Weapon Effects Creation Pipeline Using Houdini and Unreal Engine

MSc Computer Animation and Visual Effects

Sarah Shahzad

August 2018

Abstract

Weapon effects are an important element in any game that uses weapons. Creating these effects has both an artistic and technical element to them. To create the desired effect, technical difficulties can hinder the artist to create to their full potential. In our paper we propose a pipeline to create sword effects using Houdini's capabilities and integrating them into Unreal Engine 4. We will make use of the new "Game Development Toolset" in Houdini to create and export the assets, using the "Houdini Engine Plugin" to import them in UE4. Furthermore, we will use the shader and particle system in Unreal Engine 4 to create and attach the effects to pre-existing animations.

Table of Contents

Table of Figures

1 Introduction

In the world of computer games these days, it is undeniable that weapons are a big part of them. Whether these are role playing games (RPG), first person shooters (FPS), Massively Multiplayer Online (MMO), or Real Time Strategy games (RTS), most, if not all, have different weapons that are a significant part of the gameplay or even the storyline. These weapons can include swords, bow and arrows, spears, axes, guns, gauntlets, magic spells or staff etc. The types are endless. However the undeniable choice of weapon in games of a fantasy setting is swords, which is what will be the focus of our effects creation pipeline.

Various effects are applied to swords in games not just for visual appeal, but also because it enhances gameplay. The effects of a weapon can depict how strong a weapon is, how fast it is, what is its range, or even what element or type it is. It visually assists the player to immerse themselves even further in the game. One key feature of a sword effect is the trail it leaves behind after it swings. This is something we will look at creating in our project. Some games famous for their weapons and effects are “Devil May Cry” with “Rebellion”, “God of War”, many of the Final Fantasy games especially “FFVII” with the “Buster sword” to name a few.

There are many industry leading software that are used in creating weapon effects, where the choice of using one over the other is based on the specific project or workflow. These weapon effects can be created directly in the engine such as Unity (Unity, 2018), or Unreal Engine 4 (Unreal Engine 4, 2018), or from external software such as Maya (Maya, 2018), Houdini (Houdini, 2018), 3DSMax (3ds Max, 2018), Blender (Blender, 2018) etc. For our project, we will be using Houdini and Unreal Engine for our purposes. Houdini is not new in a games pipeline, but recently it has been spilling more into the games studios as agreed by Andreas Glad in an interview (CGSociety, 2018). There are not many examples of weapon effects created through Houdini into UE4, which is what we will explore in our project.

2 Previous Work

Creating weapon effects is a versatile area. Due to countless weapons and the effects that can be applied, creating them is a work of art. Though to create these effects there are a lot of technical aspects behind them. For an artist to fully realize what they are meaning to create, these technical aspects need to be simplified and streamlined to make the process as

smooth as possible. Weapon effects can be made directly in an engine or bigger companies have their own in house tools to do so. These can be coded directly, depending on the pipeline, or used with external software and integrated with the engine. Unity has many examples of weapon effects being created directly in it using its particle system, such as a sword slash effect (imn nam, 2016), an ice attack (Gabriel Aguiar Prod., 2017), or an electric sword effect (Mirza, 2017). Unreal Engine also has a few examples of creating weapon effects such as sword trails (DV7 Pavilion, 2016), or gun shots (Smyke, 2016). Some of the famous games known for being created in UE4 are Ark Survival (Ark: Survival Evolved, 2017), Sea of Thieves (Sea of Thieves, 2018), and Hellblade (Hellblade: Senua's Sacrifice, 2017) to name a few.

Houdini in the games industry is becoming more and more popular, though it still stands out the most for its node based VFX capabilities. These can be use in games to create destructible environments or dramatic weather effects. (Bannink, 2009). Since Houdini is a powerful tool when it comes to creating procedural content, more games have been utilizing it in its pipelines. Some these examples are Horizon Zero Dawn for its procedural rivers and wires (80 Level, 2017), and Ghost Recon: Wildlands for its procedural world (80 Level, 2017). With the recent game development tool set in Houdini, and the Houdini Engine plugin for UE4, transferring assets and content for Houdini to UE4 has opened a way for a new pipeline for games.

Some of the tools that we will be looking at are the “Houdini Digital Asset”, the “Vertex Animation” tool, and the “Vector Field” and “Flow maps” tool. For our project we have decided to use UE4 for our purposes, and have created a pipeline that utilizes the capabilities of both Houdini and UE4 to create sword effects.

3 Technical Background

3.1 Houdini

Houdini is a 3D animation software, that has a node based procedural workflow. It is best known for creating non-destructive workflows and strong fluid simulations such as smoke and pyro effects. Within Houdini, networks can be saved into a single node called a Digital Asset (HDA). These HDA can be set up to have control over parameters to make alterations to the variables within the network contained. Having this feature makes the networks portable in different projects, and also into

other software. This feature is key factor for our project as we will be looking into creating an HDA to import into UE4.

Houdini has introduced a new toolset for game development. Two of the tools that it features that are significant to our project are exporting vertex animation and vector fields. Vertex animations are key framed animations of positions of the vertices of the mesh. In complex simulations such as water, cloth or smoke, the computation time is expensive and such simulations cannot run in real-time in games. This is when vertex animations come into play as the calculations are already performed and the positions are prebaked. We will look at using this for adding effects to our weapons. And finally, vector field in a 3D space is a function that defines the points in the filed with a vector. This vector is essentially the direction a particle should go when it enters that space. These fields define the flow within a given space. In our project, we will make use of these vector fields with UE4 to control the flow of our particles in a specific way, which we cannot created from the engine's particle system alone.

The version used for our project is Houdini 16.0.671, student licensed. This student license or above is required to use the Houdini Engine plugin in UE4. It should be noted that there is a "Games" shelf within Houdini that has some of the same tools as the "Game Dev Toolset" that is installed separately. The vertex animation tool version 1.12 has errors at the time of the creation of the project and therefore the inbuilt shelf VA tool has been used.

3.2 Unreal Engine

Unreal Engine 4 is an industry leading games engine created by Epic Games. It has integrated tools to help developers to design and build games. For the purposes of our project version 4.16.3 has been used. Some of the following terms will be used in the paper. These are the names of the different tools or editors within UE4 and are briefly explained below.

3.2.1 Blueprints

Blueprint in UE4 is a visual scripting tool. It provides a node based approach for programming and setting up scripts that define the gameplay. Actors or assets can be attached to blueprints, whose initial variables can be defined in the viewport. Different assets can be attached to one blueprint and set up as a class or prefab. There are different types of blueprints. The two our project will make use of are the character and actor BP, and the animation BP.

3.2.2 Material Editor

The UE4 material editor provides a platform to create shaders within the engine. This is a powerful editor that can create from simple shaders with just colour value to complex shaders that can animate a fluid simulation. This editor is also node based. For our project we will be using the material editor to create our shaders rather than creating them in Houdini. This was decided to keep the implementation and transfer of the Houdini asset as simple as possible.

3.2.3 Cascade

Cascade is the editor that handles UE4's particle system. This editor has a few components in it as shown in figure 1. On the left of the editor is the viewport, where we can see the results of our emitter settings. On the right is the emitter panel. This is where all the emitters required are created. The emitter itself has modules that control a set of variables associated to that module. These variables can be seen below the viewport in the properties panel. Different modules can be added by right clicking on the emitter. The type of the emitter can also be selected this way. Below the emitter panel is the curve editor. The curve editor for each module can be accessed by clicking the green curve button next to its name. Cascade will be useful in our project when we integrate the vertex animations.

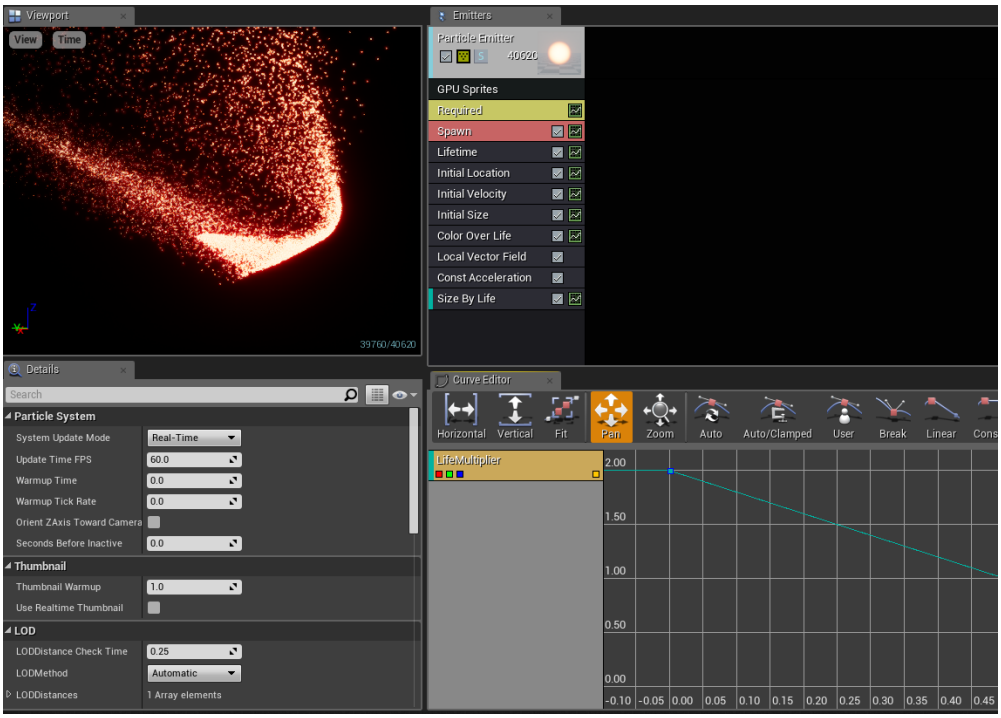


Figure 1: Cascade- UE4 Particle Emitter Editor

3.2.4 Sequencer

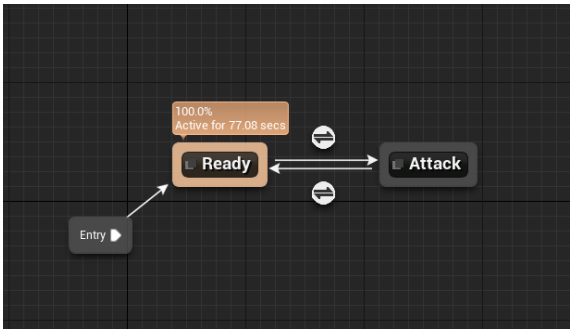
A level sequencer can be created in UE4. The purpose of the sequencer is to create cinematics and cut scenes for a game. It allows you to render and export shots or entire cinematics. However for our purpose we will be using it differently. The sequencer for our project would be used as more of a visualizing tool. To use it, actors that are already in the level can be added to it and the values of the variables associated with that actor can be keyed, such as transforms or the parameters of the material attached to it. There is a timeline and a curve editor in the sequencer tool that can be scrubbed through. The animations created by these set keys can be seen in the viewport without the need to run the game.

3.2.5 Animation Editor

The animation editor is associated with animations on an asset. The animations on the asset can be seen in the viewport. The bones or skeletal hierarchy of the asset can be seen on the left panel, and sockets can be created and attached to these bones. The best use of an animation editor is the use of “notifies”. The notify section in the animation editor has a timeline. On this timeline a “notify” can be added that can launch a particle emitter or execute timed events within the animation blueprint. Most of the “notifies” created in our project are launched using animation blueprints.

3.2.6 Setting up Character

Before we start discussing our proposed pipeline, the project was set up with two assets with animations on them. We will use these assets as a means to test our methods. The first asset is a character with two animations imported with it. To set this up an animation BP was created with an “Idle” and “Attacking” state, and a variable “Is attacking?” to define the rules. Figure 2 shows the state machine and the animation event graph. A character BP was created with the character mesh and the newly created animation BP attached. The “attacking” variable is added to the character BP to work with the anim BP script.



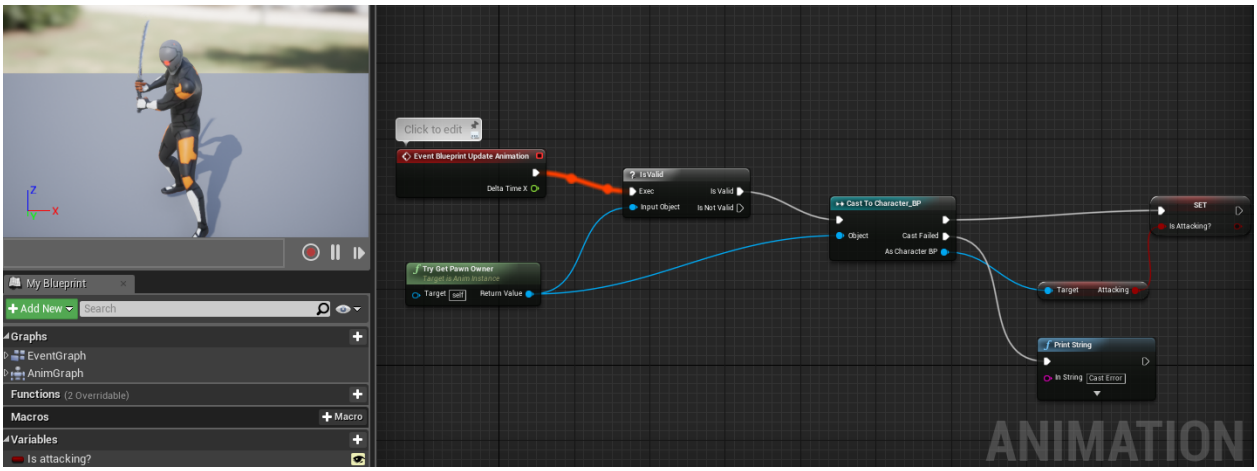


Figure 2: Character Animation BP.

Top – *State Machine*. **Bottom** – *Anim Graph*

The second asset is a sword with no animations but is imported with a skeleton. Due to this a simple animation is created within UE4 in the animation editor. In this editor there is a section for "tracks". From the left, select a bone to animate and press "Key" on the toolbar at the top. Doing this will add curves for the transforms of the bone. These curves can be keyed to create the animation as shown in figure 3. The rest of the setup is the same as the character.

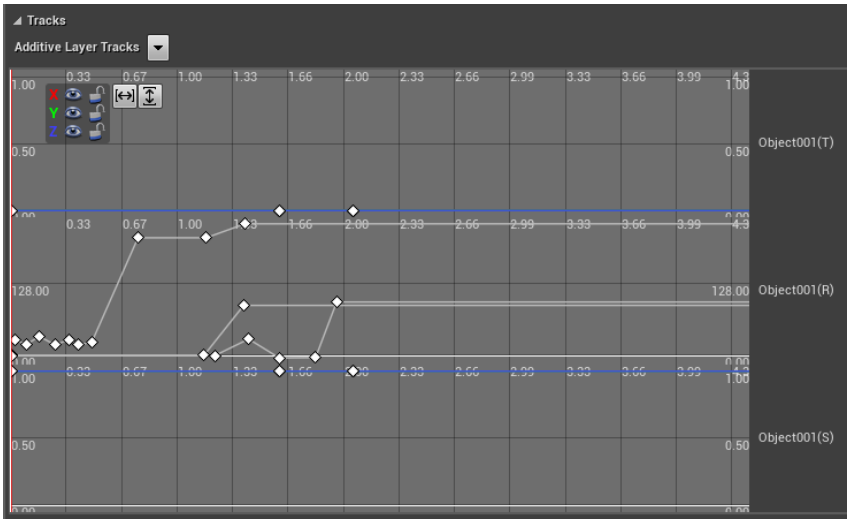


Figure 3: Creating Animation in UE4 Animation Editor

There was in issue in our project when adding input from the character/sword BP. Due to time constrains, and also because it is not a

core part of our project, the issue was not solved. Therefore the input is handled from within the level BP as shown in figure 4.

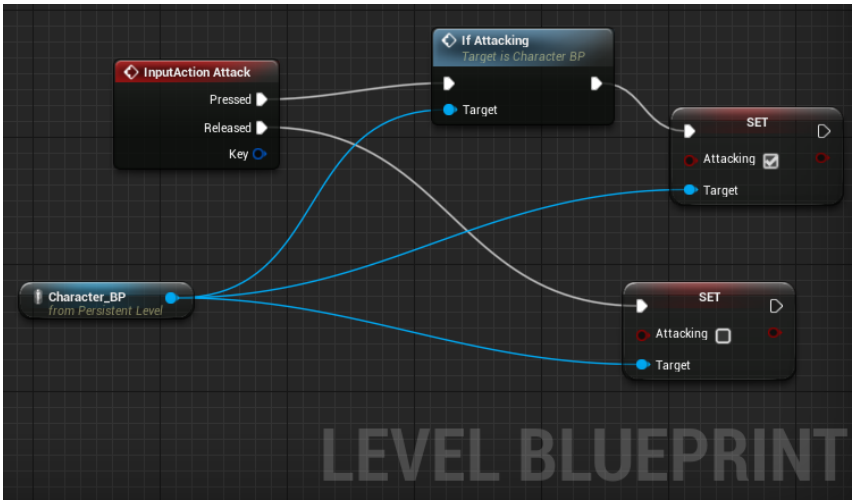
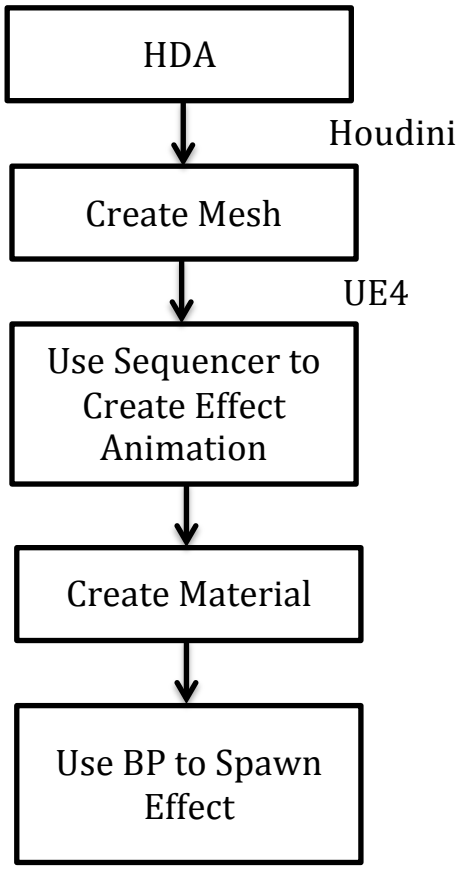


Figure 4: Level BP for Attack Inputs

4 Proposed Method

4.1 Using Houdini Digital Asset (HDA)

Flow chart of the steps:



4.1.1 Creating and Exporting the HDA from Houdini into UE4

Houdini can be used to make any network into an HDA. In our example we will look at creating a base mesh that will help in shaping the sword trail for a pre-existing animation in UE4. The design for the trail mesh HDA is at the discretion of the creator and therefore can be different than our example. The Curve Sweeper tutorial by Andreas Glad (SideFX Houdini, 2017) was used for our project as it was simple and easy to create, and is perfect to implement in this pipeline. Figure 5 shows the network for the trail asset and the controls available. It takes a curve input that will be defined in UE4. Once the network has been created it is converted into an HDA and saved. In UE4 this HDA can be imported like any other asset, provided the plugin has been installed and a licensed version of Houdini is being used as mentioned before in section 3.

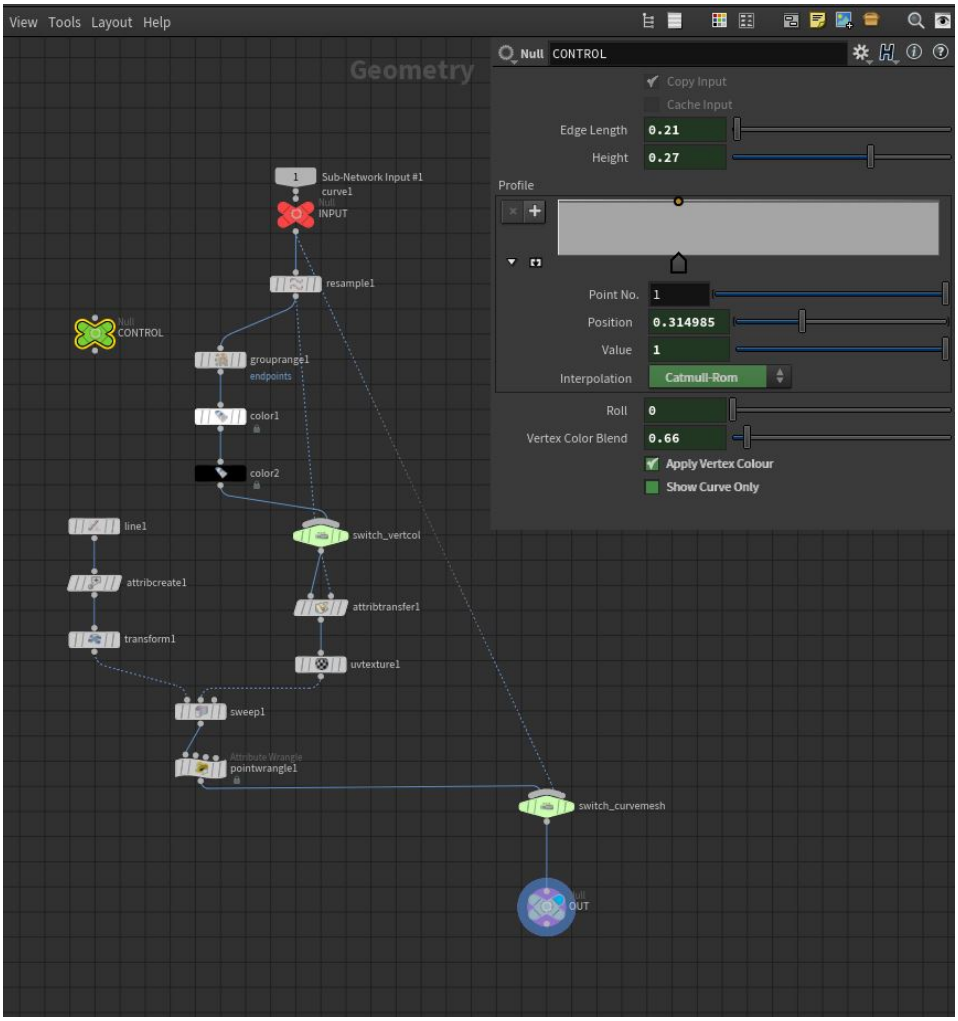


Figure 5: Curve Sweeper Network in Houdini

4.1.2 Creating the Trail Mesh Asset

Once we have the trail mesh imported into UE4 we can use it to create a trail for our sword animation. In our project we have used the Gray Fox model and animation provided by person-x (Gray Fox, 2017), and set it up so that it has an animation BP for “Idle” and “Attacking” states. And we have also created a character BP with the animation BP attached. (Refer to section 3.2.6) The character attacks on a key press. This animation provides a good example of a more complex animation for the trail effect. We have also used the Azure Sword model provided by sinchik97 (Azure Sword, 2017), and set it up similarly to the character BP. However since this model does not have any animation, we have created a very simple animation for it. (Refer to section 3.2.6)

To create the trail, drag the HDA into the viewport. In the details panel we can see all the controls we had set up in Houdini. In the “Houdini Input” section, change the input from “Geometry” to “Curve” and change the type to “Nurbs” as shown in figure 6.

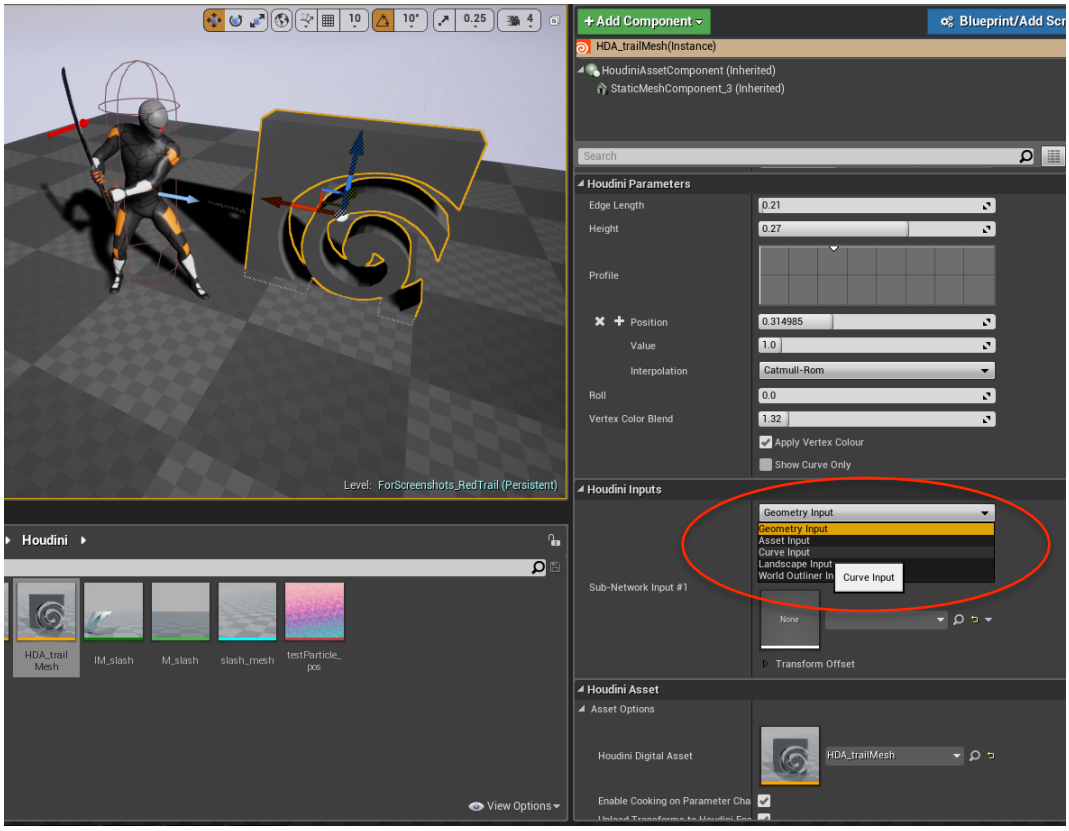


Figure 6: HDA Curve Input

The HDA changes into two points for a curve that can now be used to create a curve for our trail mesh. Pressing ALT and dragging on the last point adds

more points to the curve. Make sure the “Show Curve Only” option is turned on while drawing the curve. Before we can go about creating the curve, we need the movement of the animation as a reference. As the trail effect is being added onto an animation, using UE4’s sequencer would be the best way to visualize the animation as we create our curve. For this we need the animation asset in the level. Create the level sequencer, open it up and add the actor the sequencer, choosing the animation we have in the level. Add an animation track for the animation we are creating the curve for. Using the timeline we can now scrub through the animation and create our curve as shown in figure 7.

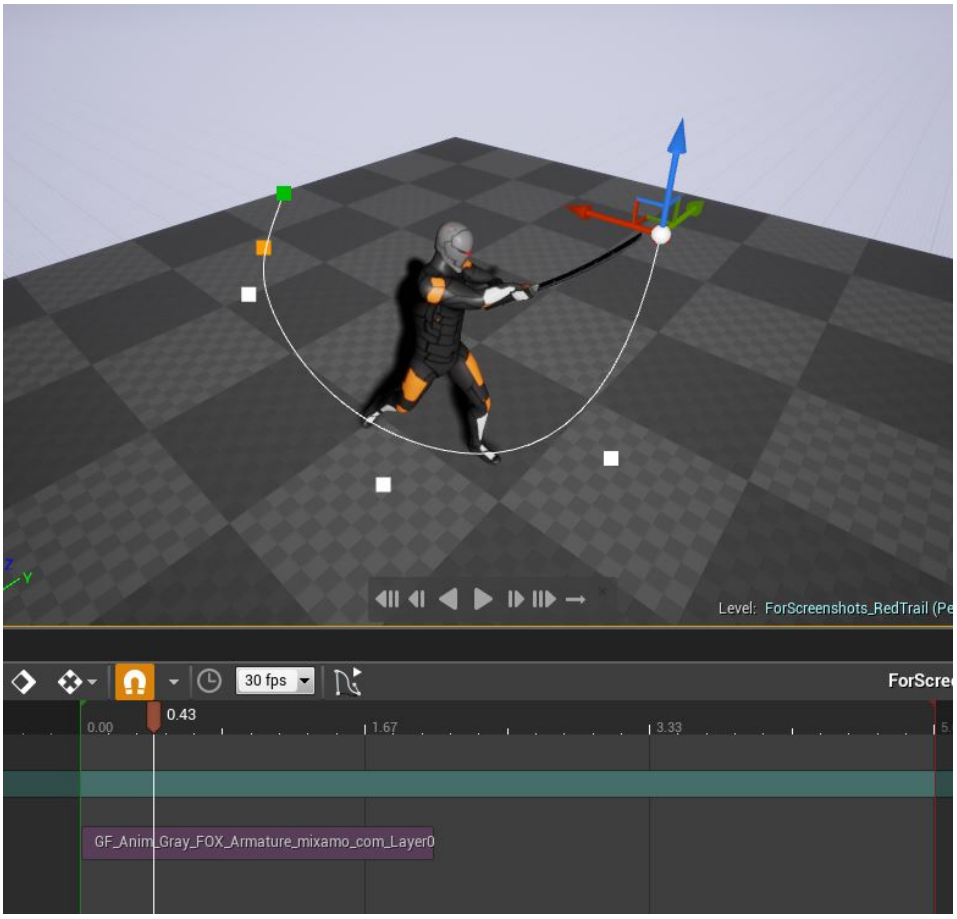


Figure 7: Create HDA Curve

Once the whole curve is created the mesh can be turned on and modified using the other settings to shape it how we want it to. Now that we have our HDA mesh, a material is created for this trail. It is important that the trail materials have parameters that can be controlled to animate the effect, esp. a panning parameter with transparency on it. The trail materials have been explained in section 4.4. Apply the material to the HDA and add it to the sequencer. Add tracks to the parameters that need to be animated. In

this example we will look at the red trail material, so we will add the location and colour track. Set keys to the parameters values while scrubbing through the timeline to create the desired animation as shown in figure 8.

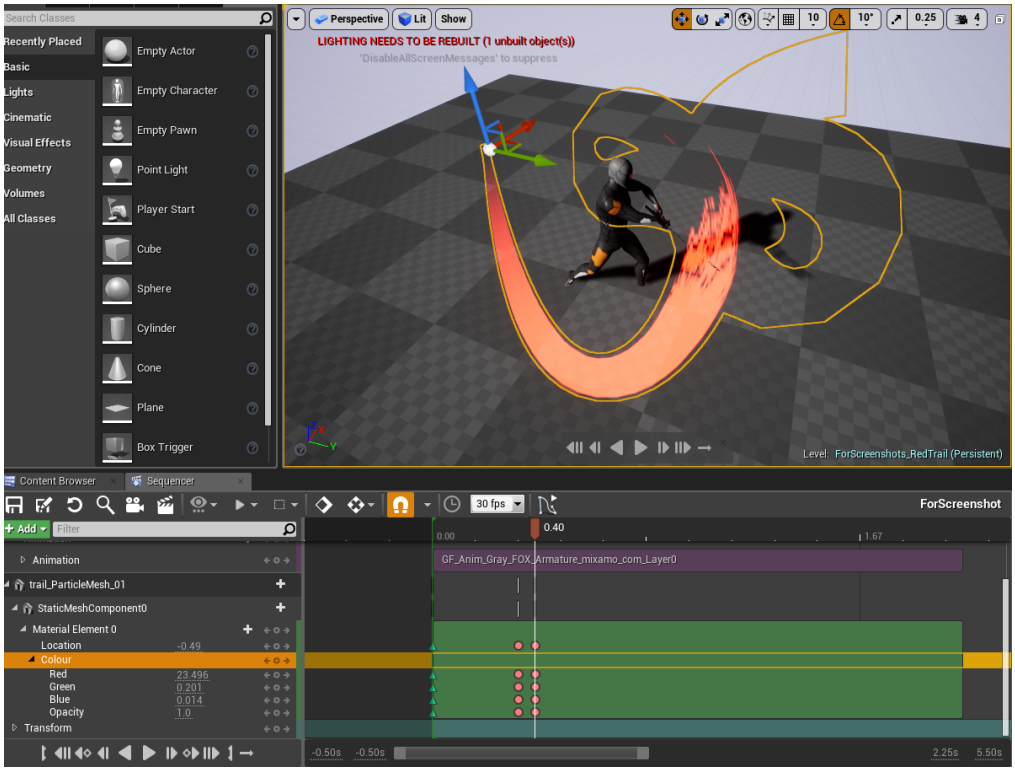


Figure 8: Animate Trail Material Parameters in Sequencer

By this point we have an idea of what the effect will look like. Since we have our HDA mesh and material, we can bake this asset into a static mesh that can be used anywhere. The save location for the static mesh is defined in the "Houdini Asset" tab, and the bake options are in the "Houdini Generated Meshes" tab in the details panel. This static mesh with the material is what we will spawn in game during the attack. The HDA and the character animation in the level are no longer needed and can be hidden, but keep them in the sequencer. From here we have two methods that can be used to spawn and animate the trail mesh. Both have their pros and cons which we will discuss later.

4.1.3 Spawning and Animating the Trail Effect (Method 1: Animating in BP)

At this point we have everything we need to add the scripts to spawn our trail effect during an attack and animate the materials accordingly. This is done using two scripts. One script is to spawn the effect, which is done in the character BP, and the other to animate the trail material, which is done

in the trail BP. The trail BP can be set up simply by creating an actor BP and attaching the trail mesh to the root node. In the character BP Event Graph the script shown in figure 9 is added.

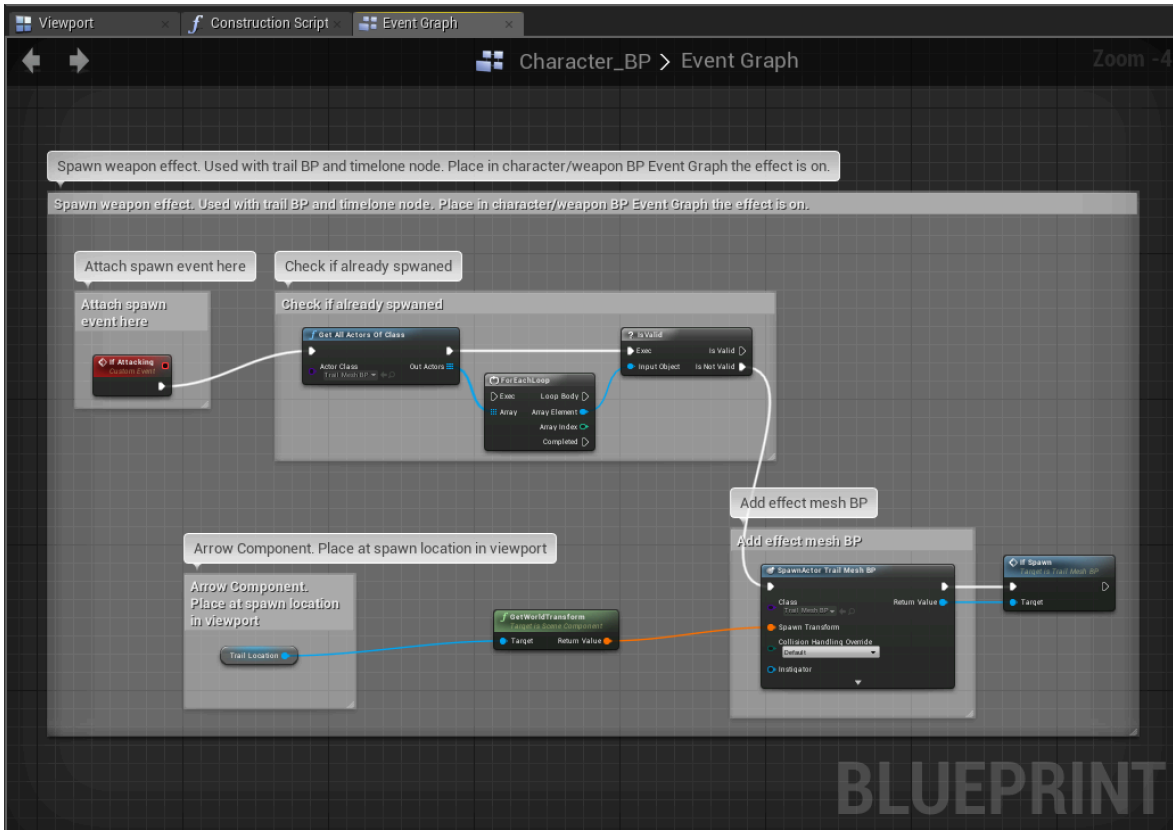


Figure 9: Script in Character BP to Spawn Trail Mesh

The “spawn event” is when the script should be launched. In our example the custom event “If Attacking” is launched on a key press, which also animates the attack animation (Refer to section 3.2.6). Then it checks if the effect mesh has already spawned. This optimizes our gameplay as we don’t want multiple instances of our effect, which also prevents the effect overlapping from multiple key presses. When this script is used in another blueprint, the “Actor Class” needs to define the trail BP specific to the one that has to spawn. The “trail location” is an arrow component that has been attached to the character mesh. A tutorial by TorQueMoD (TorQueMoD, 2015) was referenced to define the position of where to spawn the trail. The transformation of this arrow component defines the transformation of the effect. Position this arrow at the spawn location of the trail in the blueprint’s viewport, which for this example is the sword’s tip. Next, spawn the trail BP at the position of the arrow component and launch “If Spawn”. “If Spawn” is a custom event in the trail BP. Now that the trail has been spawned, it needs to animate. Figure 10 shows the script in the trail BP.

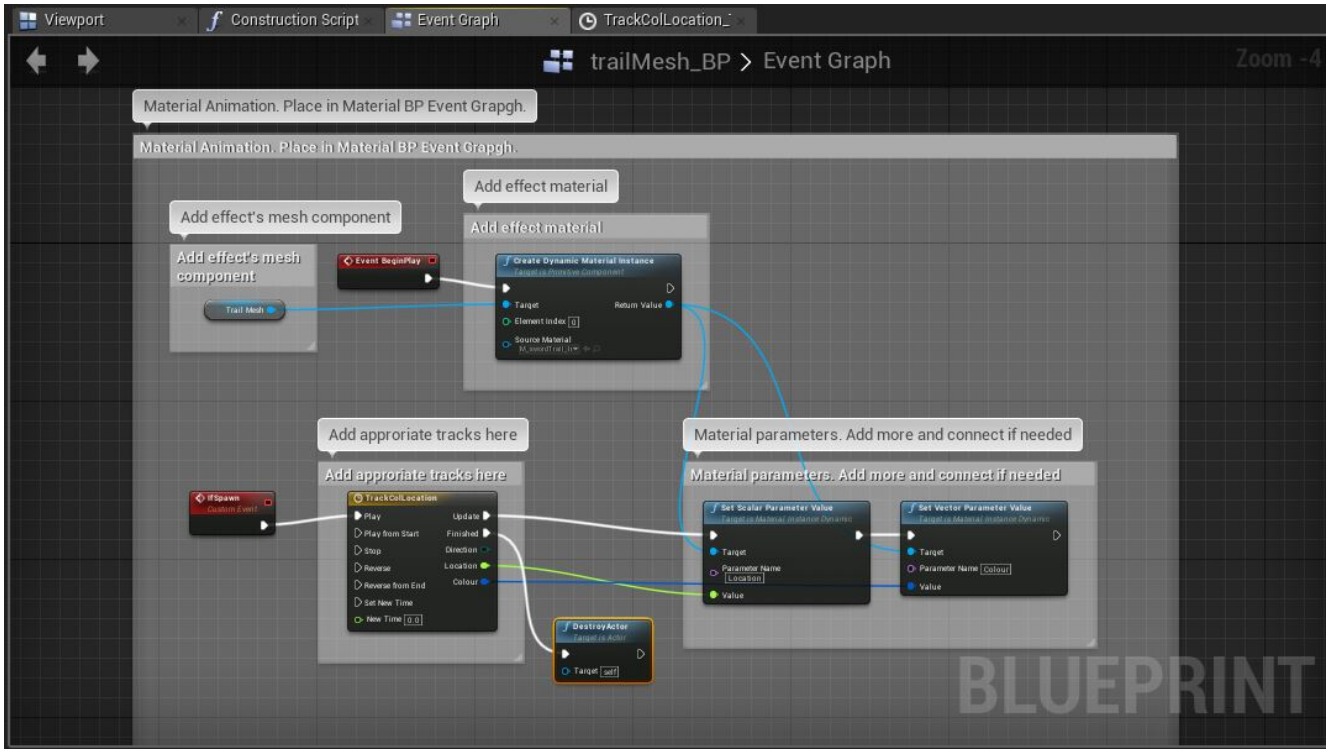


Figure 10: Script in Trail BP to Animate Trail Material

The script creates a "Dynamic Material Instance" that allows us to modify the parameters of the material specified. Set the source material to the trail material and connect the return value to the parameters that have to be animated, as shown above. These can be either scalar or vector parameters. The "Timeline" node is where the material parameter values update along with time. A tutorial by Dean Ashford (Ashford, 2017) explains how to create dynamic materials in a blueprint using the timeline node. Open up the timeline node and add two tracks, one for "Location" and the other for "Colour". Set the time range exactly as the time range for the attack animation. This can be checked by opening up the animation asset and copying the total time. This is important to keep the animations in sync. This is where the animation we defined in the sequencer comes into play. Open up the sequencer, and from the curve editor, manually copy the keys over to the track curves. Figure 11 shows the completed curves. Once the curves have been completed, connect the corresponding node to the parameter value node. Place the Character BP in the level, and the trail effect will show upon attacking.

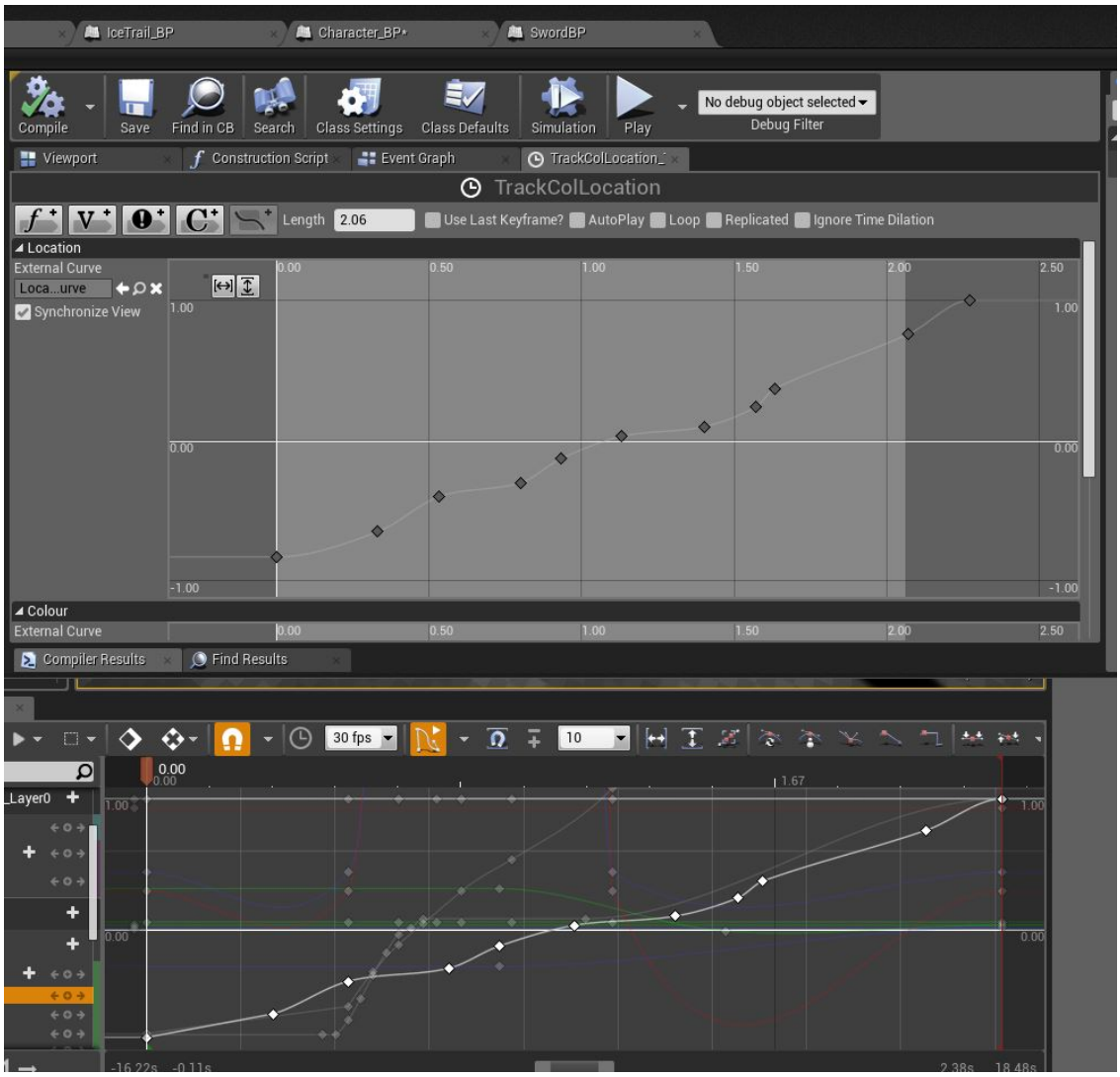


Figure 11: Copy Sequencer Keys to Timeline Curve

4.1.4 Spawning and Animating the Trail Effect (Method 2: Animating with Sequencer)

The second method is easier and less manual like the first. After creating the animation in the sequencer and the static trail mesh, create the trail mesh BP as mentioned earlier. Drag this blueprint into the level scene and add it to the sequencer. In the sequencer, track the “location” and “colour” of the mesh component as it was done for the HDA material. Select all the keys in the HDA animated material and copy them over the trail BP tracks we just added as shown in figure 12. We want the sequencer to spawn this trail wen it is playing and destroy to when the animation is complete. To do this, right click on the trail BP added to the sequence and convert it to “Spawnable”. A new variable “Spawned” is added to the tracks. Key this as enabled at the start and disabled at the end.

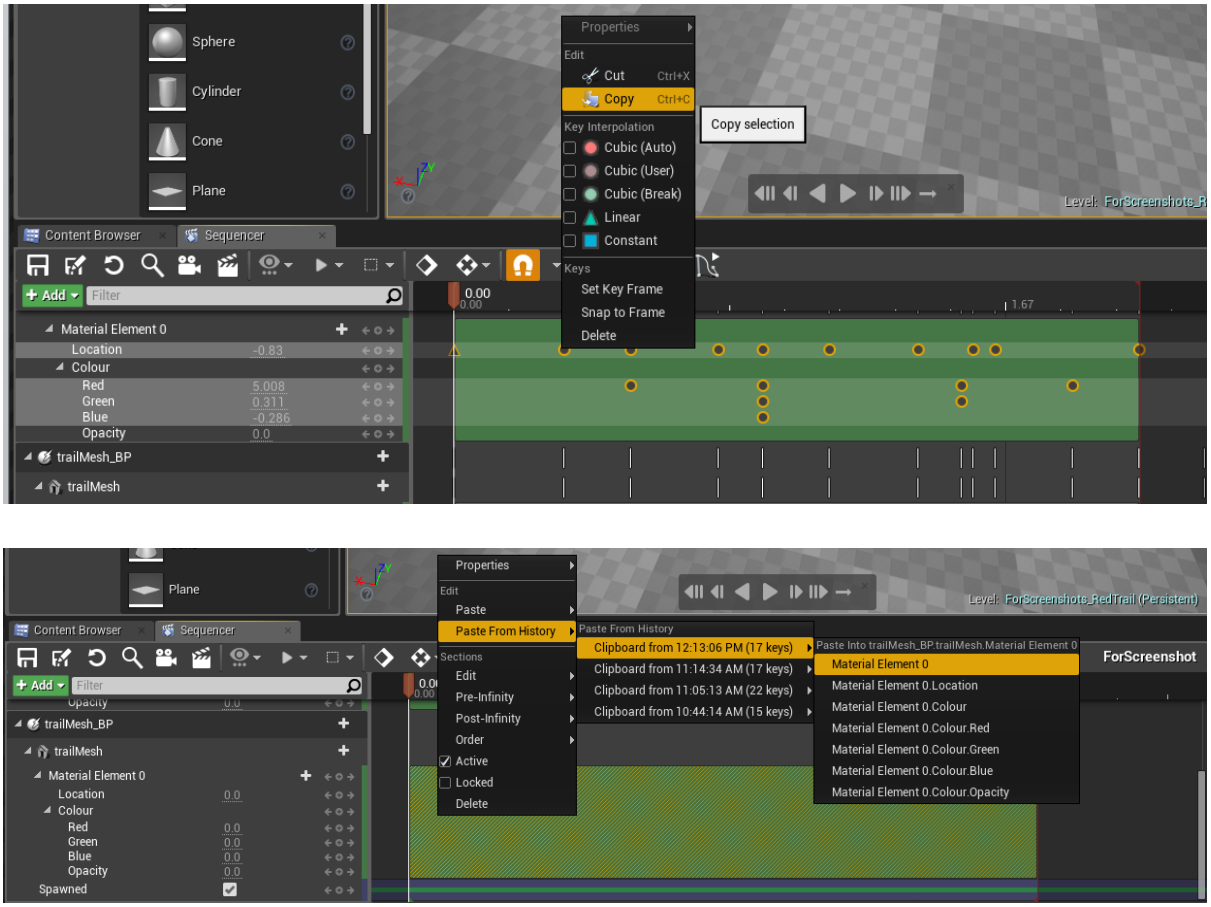


Figure 12: Copy Sequencer Keys to Trail BP Tracks

Only one script is required in this method. This script (figure 13) is added to the Character or Weapon BP. The level sequencer can be deleted from the level as we will be spawning it from the blueprint. In the blueprint we first check if the sequencer actor is already created. We need to do this so that we do not spawn a new actor every time we attack. If the sequence actor does not exist in the level then create it, otherwise play the animation. Next we need to check if the trail exists in the level. If it does then use the arrow component for the trail location and set the transforms of the trail BP as we did previously for spawning the trail BP in method 1 (Refer to section 4.1.3). Note that for the location to be updated the transform tracks in the sequencer for the trail BP have to be deleted as those values are given preference. Now if we attack we can see the trail effect. Figure 14 show the three trail effects created for demonstration.

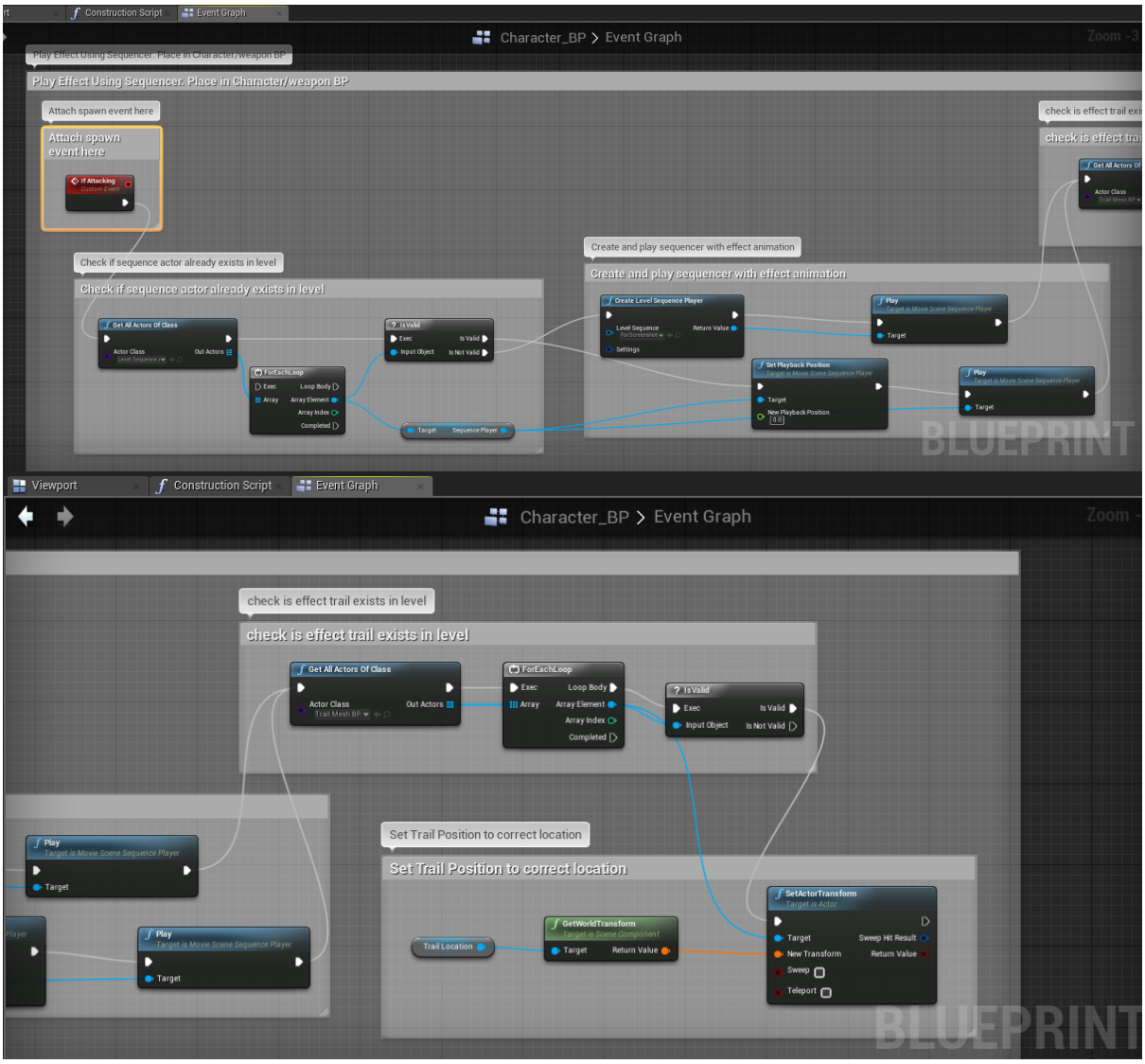


Figure 13: Spawn Trail Effect Using Sequencer in Character BP

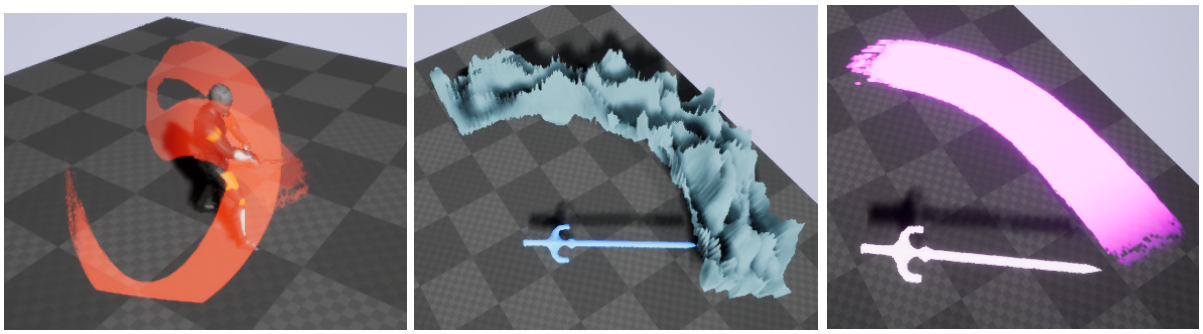


Figure 14: Completed Trail Effects.

Left - *Red trail*. **Middle** - *Ice trail*. **Right** - *Purple trail*

4.1.5 Analysis

Using Houdini to create the trail mesh has its own benefits as opposed to creating the trail in UE4 itself. If the trails were to be created in UE4 they have to be connected to two sockets to define the start and the end of the trail. The trails particle emitter is launched with an “Anim Trail notify” and is timed. There is an option to add a curve to the trail to modify its shape slightly; however it is limited as compared to the HDA and cannot be view in an interactive way. The material on the UE4 trails cannot be dynamically animated as easily as the flexibility available while animating it from the blueprint. Since the mesh is already created, changing the material can transform the trail into a completely different effect.

In method 1, although the sequencer is more interactive visually, copying over the keys to the timeline node is a slow, manual process. It would be easier if the curve from the sequencer could be exported and imported into the timeline, however the UE4 API does not provide any way to read in those values yet. Hence if a quick trail is required without any complicated animation or material, creating it directly in UE4 is a better option. However using an HDA trail gives more artistic flexibility. Upon multiple keypresses, the script prevents the effect to spawn multiple times while still playing.

In method 2, the trail BP can be created and animated directly in the sequencer without having animate the HDA and to copy keys, but for that then we have to position the trail mesh at the precise location we created the HDA to have the sword follow the curve as designed. This prevents it from staying in sync. However if the curve is not complicated and easy to set then creating the trail BP first and then using the sequencer can work just as well. One major issue with using the sequencer method is that the script does not prevent multiple key presses and on every input the effect restarts.

When copying the scripts into other blueprints, some of the nodes may need to be deleted and recreated. This is due to UE4 not recognizing that the nodes belong to a new actor of the same type. Hence some connections need to be refreshed. It should also be noted that the variables also need to be created and replace with the same ones.

4.2 Using Houdini Vertex Animation (VA)

4.2.1 Creating and Exporting the VA from Houdini into UE4 for a Looping Effect

Any animation that has been created in Houdini can be exported as a vertex animation for UE4. In our example we will look at creating a "Swirl" effect, which is just a bunch of particles following a curve. The tutorial by MIX Training (MIX Training, 2017) helped in creating this effect. Figure 15 shows the network for creating the swirls animation.

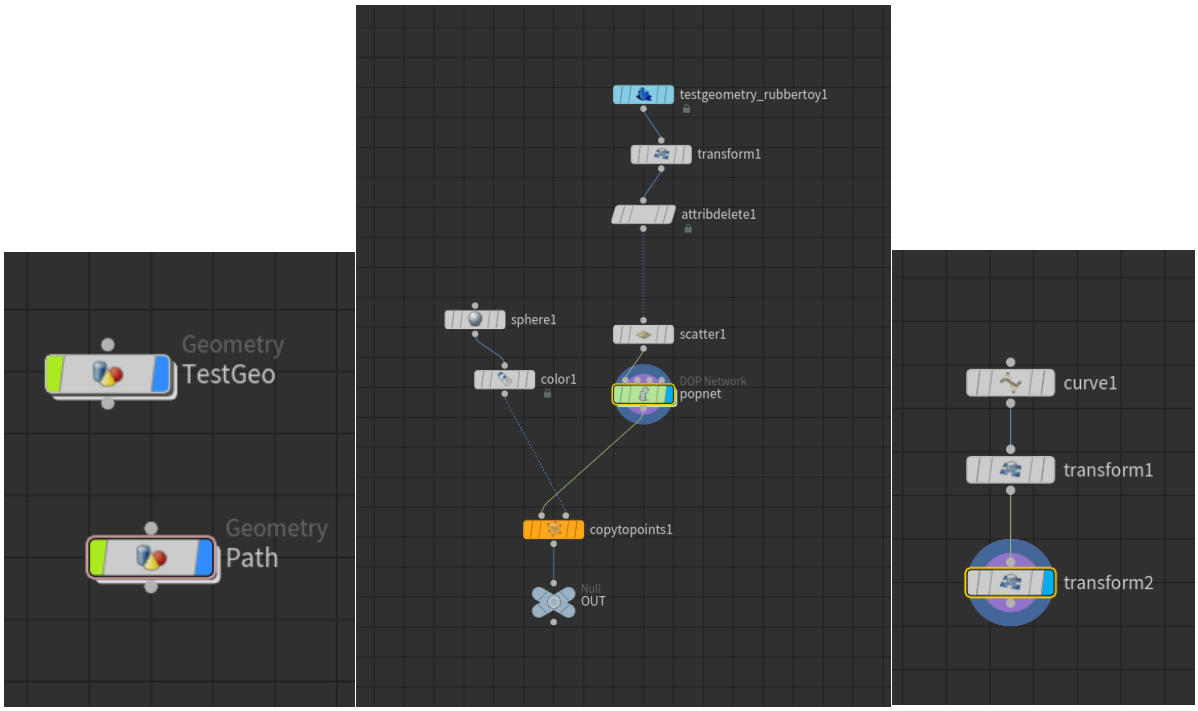


Figure 15: Swirls Vector Animation Network in Houdini

Left - *Object Level*. **Middle** - *TestGeo Network*. **Right** - *Path Network*

Once the animation is ready, in the game dev toolset, select the "Vertex Animation Textures" tool. In the "out" level the vertex animation node is created. In the parameters of this node, set the "Export Node" to the OUT node of out animation. For our swirls animation we will be using the "Sprite" option in the drop down. Set the path for the geometry and the position map and render. There are options for real time render in the parameters which we will need inside UE4 when we set up our vertex animation.

In UE4, import the FBX mesh and the position map that has been exported from Houdini, and then create a material. In Houdini, in the VA node

parameters, there is a “Sample UE4 Shader Code” section (figure 16). In that section copy everything in the “Sprite UE4 Code” and paste it in the material just created. Follow the instructions on the nodes and complete the network. This material can be modified to change colour or opacity etc. Next create an instance of the material, and in the settings replace the position map with the imported position map. The min and max bounding box values need to be replaced with the values mentioned in the Houdini VA node parameters, as well as the number of frames. Apply this material to the mesh and drag it into the level. And we can see our animation successfully set up and playing in a loop.

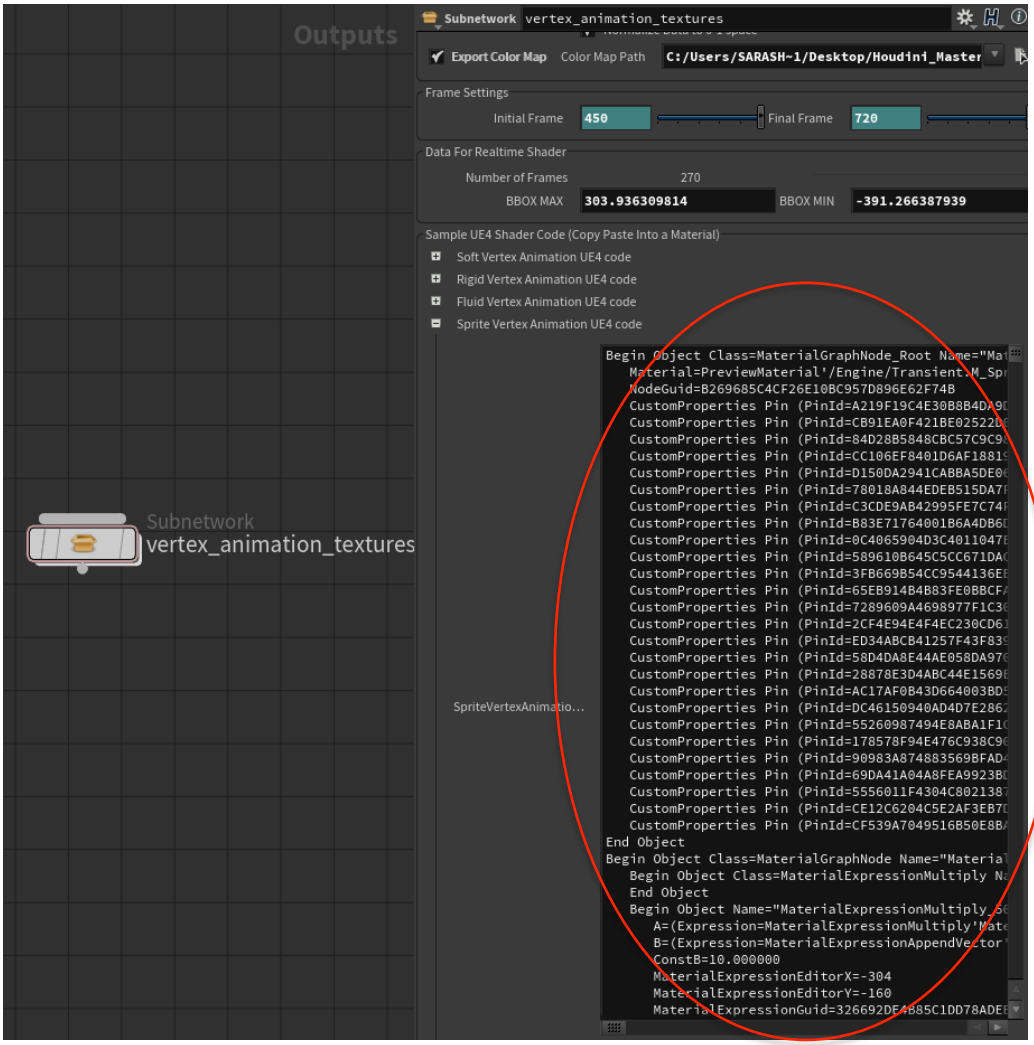


Figure 16: Vertex Animation Node Parameters

4.2.2 Using the VA as a Constant/Looping Effect

Having the Swirl animation set up, we will use it to attach it to the sword and play constantly. To do so the sword needs to have a skeleton. Open up

the sword skeleton and create a new socket and position it where the animation needs to spawn from. In our swirl animation case, it is positioned at the center of the blade. Right click on the socket name and add a preview mesh, choosing the swirl mesh. Ideally this would allow us to visualize what the swirls would look like on the sword however, the position transforms of the mesh do not update with the socket position in the preview. This is because of how the position of the asset is calculated by the material. An offset to the material can be added to modify this, which we will look at later. The scale and rotation of the asset can be set using the socket. Once the socket transforms have been set to where they are needed, we can now connect it to the actual socket rather than just previewing it. Due to the same problem with the transforms, the VA doesn't follow the sword's animations if it is simply added to the sword BP and connected to the socket. Therefore we will add a script to the sword's event graph to spawn the effect on startup, and update the positions of the effect according to the socket position. The script is shown in figure 17.

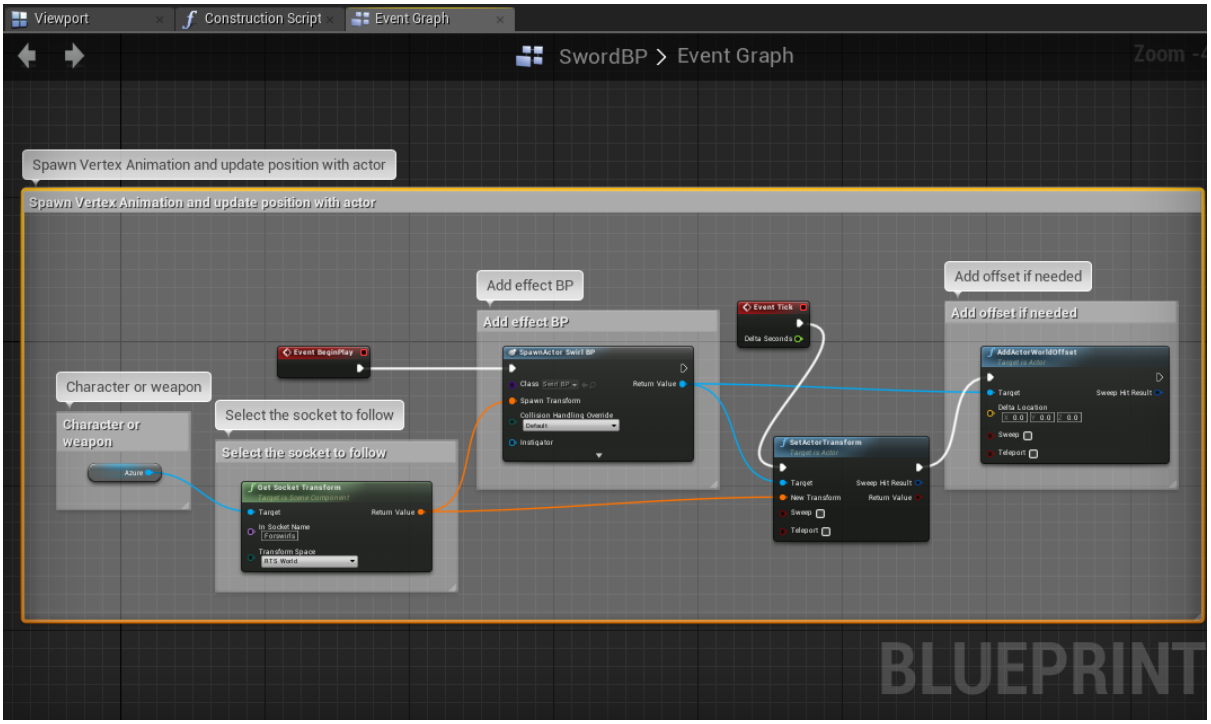


Figure 17: Looping Vertex Animation Script in Sword BP

The script uses the weapon attached and gets a socket attached to it. The name of the socket needs to be defined, which is the one we created for the swirls. Create an actor blueprint and attach the swirls to create a swirl BP. In the spawn node, add the swirl BP in the class to spawn it. Next set the actor transforms and connect the transform node of the socket to it. Update this every frame to follow the socket location during animation. Since our

swirls are centered, they do not need to be offset from the sword socket, however, if the animation needs to do so a world offset node is added.

4.2.3 Creating and Exporting the VA from Houdini into UE4 for a Timed Effect

Using vertex animation for a timed effect is slightly different than a constant effect; however exporting it is exactly the same as before. To give an example of a time effect, an ice attack has been created to spawn and animate at a particular time during an animation. To create the ice attack animation a tutorial by Mehdi (FX HIVE EVOLUTION, 2016) was partially followed. Figure 18 show the network created. In the VA export setting, we are using the "Fluid" method and shader. Export, import, and set up the VA asset as before.

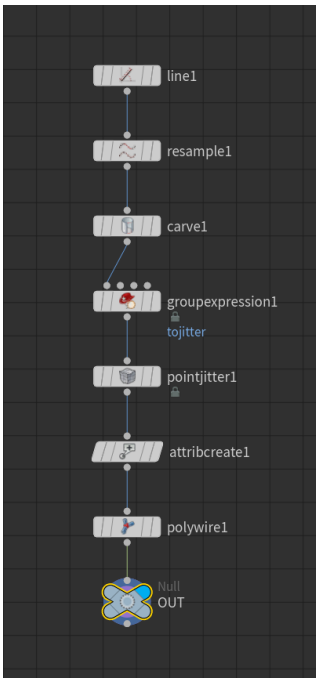


Figure 18: Ice Attack Network in Houdini

4.2.4 Using the VA as a Timed Effect

The VA is set up the same way as mentioned earlier. However this time we'll have to make modifications to the material and spawn it using UE4's particle system. Create a particle system for the ice attack and open it up which will take you to cascade. Set the emitter type data to a mesh emitter and the mesh variable to the VA mesh and its material.

The animation right now is constantly looping because of a "Time" node in the material. This time node constantly updates which means that the animation start time will not be when spawned. We need some way of controlling this so that when the emitter is spawned, the animation plays

from the start and does not loop. To do this, a “Dynamic module” is added to the emitter in cascade. This module is where we will control the time or panning of the animation. In the material editor of the VA mesh, “Dynamic Parameter” replaces the “Time” node connected at the start of the “World Position Offset”. Change the name of the first parameter to “Time”. Back in Cascade, refresh the “Dynamic” module and the “Time” parameter is updated in it to reflect our node in the material, as shown in figure 19.

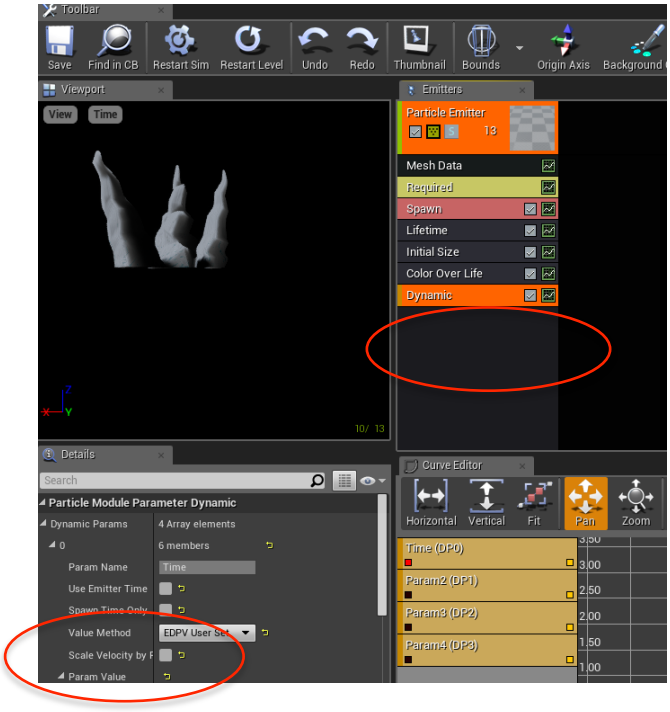


Figure 19: Set Dynamic Material in the Particle Emitter

In the Time parameters, change the distribution to a “constant curve” and add two “point” elements to this. This curve can be viewed using the curve editor of this module. The parameters for the effect will vary from animation to animation. For our ice attack the first point is set to start from emitter time 0 and animation time 0 and the second point to emitter time 0.5 and animation time 3. The total emission life is 2 seconds. The time can be set according to how fast the effect needs to be and making sure it doesn’t repeat.

Now that the animation is under control, in the sword animation sequence window, a custom “notify” is created at the time the effect has to spawn. In the animation BP of the sword, add a script to spawn the emitter at the location of a socket, which in our case was the sword tip. This socket is required since spawning the emitter needs a socket location and rotation. A

short delay has been added to deactivate the emitter. The script is shown in figure 20.

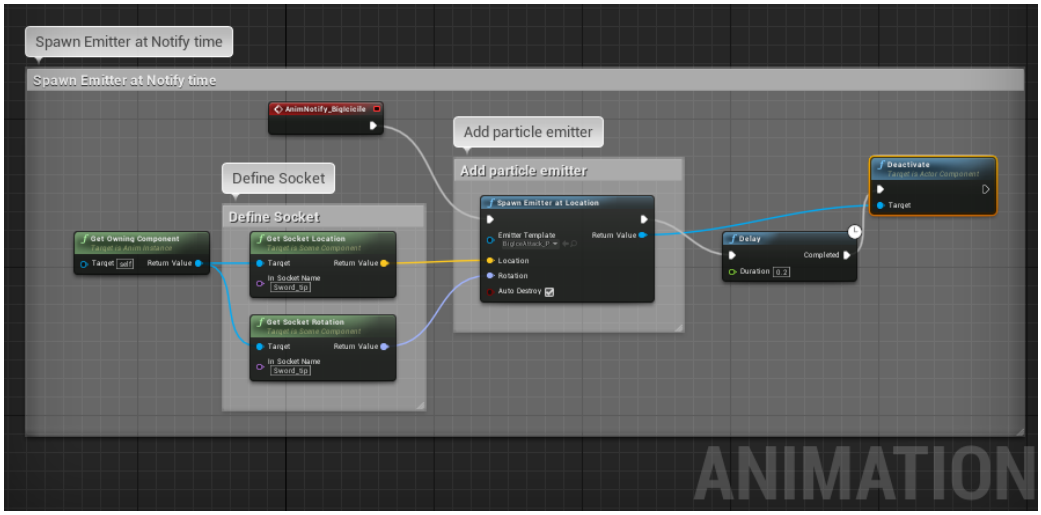


Figure 20: Spawn VA emitter at Notify in Sword Anim BP

In the material, an offset to the actor can be added and in the instance material the offset can be set so that position matches to where it needs to be when spawned. Figure 21 shows the changes in the material network and the final result.

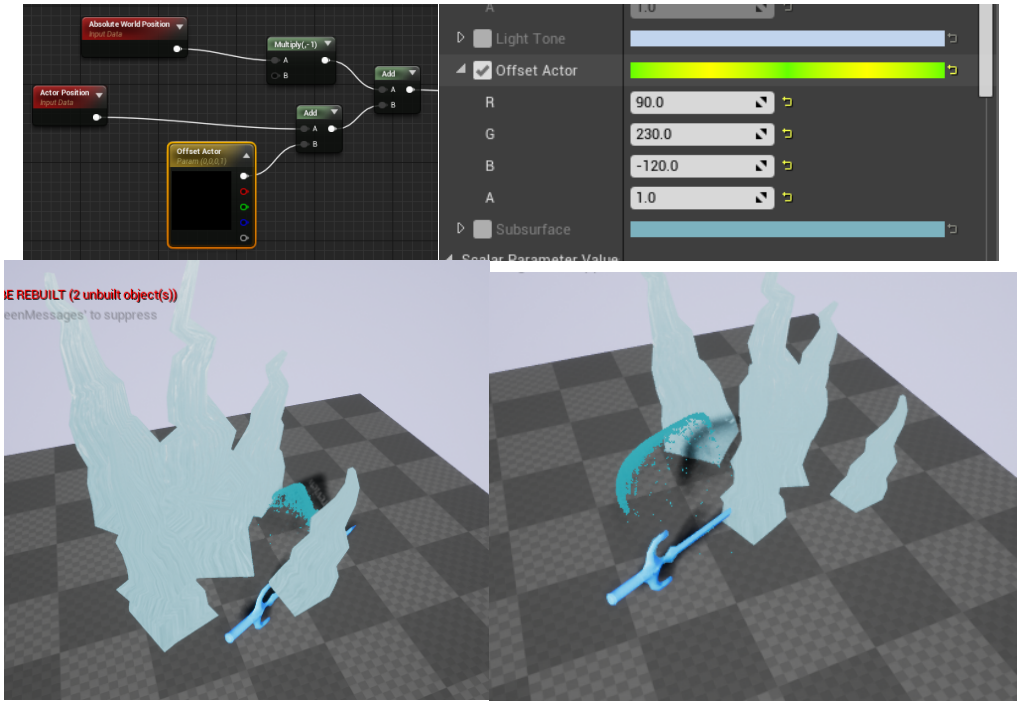


Figure 21: Add Offset in Material.

Top Left - *Modifications in VA Material*. **Top Right** - *Offset Setting in Material Instance.*
Bottom Left - *Before Offset*. **Bottom right** - *After Offset*

4.2.5 Analysis

For the looping animation, one improvement that can be made is that for now the socket is updating constantly. The other way to do that would be to attach the VA animation to the sword and its socket, and only update the position when it is animating. This is because the animation does not follow the socket position only during animation. This way it won't be constantly updating during gameplay, optimizing it further.

Using a vertex animation from Houdini in UE4 for a timed effect gives the flexibility to attach any complex animation and spawn it using notifies. This method provides ease in adding one-time event during any animation which otherwise would be complicated to do in UE4. The reason we are not using the "play particle effect" or "timed particle effect" notify in persona is because we do not want it attached to socket, which is mandatory in "timed particle effect", and we also want it to be deactivated after the animation has played once. Since no current notify gives that option we used a BP script.

One issue with the spawn count is that it only works when set to 5 or above for the ice attack. If the icicle attack is used using the timed particle effect, offset size can vary from the view in persona to the view in the editor since it takes the size of the socket. So spawn in game to test location. Another limitation with this method is that the location that is set for the animation (relative to the sword) will stay consistent. With the current script the transformations cannot be animated.

4.3 Using Houdini Vector Field (VF)

4.3.1 Creating and Exporting the VF from Houdini into UE4

Creating and exporting vector fields from Houdini is very simple. For our example we will create some sparks for the red trail that will be affected by the VF. A tutorial by Mike Lyndon (SideFX, 2017) explains how to create vector fields in Houdini. To create the VF we have used a grid and connected it to a flow map. This node sets the initial direction of the force and the visualizer can be turned on to see that direction. Create some curves that define the flow of the field. Connect these curves and the gird to a "flowmap guide" node, which is part of the game dev toolset. Set the parameters in this node to define the strength, width, and falloff of the effect the curves will have on the flow of the grid. Copy this a few times to create a box of the VF. Next connect the "ROP Vector Field" node. Change

the input type to “Particles” (since we are not using a dynamic from a volume created e.g. smoke) and set the sampling size. For our example a sample size of 10 x 10 x 10 works. Set the directory to save in and render. Our vector field has been created. Import this VF into UE4 to be used. Figure 22 shows our VF network and the flow direction.

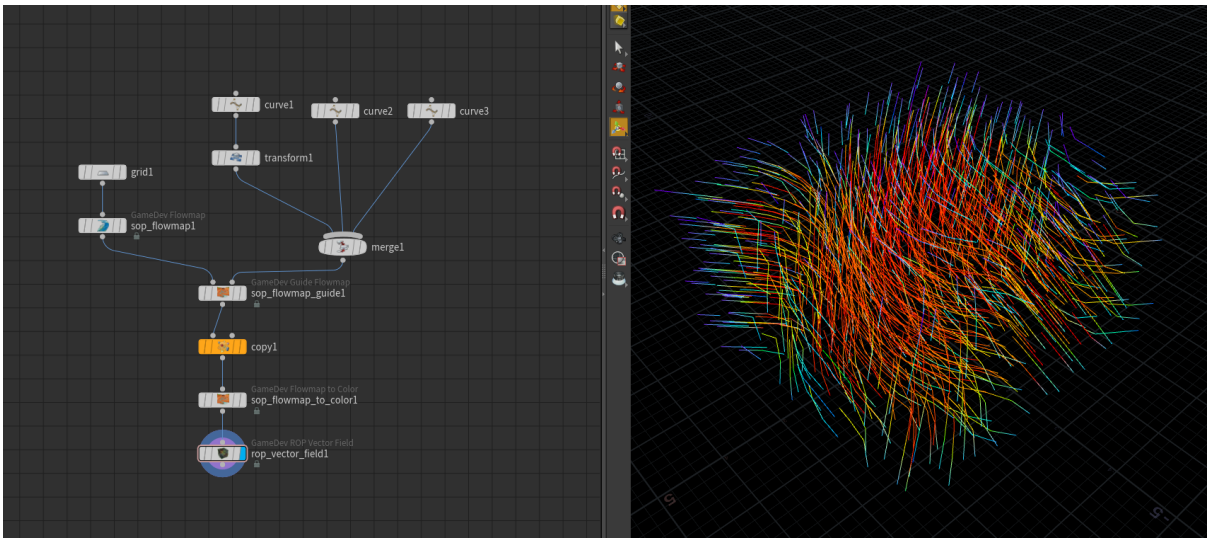


Figure 22: Vector Field Network in Houdini

4.3.2 Setting up the VF using Cascade in UE4

Using vector fields with UE4’s particle emitter allows us more control over how the particles emitted will flow. Change the particle emitter type to a GPU sprite emitter. Vector fields in UE4 can only be used with GPU sprites. As the name suggests GPU sprites are handled by the GPU rather than the CPU, which allows us to create a lot more particles at a given time. Add a “Local Vector Field” module to the emitter and set the vector field created. Add a material to the emitter to define the look if the particle, which in our case we have added the “Sparks” material explained in section 4.4.

Set the intensity and tightness to see any result. The vector fields can be moved around in the cascade viewport to place it where needed. Once the particle effect looks like how it is intended, notifies have to be set using the animation sequence window for the intended animation. Create custom notifies and place them on the timeline when the emitter needs to activate. In our case of the red trails, we add notifies named “Burst” at the mid of every slash on the timeline. In the animation BP of the weapon or character add the script to spawn the emitter at the socket location as mentioned before in section 4.1.3. Figure 23 show the VF sparks result.

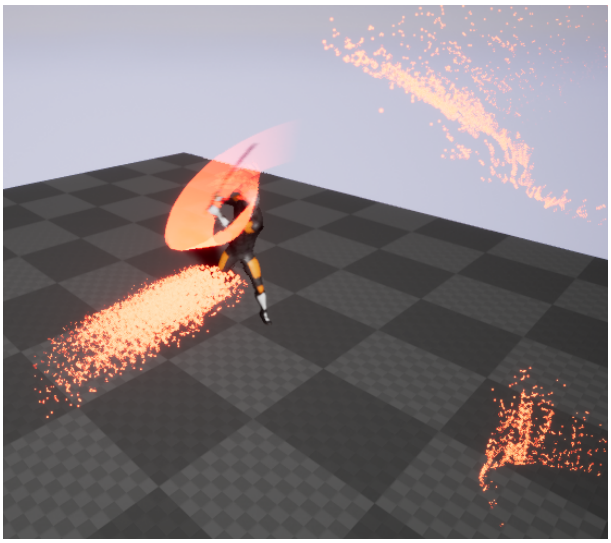
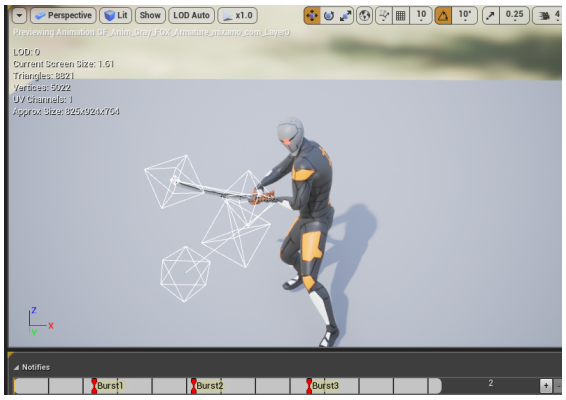


Figure 23: Snapshot of Vector Field Sparks Effect

4.3.3 Analysis

Vector fields are powerful when it comes to animating particles in a certain direction. The ease at which the vector fields can be created from Houdini cannot be replicated by just cascade. Using just the functionality in cascade allows us to control the flow of the particles is some manner but does not give any way of calculating complex movements. UE4 does not provide any preset vector fields either, and that is where Houdini comes into play.

For our sparks animation, spawning the emitter with the "Burst" at the mid of each slash did not give us the direction exactly as we wanted it at first. This was due to the animation itself where the bones do not rotate in the direction of the slash. To tackle this, 3 socket were created on the character sword and positioned at the correct place, with the correct rotation. The scale does not matter. Three different burst notify were created and set to the respective bone in the script as show in figure 24.



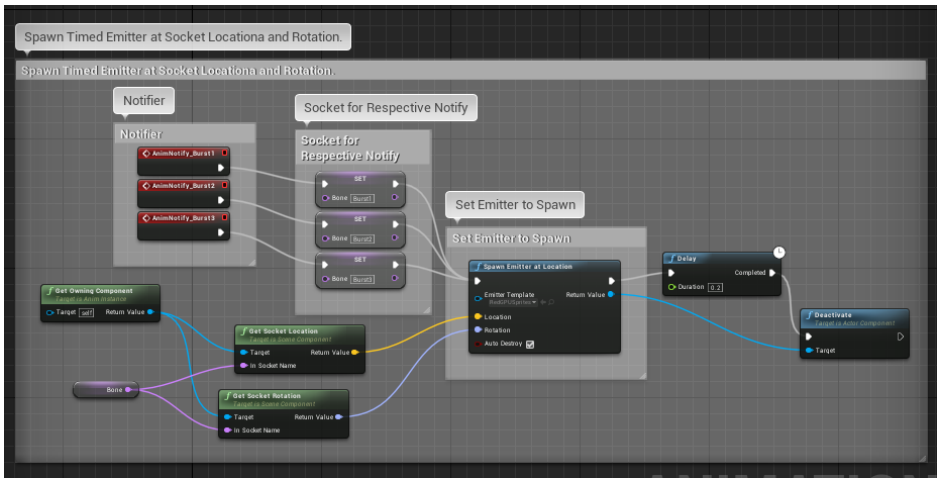


Figure 24: Modifications for VF Sprite Effect Using Multiple Sockets.

Top - *Creating Multiple Sockets for Emitter Transforms.* **Bottom** - *Modified Script For Spawning Emiiter for Multiple Sockets*

4.4 Explaining Material Networks in UE4

The following materials show an overview of the network created for each shader for our example of the effects created.

Red/Purple Trail Material

The red and purple trail materials are the same material with a different colour parameter. This was created with the help of the tutorial by Andreas Glad (SideFX Houdini, 2017). The texture used in our version is different. (Red/Purple Trail Texture, 2018)

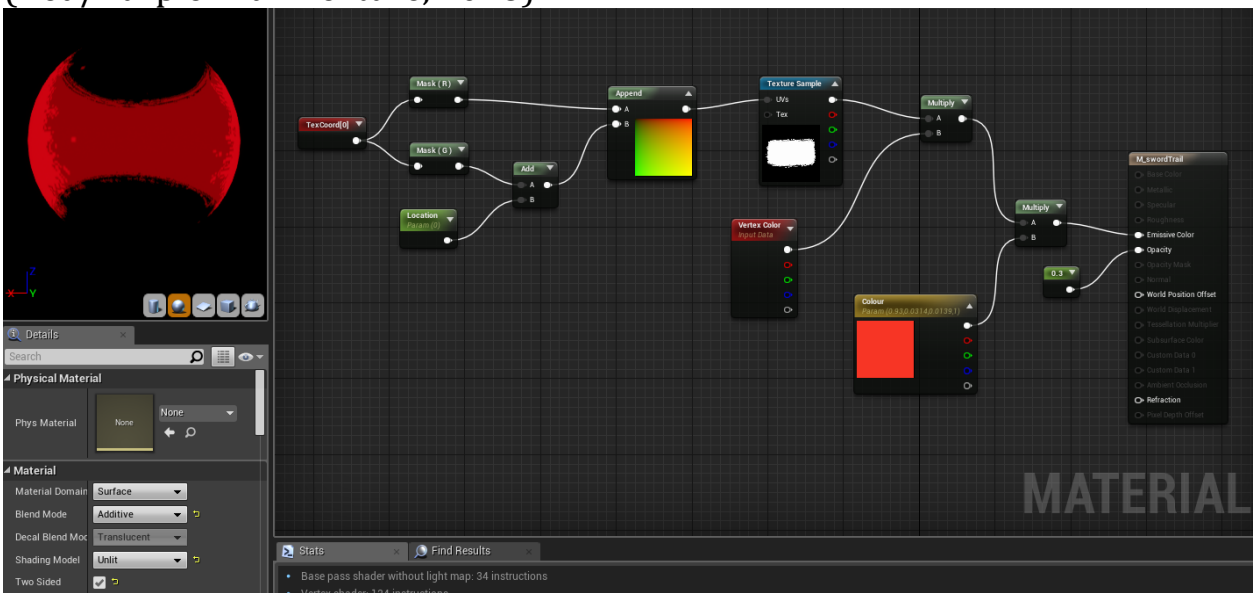


Figure 25: Red/Purple Trail Material

GPU Spark Material

The GPU Spark material is used in the GPU spirits for the red trail.

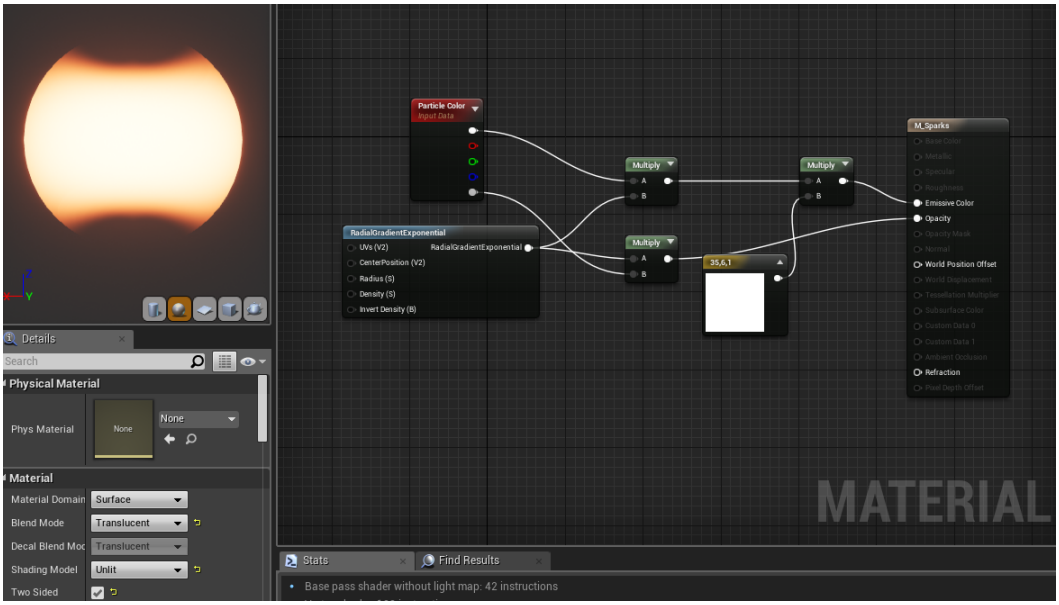


Figure 26: GPU Spark Material

Ice Trail Material

The ice trail material creates an icy shard displacement in the upward direction. The material was create with the help of this tutorial by Dean Ashford (Ashford, 2017) and was slightly modified to suit our effect.

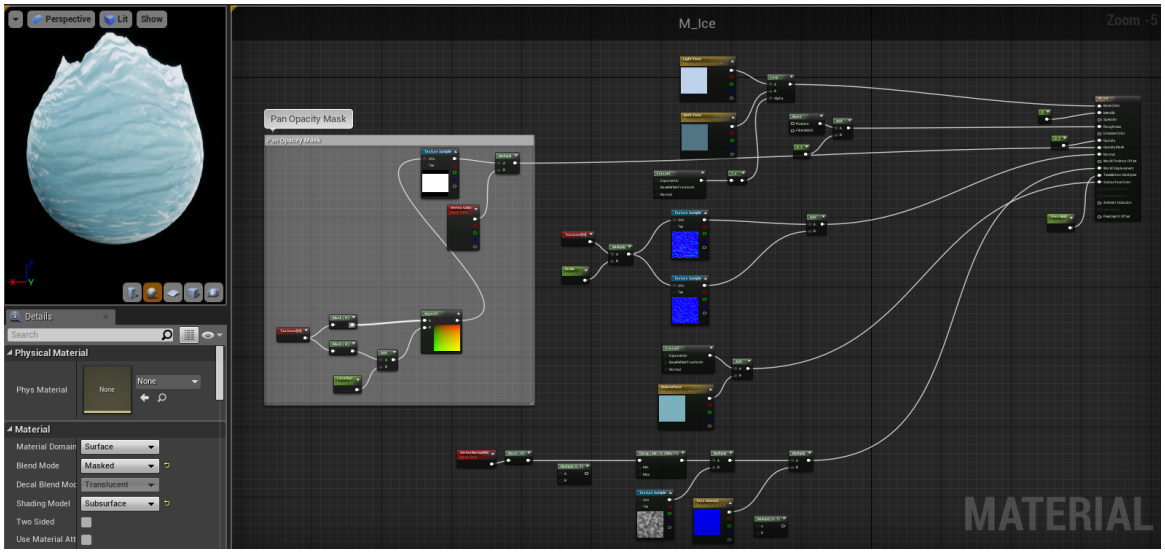


Figure 27: Ice Trail Material

Sword Material to Add Bloom Effect

To create an animated bloom effect on an asset, the material must exist on a mesh or asset that is animated. In the case of our Sword, the material shown in figure 28 is applied. Notice that the material has a “Bloom” parameter multiplying with a colour and is connected to the “Emissive Color” node.

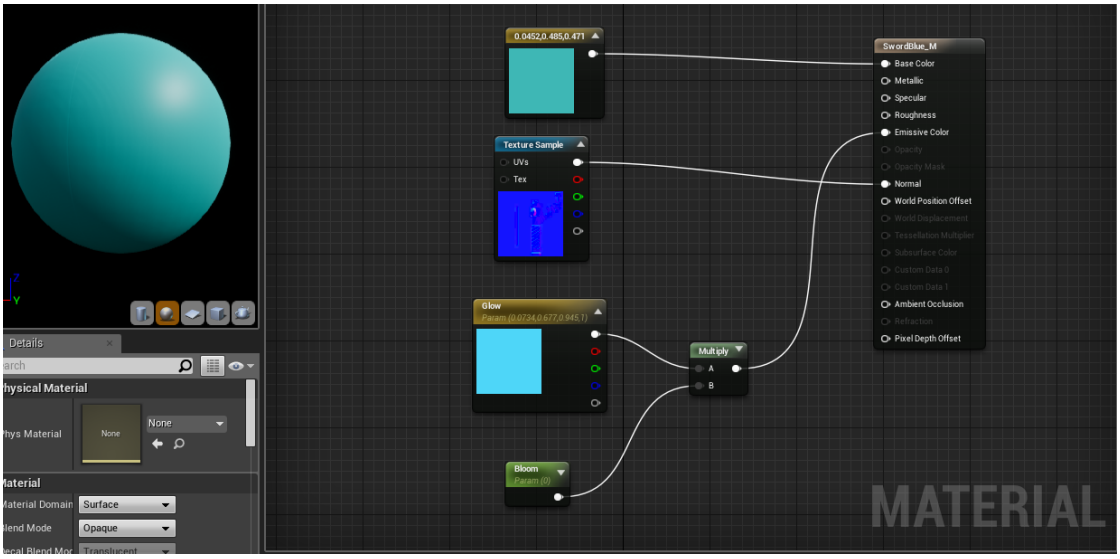


Figure 28: Sword Material for Bloom Effect

In the animation sequence window add a new curve and give it the same name as the parameter. For this curve to be recognized as a material, “Anim Curves” need to be turned on as shown in figure 29.

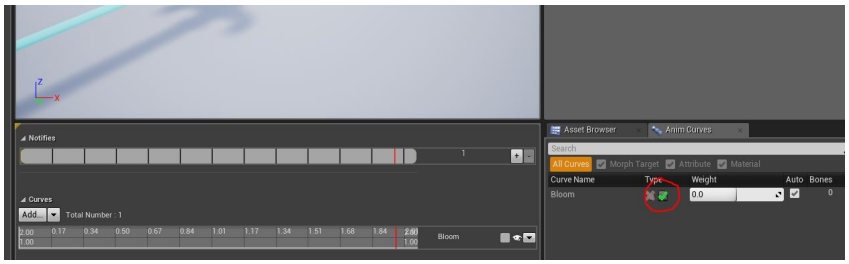


Figure 29: Creating Animation Curves for Bloom Effect

Create the curve along the timeline and the changes can be seen accordingly.

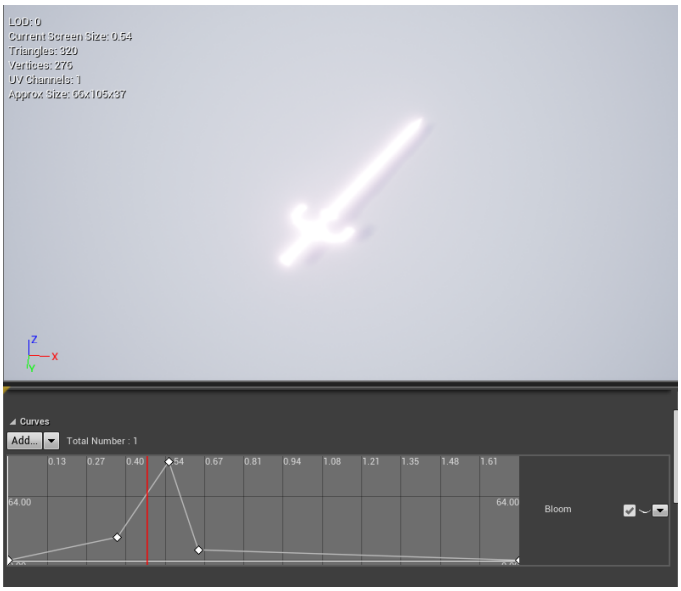


Figure 30: Final Bloom Effect

Ice VA / Electricity VA / Swirl VA Material

The materials created by the vertex animations can be modified as required. In our examples the colour, emission, and some normal maps have been modified or added to the material created by Houdini. The offset to the actor has also been added as mentioned previously.

5 Conclusion

The aim of this project was to utilize the capabilities of Houdini and use them in Unreal Engine 4 to create effects for sword weapons. The focus was to find effective ways to integrate the assets created in Houdini to work with UE4 in a reusable way, which is what the project has managed to show. The methods proposed can be used for creating various effects that can be seen in the industry. These methods can be modified with ease to

quickly see different effects just by changing the shader or the HDA parameters for a different shape.

Though the methods have a workable final result there are limitations with each method which have been discussed above after each method. These include freedom to visualize some results, tedious job of copying over curve data, or some transformation issues from the side of the engine.

In the two methods that have been discussed for creating the trail, though the first method is tedious to create due to copying sequencer keys, it is a more stable way of executing and animating the effect. This is due to the fact that multiple key presses do not affect the execution of the effect whereas the second method does, which is not ideal. The UE4 API does not allow a way in the blueprints to refuse inputs while the sequencer is playing. This however can be tackled in the future by adding in a delay for a key press while the effect is playing. The sequencer in UE4 was not meant for creating effects for game but is more for creating cinematic or cut scenes, and therefore is not tailored for handling in-game effects. However that said, the sequencer is the best way to visualize these effect animations, especially when added on later to the character animation.

When it comes to performance, each is method is performance efficient. The vertex animation is a powerful tool where simulations with very heavy computations can be converted in to VA and animated in game. There's much that can be explored with vertex animation for weapon effects and also the capabilities of Houdini Engine for UE4 itself. Using the HDA for a trail mesh is just one example. A lot more can be done for other weapons, such as arrows, guns, and magic etc.

References

3ds Max. (2018). Autodesk, Inc.

Ark: Survival Evolved. (2017). Studio Wildcard.

Ashford, D. (2017). *UE4 - Tutorial - Dynamic Materials in BluePrints! (Request!).* [video] Available at: https://www.youtube.com/watch?v=6OTaEHfRyH8 [Accessed 9 Aug. 2018].

Ashford, D. (2017). *UE4 - Tutorial - Ice! (Ice baby!)*. [video] Available at: https://www.youtube.com/watch?v=sE64iTjnoUM [Accessed 10 Aug. 2018].

Azure Sword. (2017). [3D Asset] Available at: https://sketchfab.com/models/f49c38d21e0449c387c7d6174e16f973 [Accessed 9 Aug. 2018].

Bannink, P. (2009). Houdini in a games pipeline. In: *SIGGRAPH '09 SIGGRAPH 2009: Talks.* [online] ACM, p.Article No. 61. Available at: http://10.1145/1597990.1598051 [Accessed 14 Aug. 2018].

Blender. (2018). Blender Foundation.

CGSociety. (2018). *Creating VFX for Games, with Houdini Artist Andreas Glad*. [online] Available at: http://www.cgsociety.org/news/article/3238/creating-vfx-for-games-with-houdini-artist-andreas-glad- [Accessed 13 Aug. 2018].

DV7 Pavilion (2016). *UE4 Fire Sword Trail.* [video] Available at: https://www.youtube.com/watch?v=JCfbBA42_xE [Accessed 14 Aug. 2018].

FX HIVE EVOLUTION (2016). *3D Animation Houdini Tutorial - Ice Chrystals Designs*. [video] Available at: https://www.youtube.com/watch?v=5ifMIXwBzh8 [Accessed 10 Aug. 2018].

Gabriel Aguiar Prod. (2017). *Unity 5 - Game Effects VFX - Ice Attack*. [video] Available at: https://www.youtube.com/watch?v=XqWZZejtjIk&t=1s [Accessed 14 Aug. 2018].

Gray Fox. (2017). [3D Asset] Available at: https://sketchfab.com/models/7313f22ffe3945cf83445d1bc3860d73 [Accessed 9 Aug. 2018].

Houdini. (2018). Side Effects Software Inc.

imn nam (2016). *Game effect tutorial - Sword Slash.* [video] Available at: https://www.youtube.com/watch?v=wFgS5pzG1Qs [Accessed 14 Aug. 2018].

Hellblade: Senua's Sacrifice. (2017). Ninja Theory.

80 Level. (2017). *Horizon Zero Dawn: Procedural Rivers & Wires.* [online] Available at: https://80.lv/articles/horizon-zero-dawn-procedural-rivers-wires/ [Accessed 14 Aug. 2018].

Mirza (2017). Unity VFX - *Weapon Effect: Electric-Spark Blood Sword (Particle System Tutorial)*. [video] Available at: https://www.youtube.com/watch?v=PswVoS5dxtA [Accessed 14 Aug. 2018].

Maya. (2018). Autodesk, Inc.

MIX Training (2017). *VMT 012 - HOUDINI - Particles Follow a Curve*. [video] Available at: https://www.youtube.com/watch?v=pBZM387_SYg&t=209s [Accessed 10 Aug. 2018].

80 Level. (2017). *Procedural Technology in Ghost Recon: Wildlands*. [online] Available at: https://80.lv/articles/procedural-technology-in-ghost-recon-wildlands/ [Accessed 14 Aug. 2018].

Red/Purple Trail Texture. (2018). [image] Available at: http://jeffreycollins.us/wp-content/uploads/2018/02/paint-stroke-lovely-white-paint-stroke-08-of-paint-stroke.png [Accessed 10 Aug. 2018].

Sea of Thieves. (2018). Microsoft Studios.

SideFX (2017). *GAME TOOLS | VECTOR FIELDS TO UE4.* [video] Available at: https://www.sidefx.com/tutorials/exporting-vector-fields-to-ue4/ [Accessed 10 Aug. 2018].

SideFX Houdini (2017). *CurveSweeper - Intro to Houdini Engine // Houdini for Games.* [video] Available at: https://vimeo.com/223354261 [Accessed 9 Aug. 2018].

Smyke (2016). *UE4 - VFX weapon effect*. [video] Available at: https://www.youtube.com/watch?v=56PctWEYSKY [Accessed 14 Aug. 2018].

TorQueMoD (2015*). UE4 Quick Tutorial - Spawn Actor at Player Location with Blueprint*. [video] Available at: https://www.youtube.com/watch?v=jMBlz0o5wNk [Accessed 9 Aug. 2018].

Unity. (2005). Unity Technologies.

Unreal Engine 4. (1998). Epic Games.

Bibliography

Kruel, L. (2017). *Vertex Animation GDC Project Overview*. [video] Available at: https://vimeo.com/207832662 [Accessed 14 Aug. 2018].

mamoniem (2017). *Unreal Engine, Houdini Vertex Animation to UE4 - UE4U.XYZ*. [video] Available at: https://www.youtube.com/watch?v=zGmElCpxZnk [Accessed 14 Aug. 2018].

Mckinnon, L. (2015). *Unreal Engine 4 Weapon Animation Trails Tutorial*. [video] Available at: https://www.youtube.com/watch?v=an1ndqt-iO0 [Accessed 14 Aug. 2018].

SideFX Houdini (2017). *Flowmaps! // Houdini for Games*. [video] Available at: https://vimeo.com/243733565 [Accessed 14 Aug. 2018].

Unreal Engine (2015). *Skeleton Assets: Using Anim Notifies, Curves & Slots | 05 | v4.8 Tutorial Series | Unreal Engine*. [video] Available at: https://www.youtube.com/watch?v=per6KmuvRlQ [Accessed 14 Aug. 2018].

Unreal Engine 4 Documentation. (n.d.). *Animation Curves*. [online] Available at: https://docs.unrealengine.com/en-us/Engine/Animation/Sequences/Curves [Accessed 14 Aug. 2018].

Villani, R. (2017). *UE4 Tutorial 101 — Control Materials in Blueprints*. [video] Available at: https://www.youtube.com/watch?v=I8s-Bt-YOUg [Accessed 14 Aug. 2018].

Abbreviations

Anim	Animation
BP	Blueprint
HDA	Houdini Digital Asset
UE4	Unreal Engine 4
VA	Vertex animation
VF	Vector Field