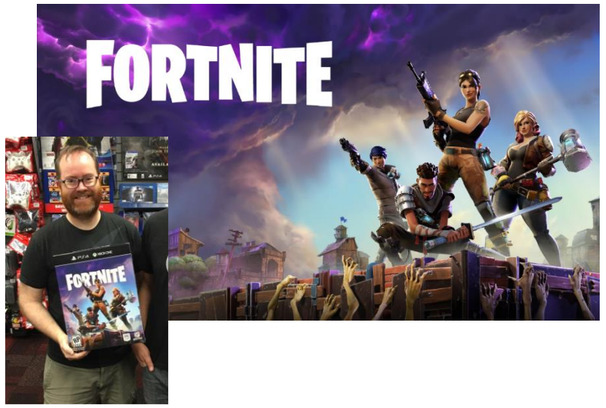
- **Realtime visual effects is more than shooting particles everywhere.**
- The VFX Artist of the future must be just as smart technically as they are talented artistically.
- The role of future VFX Artists will require them execute their own custom tools.
- We need to have a solid understanding mathematical concepts to turn our crazy ideas into awe-inspiring realities.

Learning Resources

- **Content Examples – Math Hall** – Ryan Brucks
  www.unrealengine.com (learn tab inside the launcher)
- **Linear Algebra For Games (3 parts)** - David Rosen
  http://blog.wolfire.com/2009/07/linear-algebra-for-game-developers-part-1/
- **Practical Use of Vector Math in Games**
  https://www.gamedev.net/articles/programming/math-and-physics/practical-use-of-vector-math-in-games-r2968/
- **Rendering Wounds on Characters in UE4** - Tom Looman
  http://www.tomlooman.com/rendering-wounds-on-characters/



- **Developing the Art of 'Fortnite'**
  - Pete Ellis (Art Director on Fortnite)
  - Wednesday, March 21 - 5:00 - 6:00pm
  - Room 2005, West Hall



- **We're Hiring!**
  - https://epicgames.avature.net/careers
  - 150+ positions currently open worldwide!

- **Programmable VFX with Unreal Engine's Niagara**
  - Wyeth Johnson (Technical Artist)
  - Wednesday, March 21 - 5:00 - 6:00pm
  - YBCA Theater

- **General Fortnite / Battle Royale Questions?**
  - Nick Chester, PR Manager
  - nick.chester@epicgames.com
  - @nickchester / Twitter