Tero Vääräniemi

Example Solutions of Visual Effects in Fantasy Games

Bachelor of Business Administration

Spring 2016

KAJAANIN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

TIIVISTELMÄ

Tekijä: Tero Vääräniemi

Työn nimi: Esimerkkitoteutuksia visuaalisista efekteistä fantasiapeleissä

Tutkintonimike: Tradenomi (AMK), Luonnontieteiden ala

Asiasanat: visuaalinen efekti, peligrafiikka, partikkeliefekti, vfx, fx, fantasiapelit

Tämän opinnäytetyön tavoitteena on antaa lukijalle käsitys miten visuaalisia efektejä on mahdollista luoda peleihin, jotka sijouttuvat fantasiamaailmaan. Tämä toteutetaan käymällä läpi asioita joita on hyvä noudattaa visuaalisia efektejä tehdessä sekä analysoimalla kahdeksaa eri visuaalista efektiä, jotka luotiin tätä opinnäytetyötä varten. Efektit on tehty käyttäen Unreal Engine 4 –pelimoottoria, jonka eri työkaluja käydään myös läpi visuaalisten efektien kannalta.

Tekijän toiveena on että tästä opinnäytetyöstä voisi olla hyötyä pelialaa opiskeleville tai henkilöille, joiden yleisen kiinnostuksen kohteena ovat visuaaliset efektit tai niiden toteutus videopeleissä. Visuaalisten efektien luonti on kuitenkin taidetta, joten tämä opinnäytetyö kuvastaa vain tekijän henkilökohtaisia töitä, työtapoja ja kokemuksia.

ABSTRACT

Author: Tero Vääräniemi

Title of the Publication: Example solutions of visual effects in fantasy games

Degree Title: Bachelor of Business Administration (UAS), Business Information Technology

Keywords: game graphics, visual effects, particle effects, game vfx, game fx, fantasy games

The goal of this thesis is to give the reader an insight into how visual effects can be implemented in a fantasy style game. This will be mainly done through observing important factors to remember whilst creating visual effects and by breaking down eight different visual effects created specifically for this thesis. The effects are created using Unreal Engine 4 game engine, its tools will also be explored and explained from the viewpoint of visual effects.

The author wishes that this thesis may be of some use to people studying game development or whom are simply interested in visual effects and their implementation in video games. However, creation of visual effects can be classed as art, thus this thesis merely depicts the authors personal work, workflow and experience.

FOREWORD

In mid-December 2015, I was pretty bummed out that I did not get the internship of my dreams that seemed so possible and real just a few weeks prior. In hindsight it seems to me that it was probably for the best that my internship application was declined, as it gave me this chance of really putting all the time I wanted into learning visual effects, time which I had not really had until this point.

It has been a great learning experience and a great opportunity for me to strengthen my core skills before trying to break out into the industry. I want to give special thanks to my fellow students and team members on the Ancestory project, who first gave me the chance to create visual effects for the game and a great chance to learn. Without them or the project we all worked so hard for, I would not be making this thesis.

Contents

LIST OF SYMBOLS

Bloom = an effect to make an object appear bright by extending the light from the borders of bright areas

Depth of field = an effect mimicking the focus range of human eyes and cameras

Fresnel = an effect where light reflects at different intensities according to the viewing angle, commonly used in video games to highlight silhouettes of objects

HLSL = High Level Shading Language, shading programming language

Juiciness = a loose term used in game development to describe how good the gameplay feels when a game is played

Motion blur = apparent streaking of rapidly moving objects

Particle = most often a small image or a series of imaged projected on a plane, though there are many types of different particles, such as mesh particles which use 3d meshes

Particle emitter = object that emits particles

Skeletal mesh = animated mesh using hierarchical skeletal structures for animation

Static mesh = a 3D model within Unreal Engine 4 which is not rigged

Subsurface = mimics the scattering of light inside materials such as wax, which seem opaque but due to the light scattering inside some of the light from the opposite side of the surface shows through

Topology = the layout of a 3d model, how vertices and edges are placed to create the surface of a mesh

UV = projection of 2d images onto 3d surfaces

Vertex animation = animating a 3D model by storing a series of vertex positions

1 INTRODUCTION

Visual effects and artists that create them are in a quite unique position in game development. They are in the middle ground between art and programming, between creativity and technicality. Creating and implementing visual effects requires skills, techniques and understanding from both of these areas. VFX artists tend to gravitate toward being jack of all trades, as creating and implementing visual effects requires skills from a wide base from 3D modelling to scripting or programming. It may be daunting and challenging at times, but also very rewarding. Effects artists get to work closely with other artists and programmers, which stresses the importance of social aspect of game development and teamwork.

1.1 What are visual effects and why are they needed?

Visual effects are often defined through the viewpoint of the film industry, as there is very little literature dedicated wholly to video game visual effects. Which is shame, hopefully as the game development industry matures the literature related to different fields within the industry will mature as well.

From a game development perspective, a visual effect is a special effect, which is used to give visual feedback to the player. This can range to great deal of different things and purposes. Visual effects can be simply used to add to a scene or something that the player expects to see when he or she does something. Environmental effects such as wind blowing up dust on a desert or a muzzle flash of a gun firing are good examples of these two different purposes. However, visual effects can also be used to communicate gameplay and game mechanics. Say a character swings a sword with nothing added to the actual swing of the sword. Compared to that, a sword with a swing that has particle effect trailing the arc of the swing (See figure 1) has a far greater sense of power to it. The effect can also

improve the overall graphical outlook of the sword swinging animation, as it gives something clear for the player's eyes to get attracted to. The effect improves the "juiciness" of the animation and in general, visual effects tend to enhance the overall responsiveness of games.

Figure 1. Sword swinging animation with trailing particle effect in World of Warcraft: Legion. [1]

The term "visual effect" can be used quite broadly, as it includes not only things like particle effects and such but also effects that mimic the behavior of a camera or a human eye, such as bloom, depth of field and motion blur. When the term is used within this thesis, the term will refer to visual effects which are used to provide visual feedback in regards to gameplay, unless stated otherwise.

2 KEY THINGS TO CONSIDER IN VIDEO GAME VISUAL EFFECTS

2.1 Shape

Shape, along with timing, are the two most defining factors of a visual effect. Without having the correct or expected shape, a visual effect will simply not be convincing. A lightning bolt without the noisy, sharp turns and thin form simply would not be recognizable as lightning. Fire, especially when attempting to create realistic fire, is not credible without the internal motions of the shapes in fire, which are caused by the complex and turbulent movements of hot and cold air colliding. Symmetry should be something to be avoided especially when dealing with something related to the forces of nature. An explosion does not expand smoothly as a sphere but rather as something that can only be described as chaotic. Getting the shape right is essential when creating visual effects. Studying reference closely is a good starting point. According to Gilland, when creating effects, whether it is for classical hand drawn animation on paper or digital art, it is important to understand natural phenomena and physics at work in order to create something convincing. [2]

2.2 Timing

The importance of good timing in visual effects cannot be stressed enough. Visual effects and animation share some common ground through this, as timing is essential in animation as well. Timing can be used in many different ways to achieve the desired effect, as demonstrated by Gilland, for example a splash of water will seem instantly more massive simply by making the splashes of water travel slower through the air as they are shot up from a body of water. [3]

It might be easy to think that you cannot have particles appearing out of nowhere. Most of the time this might be true, as particles suddenly appearing out without

any transition is not very pleasing to the eye. Though this does not hold true always, as in the case of explosions. In the first few fractions of a second in a typical explosion, the volume of the gasses increase extremely quickly. To recreate this phenomenon as a visual effect some shortcuts must be taken, as in games there are a limited number of frames to work with. The first frame of an explosion can be huge and things can change very quickly during the first few frames as the energy of the explosion pushes outwards. The first frame can also be a flash with the main explosion event following a few frames later, this timing difference can be used in an attempt to communicate the distance between the camera and the explosion to the player.

Having clear “beats” tend to work well in visual effects. It improves the readability of an effect. Imagine a magical visual effect of icicles appearing from the ground. Having the icicles coming in distinct waves opposed to having a steady stream of icicles can be generally more satisfying and more easily understood. From game mechanics perspective, in most cases the wave approach should be preferable as it is easier for the player to understand when the icicles cause their damage or other gameplay effects. Having clear beats also makes it easier for the sound effects to match the visuals, which improves the gameplay feel and responsiveness of the visual effect as everything lines up neatly.

Anticipation is another interesting facet of timing, though in visual effects it can be dictated by the game mechanics related to the effect. Having some sort of anticipation is good as it gives a chance for the player to be prepared for something to occur, though it is not always possible due to the requirements of gameplay. In most cases the visuals would generally benefit from having anticipation but when something has to happen as quickly as possible to retain the responsiveness of the gameplay, the visual effects must cater to the needs of gameplay.

2.3 Performance

Performance is something to always keep in mind when creating visual effects for games. As opposed to films, video games are rendered in real time. This means the available rendering time for visual effects in games are measured in milliseconds, whereas the rendering of visual effects in films can be measured in minutes or even hours. Thus in game development, visual effects artists need to achieve more with less. Keeping the effects visually appealing while maintaining performance can be challenging. Mobile games have traditionally had quite limited visual effects as mobile devices do not have the graphical processing power compared to consoles and PCs. Though things are improving as mobile devices become more and more powerful, which in turn enables game engine developers to improve the graphical fidelity of mobile games by giving game developers access to more powerful rendering tools.

Knowing how to profile the performance of a game is essential when attempting to optimize the performance of visual effects. Most game engines have built in profiling tools tailored to this task, usually these tools provide great analytical methods to inspect the rendering and computing process of a given scene. The most common performance issue with visual effects and especially particle effects is overdraw. This is a situation where many translucent objects are stacked upon another, which is bad for performance. Some overdraw is manageable as long as the area with overdraw is relatively small in the overall screen space. Rendering performance decreases massively when overdraw increases to larger and larger sizes in the overall screen space.

3 MAGIC IN FANTASY GAMES

Magic is a key part of most fantasy worlds, including games. Fantasy as a genre requires aspects that are beyond the real world in order to be considered fantasy. Most often this is the existence of magic and people or beings whom are able to use it [4]. In fantasy games, magic and how it is presented to the player and how it is constructed within the game is an essential part of a fantasy game's attempt to build a credible world where the player can have his or her adventures in.

3.1 Division into elements

Most fantasy games, but not all, divide their magic into different elements. This seems to be inspired by different classical elements as described by different ancient philosophies. These classical elements of fire, earth, air, water and so on, have inspired artists throughout history. Games are not an exception to this. Though the classical elements have been proven to be incorrect explanations of our physical world, they cater to the basic human need of putting things into a context, putting labels onto things and organizing them into neat little boxes.

The elemental approach works well in regards to creating interesting game mechanics. Different elements can be setup to have their strengths and weaknesses, creating engaging gameplay when an element counters another. It's easy to imagine game mechanics with combination of different elements of magic, which would augment the power of the magic. For example, it is easily understood how combining the elements of water and lightning or fire and air would be more powerful.

3.2 The challenges of implementing visual effects in a fantasy setting

The main challenge and also the most interesting part of creating visual effects for a fantasy game is the variety of different elements and effects usually required for a fantasy title. Compared to, for example, a modern military shooter game, which would mainly require different kinds of fire, smoke and explosion effects, fantasy titles usually have fire, lightning, ice and imaginative elements such as arcane or holy. Creating these elements that do not exist in the real world while keeping the effects easily recognizable can be challenging because of the lack of real world references. Though this gives the artist a great creative opportunity and freedom to do things as he or she wishes, due to the lack of constraints or expectations from the real world.

4 TOOLS FOR VISUAL EFFECTS WITHIN UNREAL ENGINE 4

4.1 Cascade

Particles in Unreal Engine 4 are created with Cascade, which is a modular particle effects editor that is fully integrated with the game engine. In the following figure the default view of Cascade can be seen, featuring the preview window, a particle emitter with its modules, curve editor, and selected module parameters (See Figure 2).

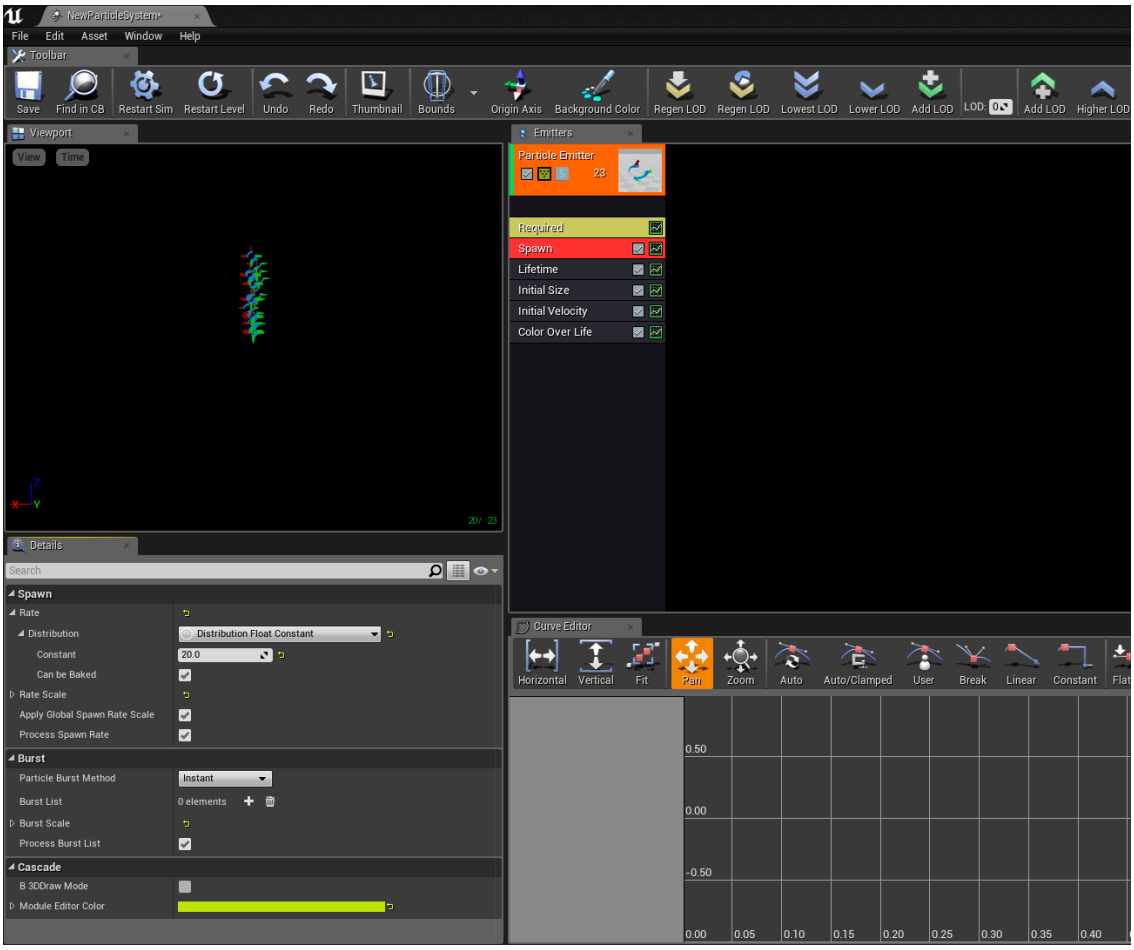

Figure 2. Default view of Cascade

The main function of Cascade is to control the behavior of particle systems, while the appearance of particles is controlled by materials used by the particles. The

material editor and Cascade are setup in a way that the two systems can talk to each other. This is most often used to control different parameters of the materials inside Cascade throughout the lifetime of particles.

The modular approach of Cascade allows the user of the particle editor to create a great amount of different and unique particle behavior and movement. The different modules are sorted into different types, as seen in the figure below (See Figure 3), which affect a specific property of the particles in the emitter. By using different modules and tweaking their parameter values the user of the particle editor can achieve the desired particle behavior.

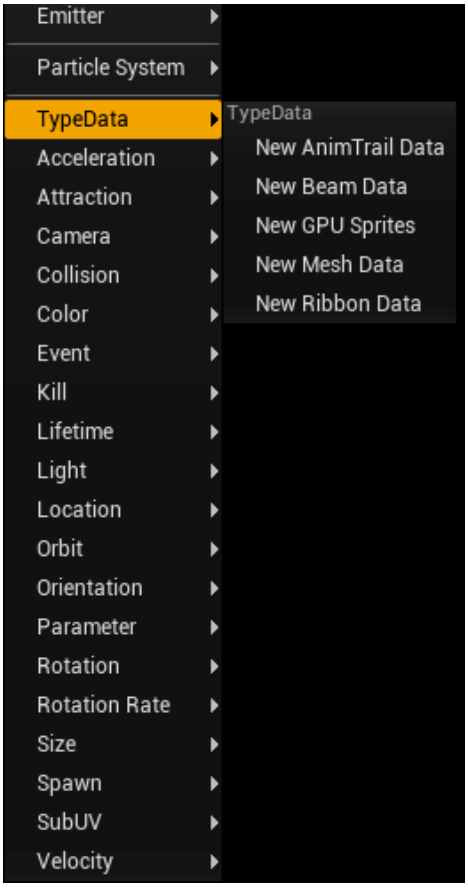

Figure 3. Different types of modules within Cascade

In addition to modules that affect particle behavior, there are five different type data modules that change the emitter type. By changing the emitter type some modules and other features exclusive to that emitter type will be available for use.

4.1.1 CPU particles

By default, particle emitters in Cascade are CPU emitters, meaning they are controlled by the processor. A CPU emitter allows for several thousands of particles in a frame. CPU particles are also quite flexible, as CPU particles have access to various modules that change a given attribute over the lifetime of the particle. One of these modules, which is called Dynamic Parameter, can be very useful when the particle in question requires visual behavior which cannot be with Cascade alone. By using Dynamic Parameter, material properties can be changed as needed to achieve the desired visual appearance.

4.1.2 GPU particles

GPU particles do not have all the features available to the default CPU particles, but they are far more efficient and also possess a few unique features. A GPU emitter is able to simulate and render hundreds of thousands of particles efficiently [5]. The most interesting feature of GPU particles, alongside their efficiency, is that they can use Vector Field modules. Vector fields allow particles within the emitter to be influenced by a uniform grid of vectors. These vector fields can also be rotated and scaled. Very interesting particle behavior and motion can be achieved by using them [6]. In the following figure and related video footage, one million GPU particles are being influenced by a turbulent vector field, which is rotating around its Z-axis (See figure 4).

Figure 4. One million GPU particles influenced by a rotating vector field. See video here.

Vector fields can be generated through various means, the vector field used in the particles above (See figure 4) was created in Autodesk's Maya. Epic Games, the creator of Unreal Engine, provides a Maya script which can be run to take a snapshot of a single frame of Maya fluid simulation to generate a vector field that can be utilized within Cascade [6]. Additionally, it is worth mentioning that there is a script for Autodesk's 3ds Max by Ruben Henares, which allows a user to generate vector fields from spline shapes inside 3ds Max. In general, vector fields created from splines are helpful when trying to achieve specific motion or shape with the particles, whereas vector fields generated by Maya fluid simulations are more suited in cases where complex or chaotic shapes are required.

4.1.3 Mesh particles

Mesh particle emitters are similar to CPU particles in terms of the different modules that are available to them. As the name suggests, instead of emitting billboards a mesh emitter emits instances of a static mesh, as seen in Figure 5.

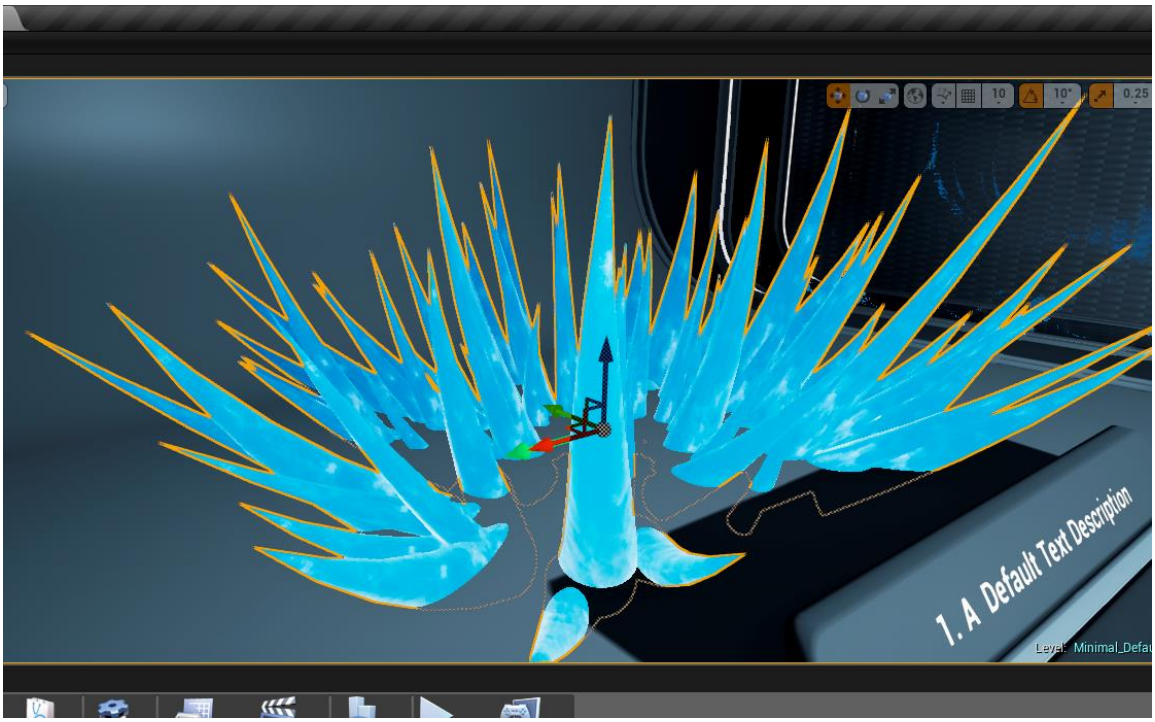

Figure 5. Emitting mesh particles

Within Unreal Engine, one of the more interesting applications of mesh particles is using a mesh that is animated through vertex animation. This greatly expands the variety of effects that can be done by visual effects artists, as it allows having animated meshes in particle systems. However, the vertex animation tool provided by Epic Games does have some limitations. As stated by Unreal Engine's documentation, the tool works by storing the animation data into a 2D texture according to the following formula:

Final texture resolution: X = number of vertex in the mesh, Y = Number of frames captured.

As DirectX 11 maximum texture size is 8192 pixels, this allows for a maximum of 8192 vertices in a mesh to be animated. This vertex limit is not a great issue for

visual effects work, as the meshes used in visual effects in games do not usually possess that many vertices. [7]

4.1.4 Ribbon particles

Ribbon emitters are used to generate trails of particles by connecting particles together to form ribbons. Another particle emitter within the particle system can be selected as a source for the ribbon particles, meaning that the ribbon particles are generated as the source particle moves. Ribbons can be applied in many ways and in different kinds of effects. For example, they can add trails to shrapnel of an explosion, or in more magical type of effects. In the following figure, a magical effect is utilizing ribbon particles to form trails that surround a character. The following effect was made in Ancestory for a spell that purifies a friendly target of any negative status effects.

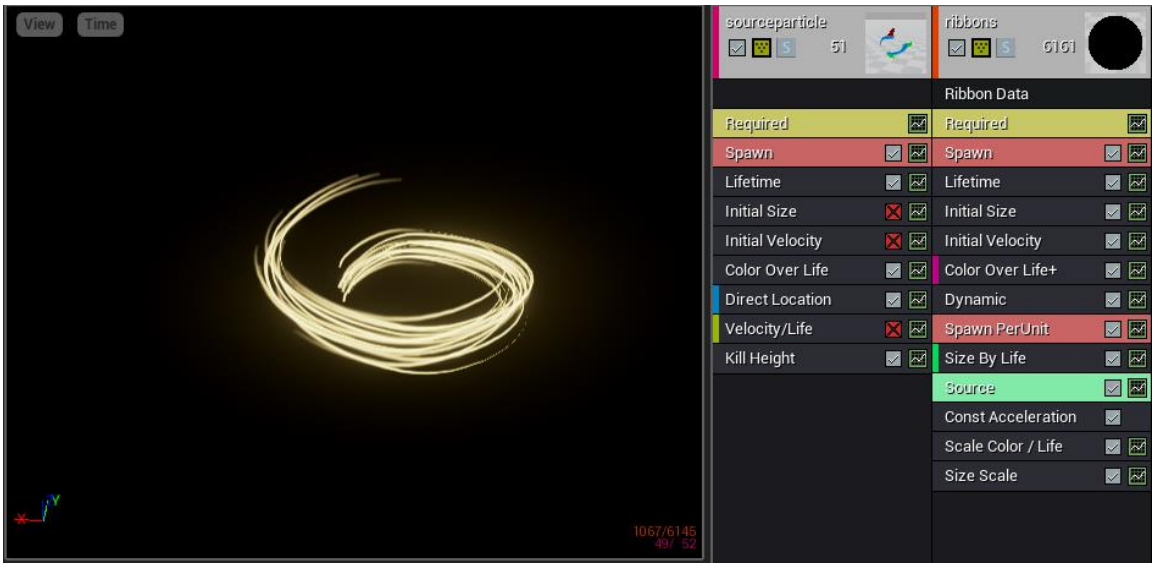

Figure 6. Ribbon particles in a magical effect

4.1.5 Beam particles

Beam emitter creates beams of particles by connecting a source point and a target point with a stream of particles. Source and target points can be defined in different ways: they can be fixed locations defined by 3-dimensional coordinates or they can be parameters that can be changed in runtime. Beam emitters have access to a noise module, which is especially useful when trying to simulate lightning or electricity. The following image is from a particle video guide by Epic Games, the creator of Unreal Engine. The beam in the image (See figure 7) is set between two fixed points, the noise in the beam and the arc is achieved through the Noise module in Cascade. [8]

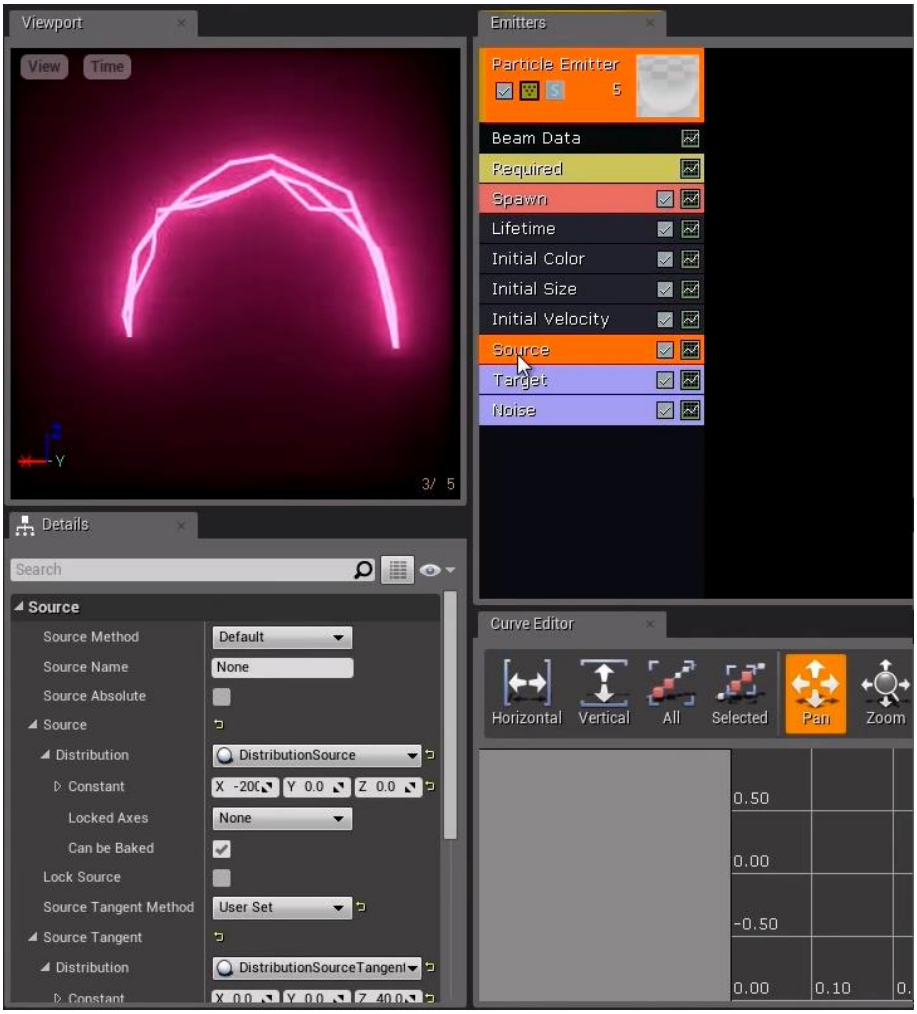

Figure 7. Beam emitter in Cascade with Noise module [8]

The arc is achieved by giving the noise a tangent both at the source and target locations, as shown below by Figure 8.

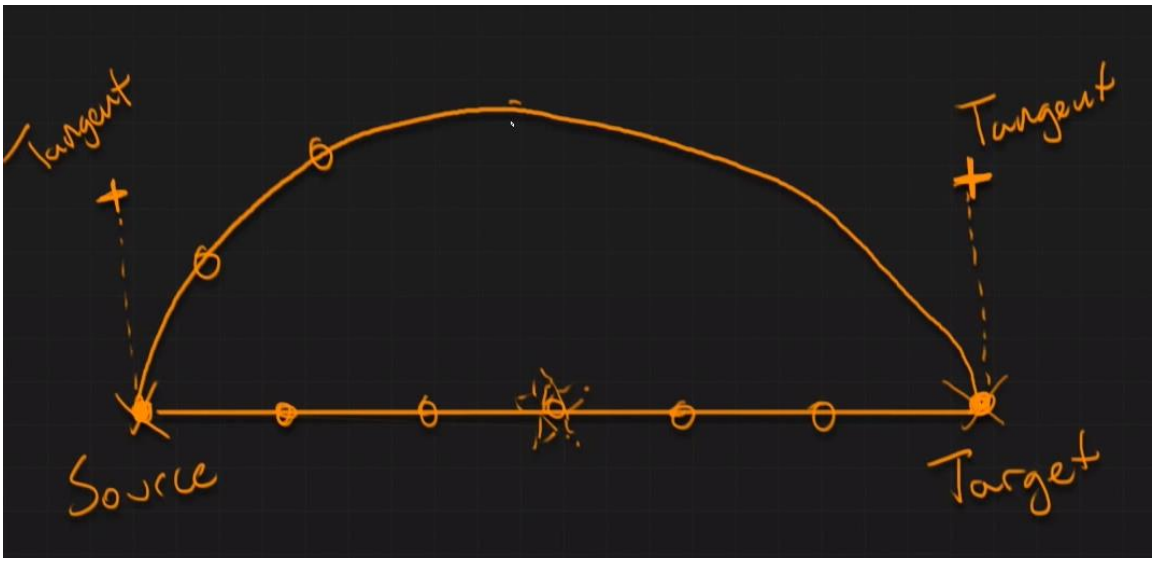

Figure 8. Illustration of creating an arc in a beam emitter through the use of noise tangents. [8]

4.1.6 Animtrail particles

Animtrail particles are polygon sheets, which are emitted from sockets, also known as bones, of a skeletal mesh. These particles fill a very specific need of creating trailing particles according to the movement of an animated mesh. One of the more common uses for these particles is to create trailing particles when a character swings a weapon, as seen in the following image from Ancestory (See figure 9).

Figure 9. Animtrail particles being created as a character swings his weapon in Ancestory. See video here. [9]

The trail is generated between two defined sockets during a specified time in the animation of the character. The sockets do not have to be created as the model is being animated, as sockets can be created on imported skeletal meshes within Unreal Engine after the animation work is done. In the following figure the setup of the example above can be seen in the game engine itself (See figure 10).

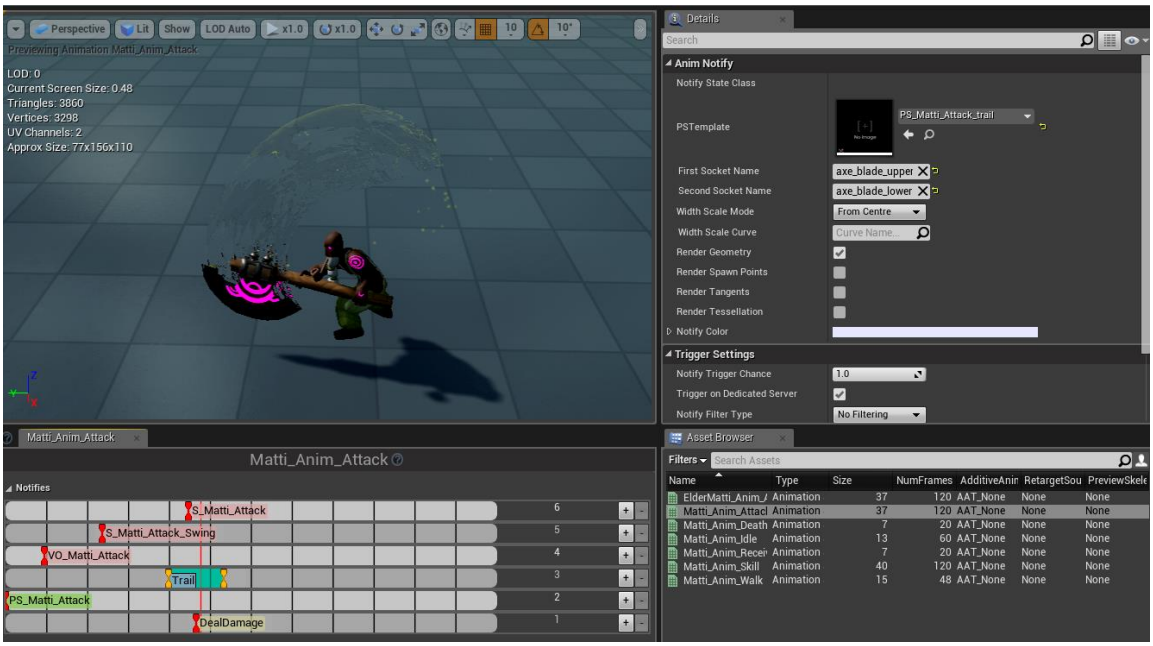

Figure 10. Setting up an animtrail inside Unreal Engine 4.

4.2 Materials and material editor

Materials control the visual appearance of graphical assets in Unreal Engine 4. According the game engine documentation, materials can be compared to paint which is spread upon a surface, while also defining the surface properties. A material defines the color, shininess, translucency, and other properties of the object that the material is applied to. From a more technical perspective, materials are used to calculate how light behaves when it interacts with a surface. The calculations are done using image data (textures) as input and mathematical expressions, while also being affected by various settings of the material [10]. Materials are created with a network of visual scripting nodes within the Material Editor. Each node contains a piece of HLSL code, which is designed to perform a specific task. [11]

To put it simply, most materials contain textures, some math and they have some settings enabled or disabled according to what the material is going to be applied to. One of the simpler and also quite useful particle materials used in visual effects is the radial gradient, as seen in figure 11.

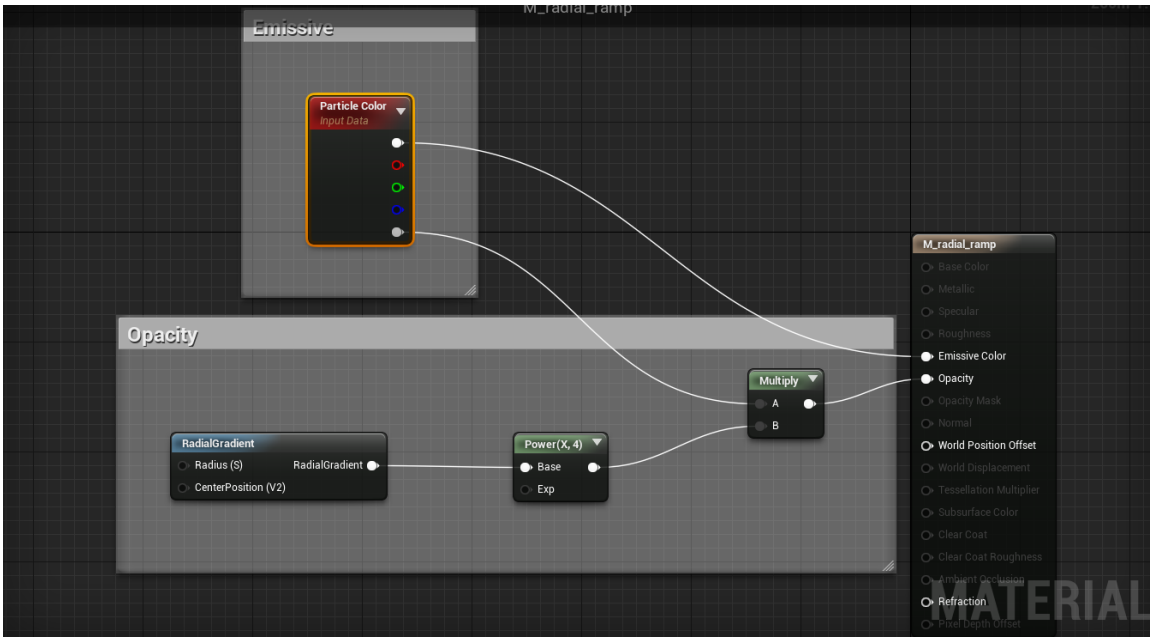

Figure 11. Radial gradient particle material nodes

This material produces the following particle, which is a simple white dot (See figure 12).

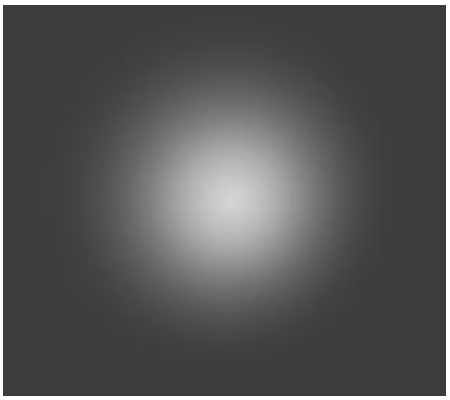

Figure 12. Radial gradient particle.

The color and opacity of the particle can be controlled within Cascade because the material contains the Particle Color node. The white dot texture is generated by a material function by using some math expressions. Materials can be affected through particles in Cascade by using a node called Dynamic Parameter within the material. This is useful for visual effects in many ways, such as the example seen in the following image (See figure 13). The figure contains a part of a fire particle material, which is created mainly by multiplying three rotating and panning textures together. To give the particles more variance, an offset is added to each texture in order to change which part of the texture is used for each particle.

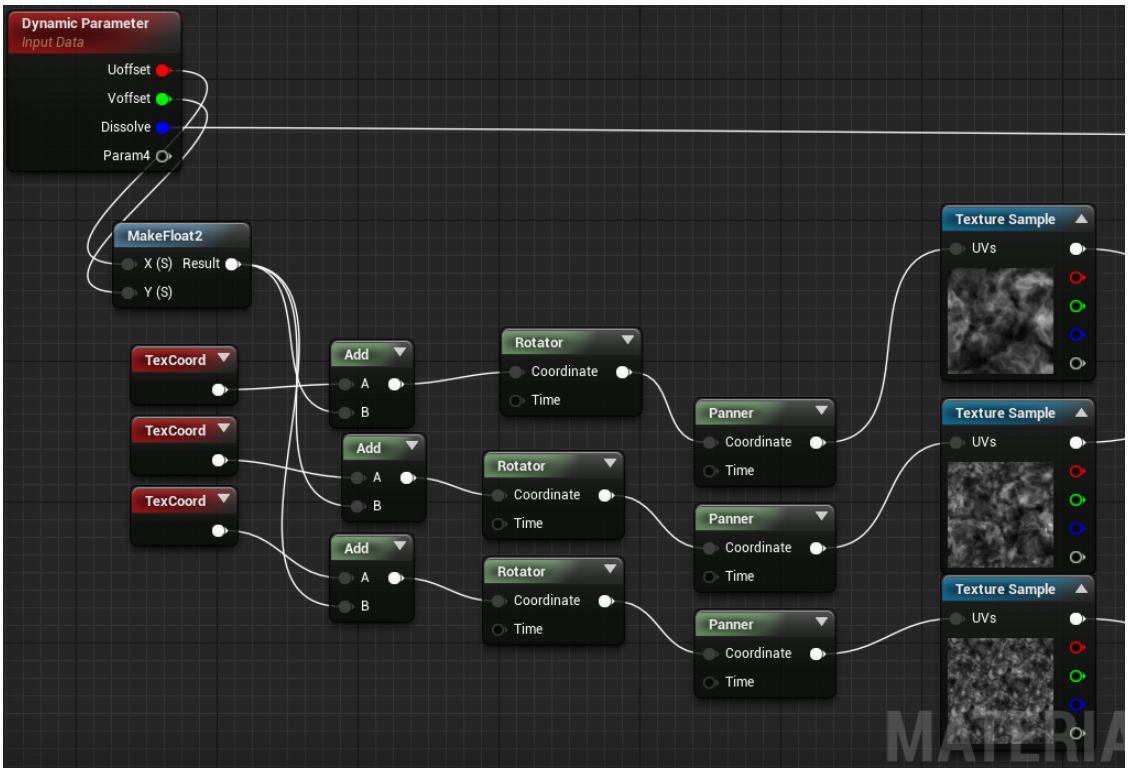

Figure 13. Dynamic Parameter node is used to randomize an UV offset

For an artist, the material editor is a priceless tool. It is fairly easy to get into thanks to the node based structure and extensive documentation by Epic Games. From a visual effects viewpoint, setting up materials represents the more technical side of creating visual effects. Understanding of math, especially vector math is quite beneficial when attempting to create and understand materials. Not much can be said in detail about creating materials for visual effects, as the implementation of materials is so heavily dependent on its intended use. At least personally, I find myself creating a new material quite often to meet the needs of the effect I am working on, instead of using a master material and creating material instances from that.

5 EXAMPLE MAGICAL EFFECTS AND BREAKDOWN

5.1 Ice spikes

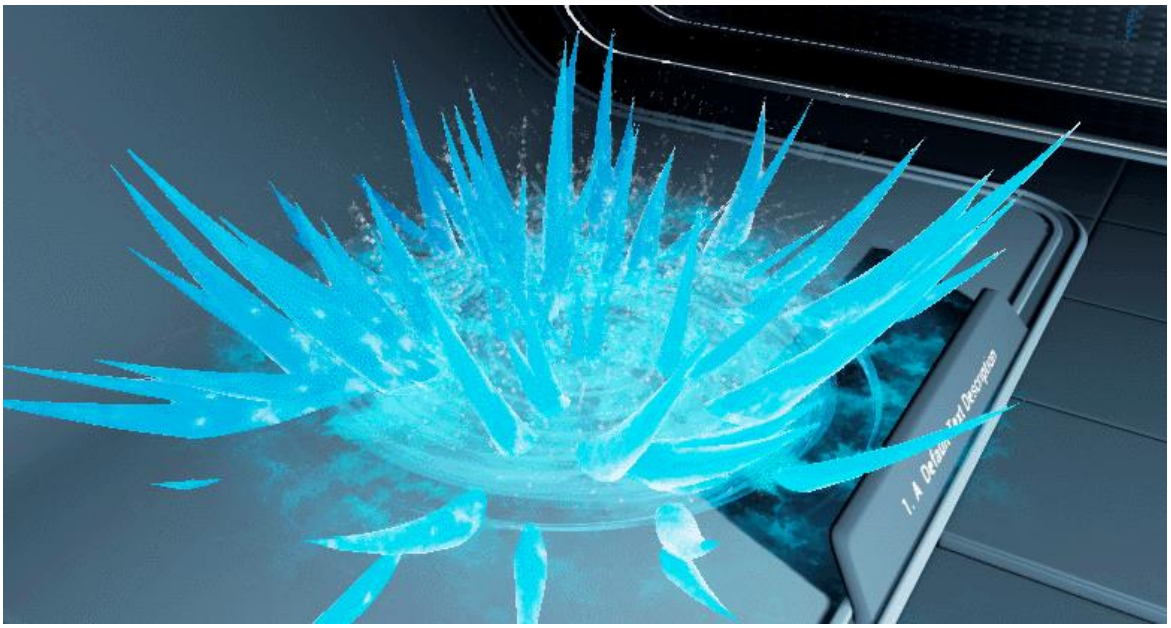

Figure 14. Ice spike magic effect. See video here.

When I started working on this effect I did not really have any specific reference in mind directly. Looking back on it my subconscious mind may have been drawing reference from a similar looking visual effect, as seen below, from Path of Exile, created by Grinding Gear Games (See figure 15). It is always good to have some reference as it gives an initial direction to follow when creating a visual effect.

Figure 15. Ice crash spell in Path of Exile. Screenshot from Path of Exile.

Sadly, the file format of this thesis do not support moving images so these still figures are the most I can provide within this document. Hyperlinks to video footage will be included to each effect in the image descriptions, as moving image is the only viable medium to communicate visual effects properly.

The key to make this effect come together was to create nice timing for the spikes themselves. Quite a bit of iteration went into how the spikes should appear. Through trying out different timings and spike orientations the end result of three quick but distinct successive waves of spikes was achieved. The spikes that are being spawned change their initial rotation of the mesh throughout the duration of emitter. This is done by using the Initial Mesh Rotation module in the mesh emitter that spawns the spikes. The distribution of the rotation values are set according to a uniform curve, as seen in figure 16. Uniform curve simply means that the value of the rotation is chosen between a minimum and maximum values of two curves.

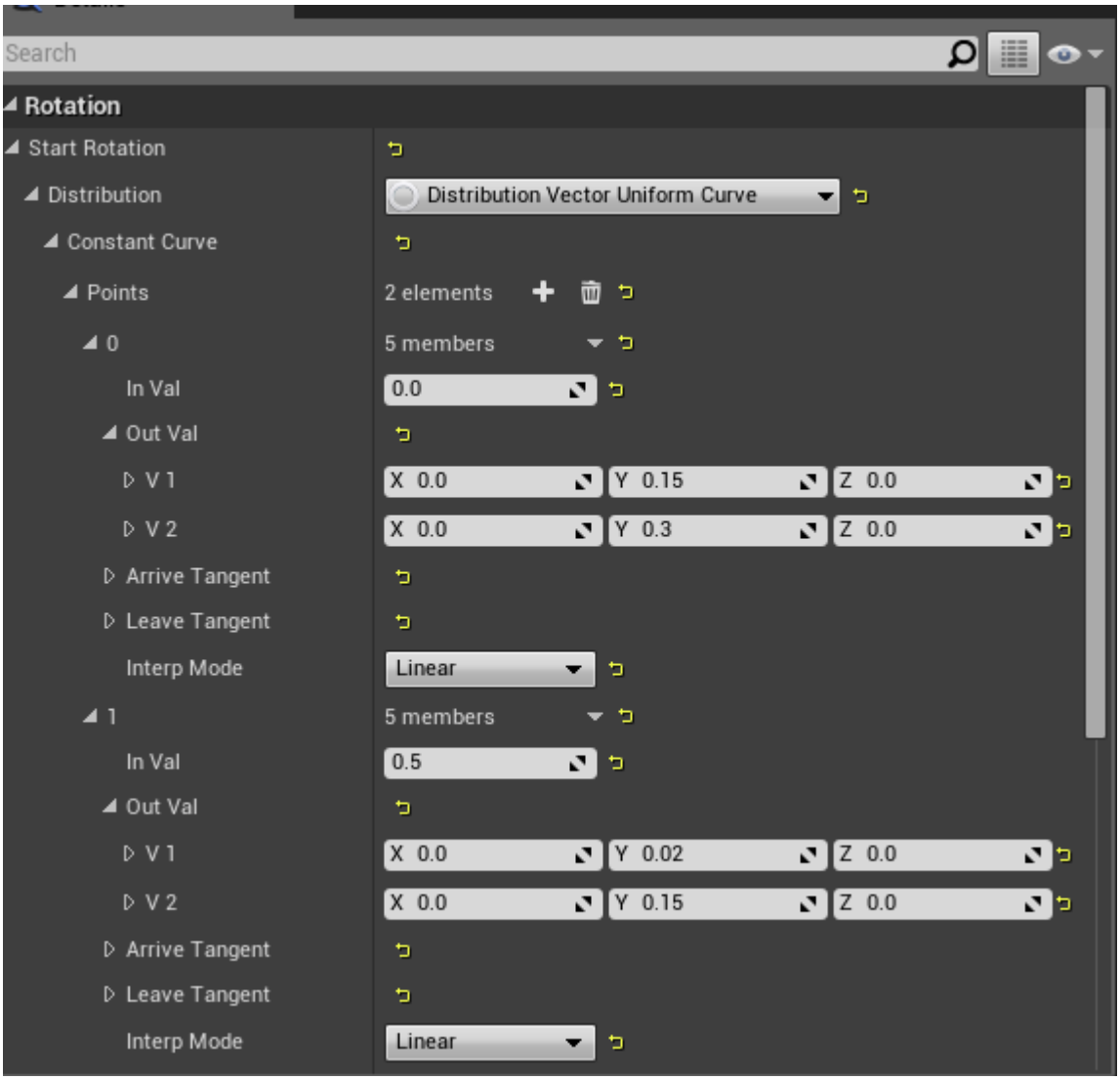

Figure 16. Setting the initial mesh rotation according to a uniform curve.

As shown in the image above (See figure 16), at the beginning of the emission of particles at time 0.0, the pitch of the particles is set to a value between 0.15 and 0.3. In this case, 0.0 pitch rotation means the particle is level with the ground and 0.25 rotation would have the particle pointing upwards at a 90 degree angle. Thus, at first the particles are pointing upwards between angles of 54 and 108 degrees. At time 0.5, which is when last of the particles have been spawned, the particles being spawned are now pointing between 7.2 degrees and 54 degrees.

The spike itself is a simple curved spiky mesh, as seen in the following image, which are used as mesh particles (See figure 16).

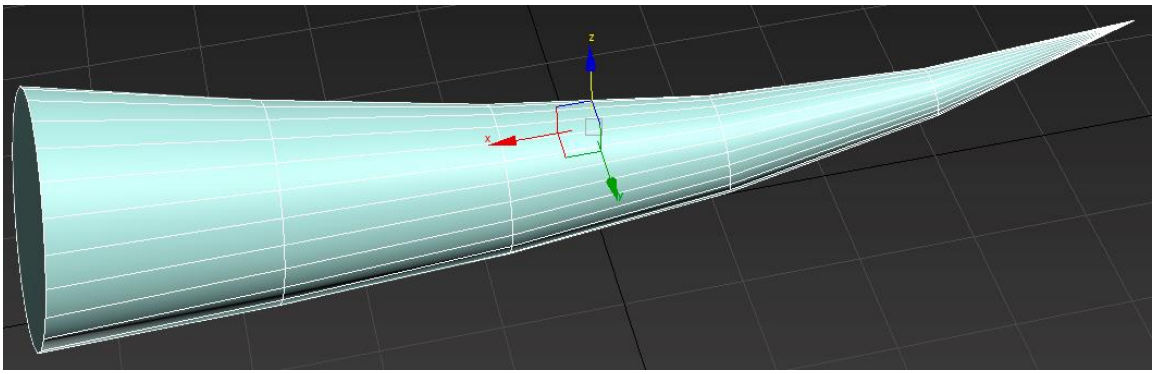

Figure 16. Ice spike mesh

The spikes icy appearance is achieved through a material that mainly utilizes Fresnel and subsurface shading, as the following image demonstrates (See figure 17).

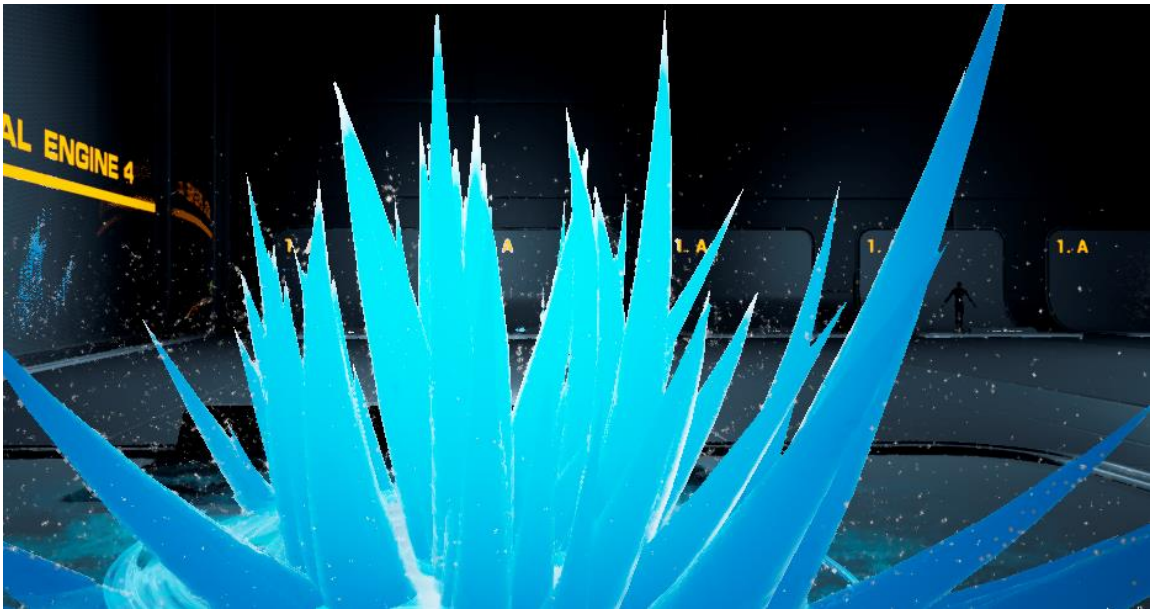

Figure 17. Subsurface shading shown on the ice spike material

When creating this effect I had in mind that the spell in question would be used by the player himself, hence there is very little anticipation to the effect as a whole. In the beginning of the effect there is a shockwave that adds to the punchiness of the

effect. The frost on the ground and snowflakes are there to give flavor and fitting detail to the overall effect. They give the impression that the spell's power pushes out with great force as the snowflakes are flung outward from the center and the cold powers at play as the ground freezes below. The frozen ground texture can be seen in the following image (See figure 18).

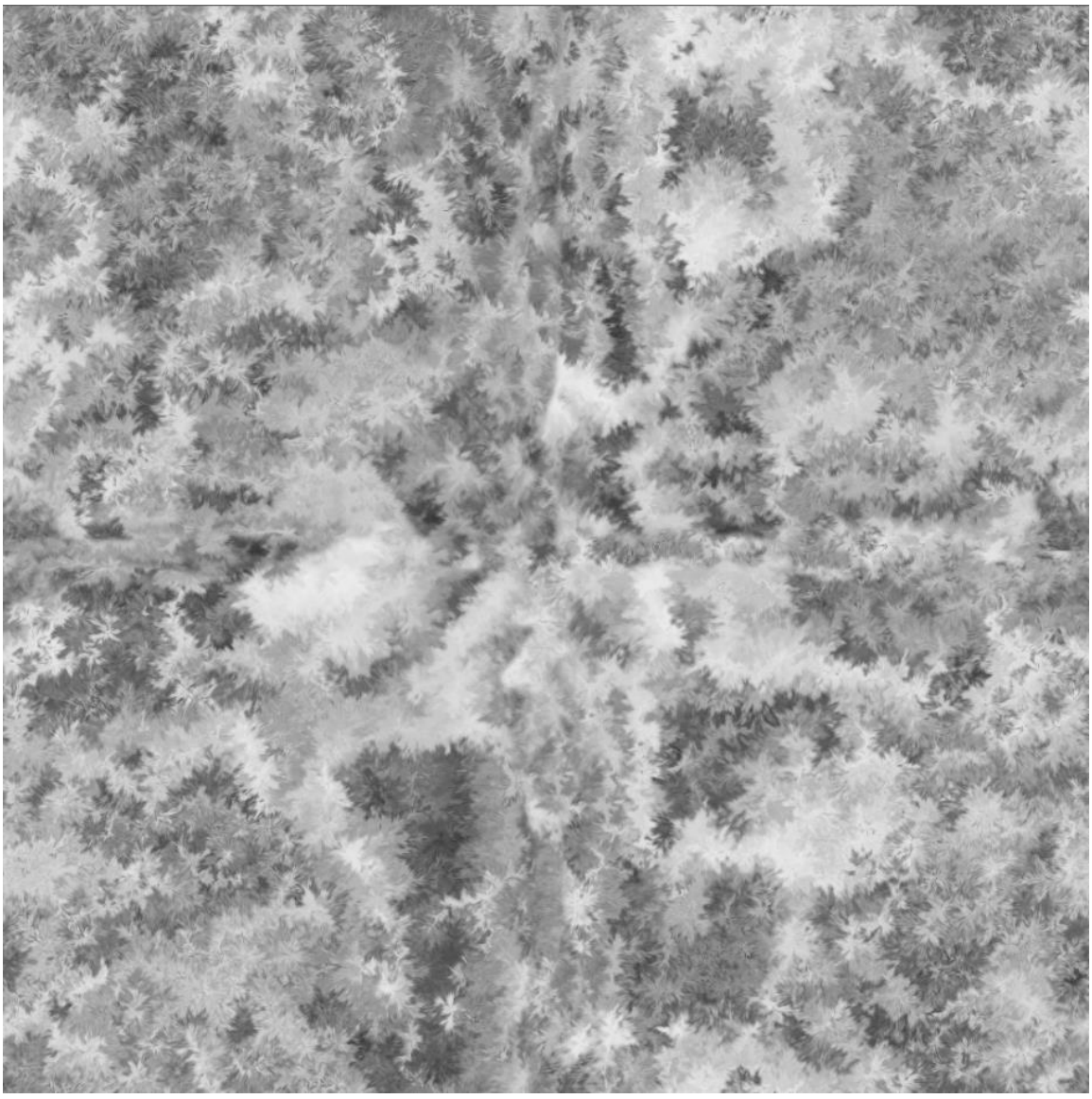

Figure 18. Frozen ground texture.

The textures for this visual effect were mostly created with Adobe After Effects by manipulating noise into different textures by using various effects within After

Effects. The snow flake texture was generated from a mesh created by a great free 3ds Max plugin called Debrismaker2, created by Aaron Dabelow.

5.2 Freezing spell

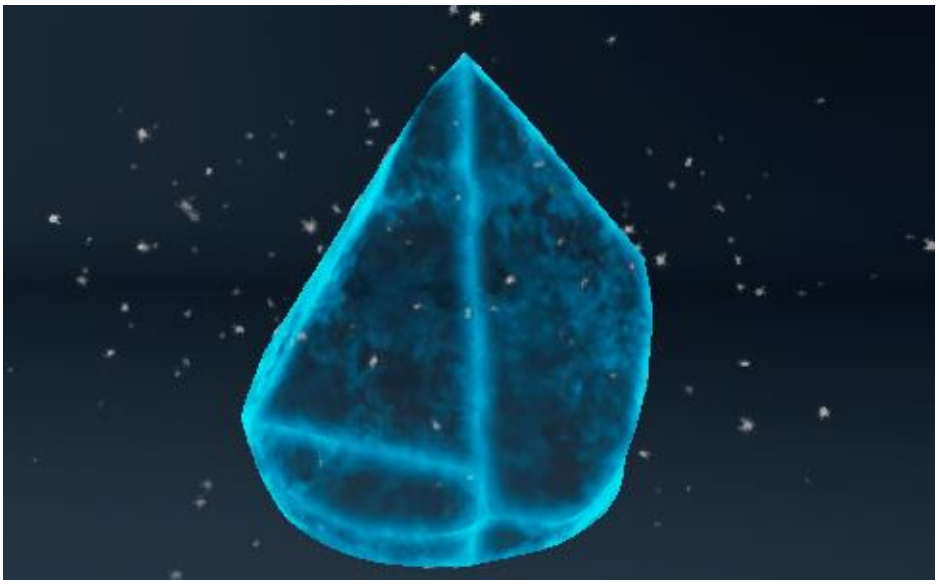

Figure 19. Freezing spell. See video here.

I wanted to create this effect mainly because I had worked on a similar one in Ancestory but I was not in charge of creating the ice block mesh in that one. I wanted to use this opportunity to make my own, and maybe improve on the overall effect as whole compared to the one I had created last year. I drew inspiration to the look of the ice block itself from mainly from World of Warcraft (See figure 20) to give me a starting direction how the ice block should look to make it appear to be made out of ice. I wanted to keep the ice block fairly translucent, as the idea of the effect and spell is that something gets frozen inside the ice block, while keeping the frozen object easily recognizable

Figure 20. Ice block from World of Warcraft. [12]

The main challenge with this visual effect was to make the ice block look appealing and convincing. The mesh itself is a good example on how to not create a 3D model that is to be used in games (See figure 21). It is not optimized for use in games as it has unnecessary polygons, which would cost more in terms of performance. Having good topology and baking normal maps from a detailed high poly version of the mesh to a low poly is preferred in 3D models used in video games.

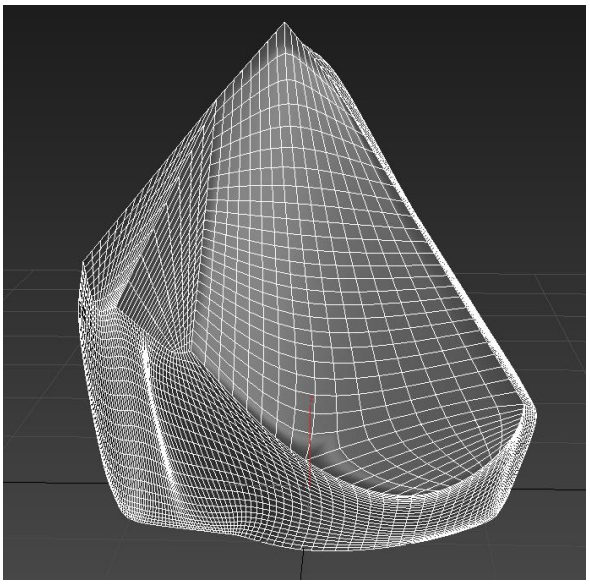

Figure 21. Ice block mesh

It took quite a bit of iteration with the ice material of the mesh but in the end the result was achieved quite simply. To achieve the frosty, icy look for the ice block I used the same ice texture as in the previous effect as base. The texture is multiplied by up- and downscaled versions of itself in order to make it different from the base texture, then simply multiplied with a pale blue color to make the ice block appear icy and frosty on the surface. This alone was not enough though, as the seams on the UVs on the mesh do not match with the texture. This was solved simply by taking the UVs to Adobe Photoshop and covering up the seams (See figure 22).

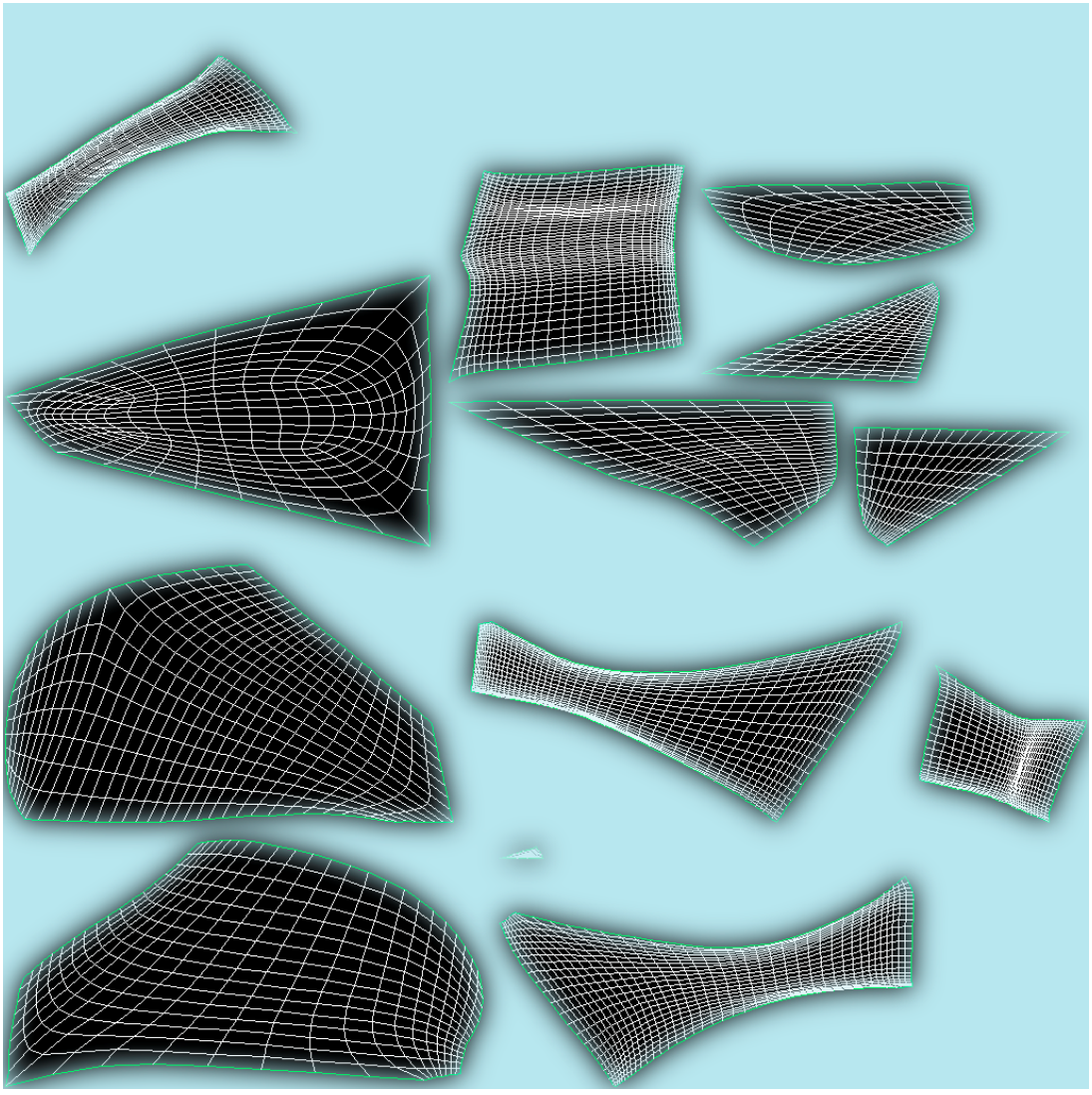

Figure 22. Covering the seams, the actual seams that need to be covered are highlighted in green.

The edges that are more perpendicular to the camera viewing the ice block mesh are highlighted by using the Fresnel module, as seen in the following image (See figure 23).

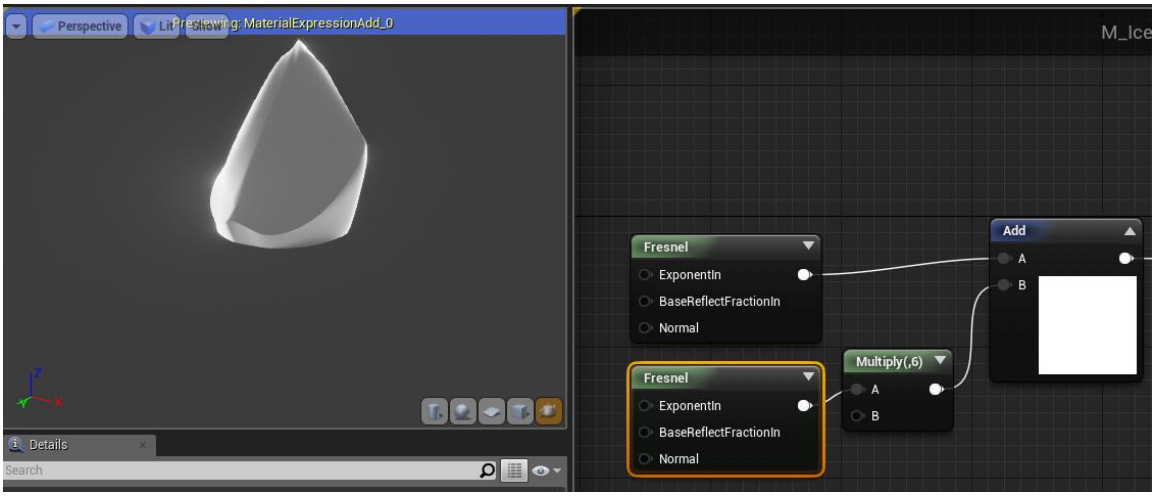

Figure 23. Fresnel nodes used to highlight edges

As described by the documentation of Unreal Engine, Fresnel is a term used to describe how light reflects at different intensities according to the viewing angle. This can be seen in everyday life, for example with water in a lake. If you look directly down while standing by a body of water, the water appears quite translucent and not that many reflections can be seen. However, if you look further away where the surface of the water becomes more parallel to your eye level, more and more reflections will be visible and the water will not appear translucent anymore. The following image demonstrates this effect (See figure 24). As the camera in the image is pointing toward the sphere center, the camera is facing directly at the surface normal. This is seen in the above image as the more grey areas of the ice block. As the surface normal becomes more and more perpendicular to the camera, the Fresnel effect increases in visibility. This can be seen as in the more white areas of the ice block mesh (See figure 23). [13]

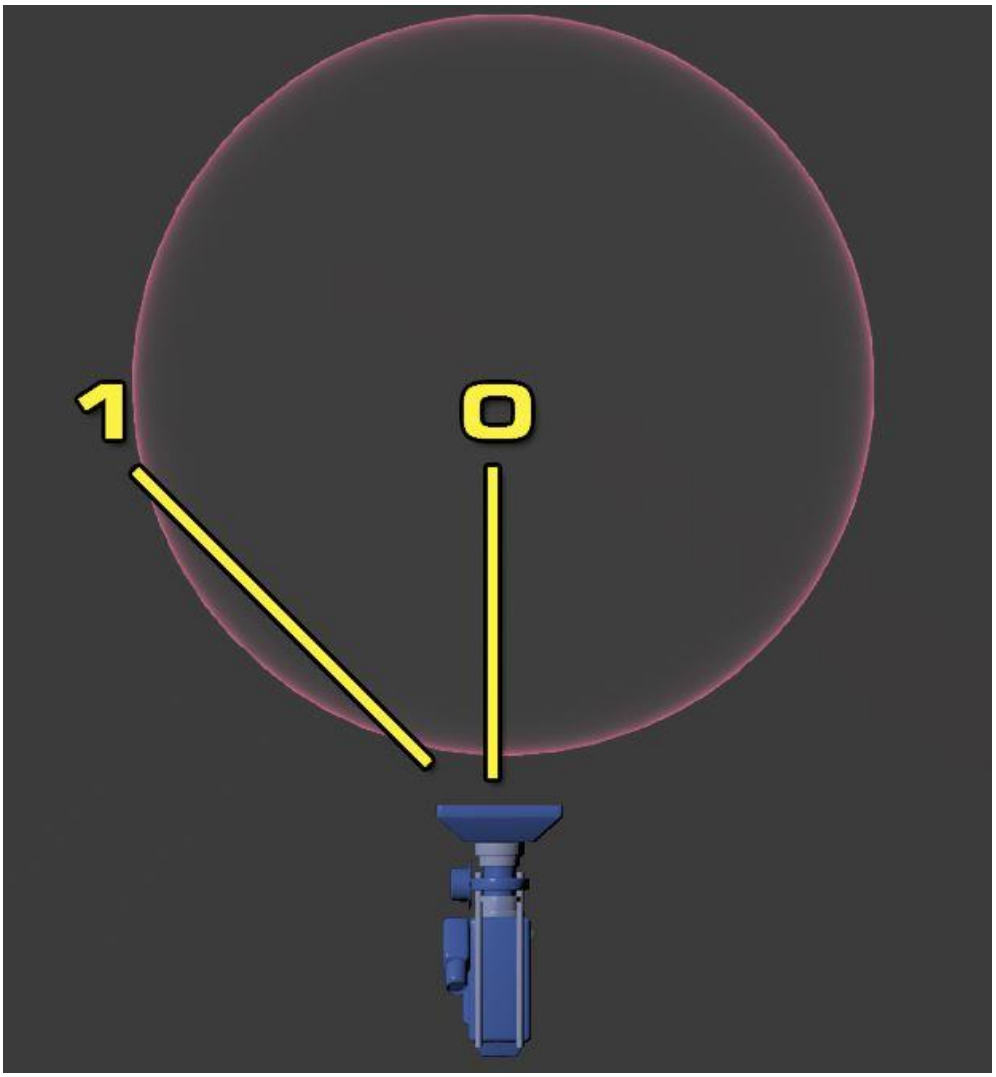

Figure 24. Demonstrating how Fresnel works [13]

The ice block needs to appear from somewhere, in this case it is simply scaled up from nothing to its full size quickly, while being obstructed by other particles as having an ice block growing seemingly from nothing is not great. The part of the effect that obscures the ice block whilst it grows consist of stretched out spheres with a fairly translucent material. The material uses a noise texture which has been processed in After Effects to transform the noise into quite cloudy appearance, as seen in the following image (See figure 25). I used the same snow flake particles as in the previous effect to communicate the cold forces at work in the effect and the same shockwave texture beneath the ice block to give the effect some punchiness and show the point of origin of the magic.

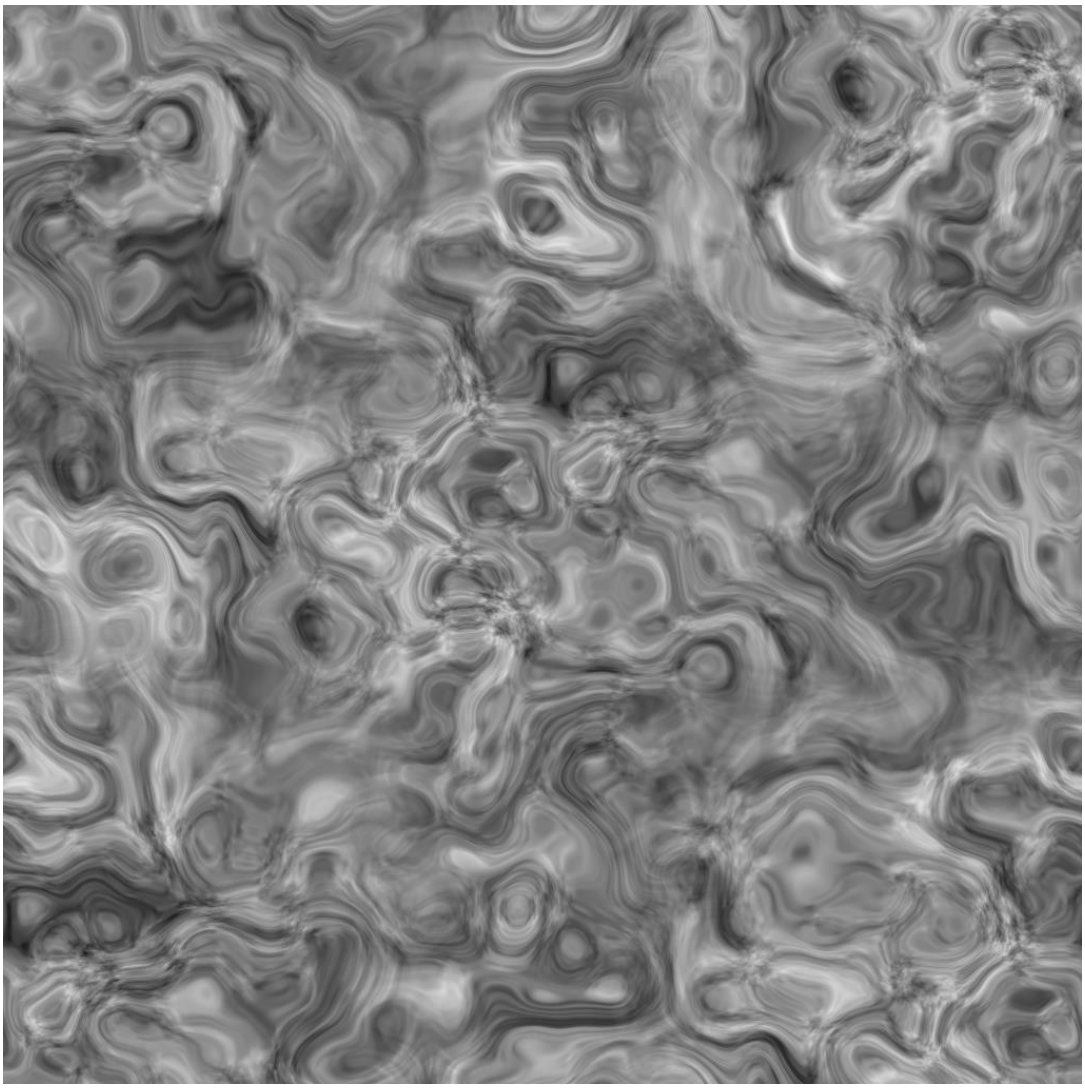

Figure 25. Processed noise texture

5.3 Meteorite

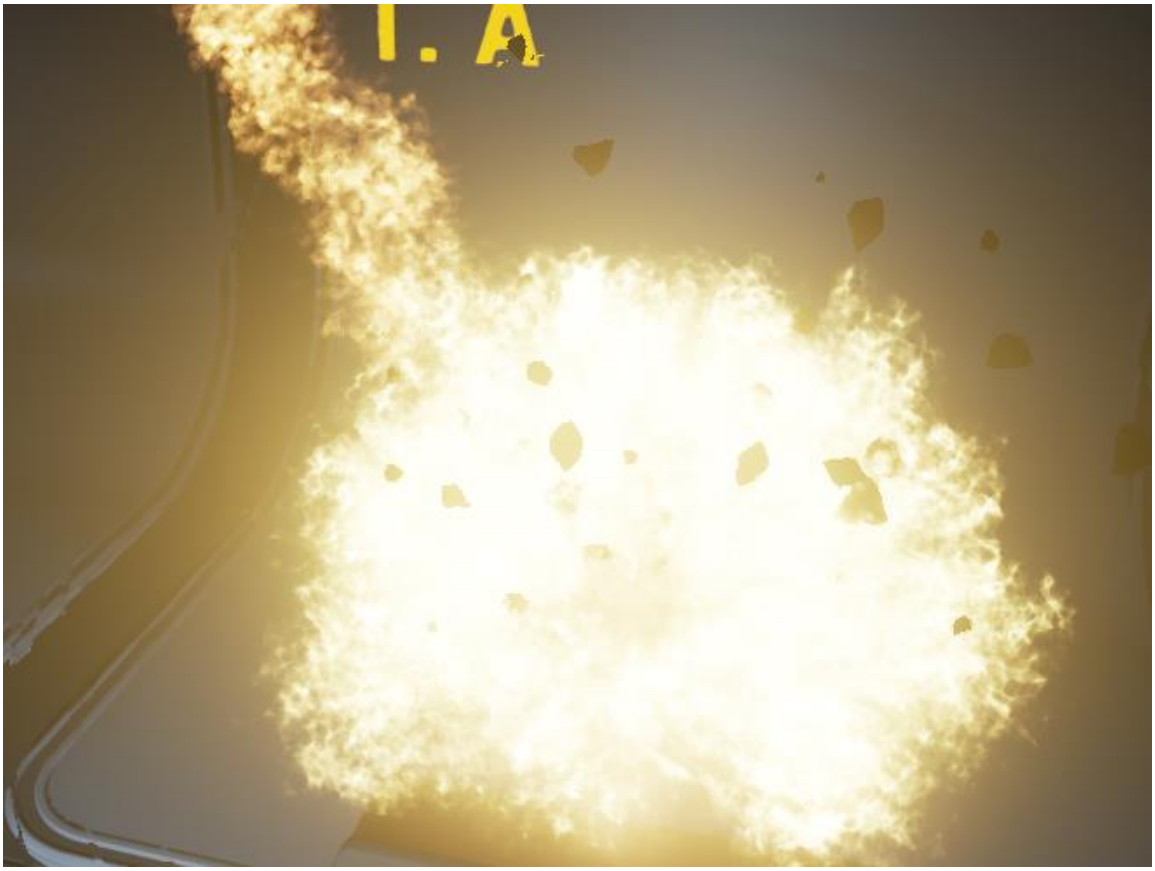

Figure 26. Meteorite effect. See video here.

A similar effect was done in Ancestory. Like in the previous example, I wanted to see how it be improved, since the time for extensive iteration was not simply there when Ancestory was being developed.

The most important part of the effect is explosion that occurs when the meteor hits the ground, but it is also important to get a nice trail for the meteor as it hurtles through the air. A fire texture was generated in After Effects for the explosion particles. In order to make the fire look more convincing, it requires internal motion of some sort. This was achieved through a method that was presented by Julian Love in his GDC 2013 presentation “Technical Artist Bootcamp: The VFX of Diablo”. It works as follows, take textures at different scales and multiply them together while panning them at different speeds (See figure 27). It is quite simple

but with this method one can generate surprisingly complex shapes and motion. [14]

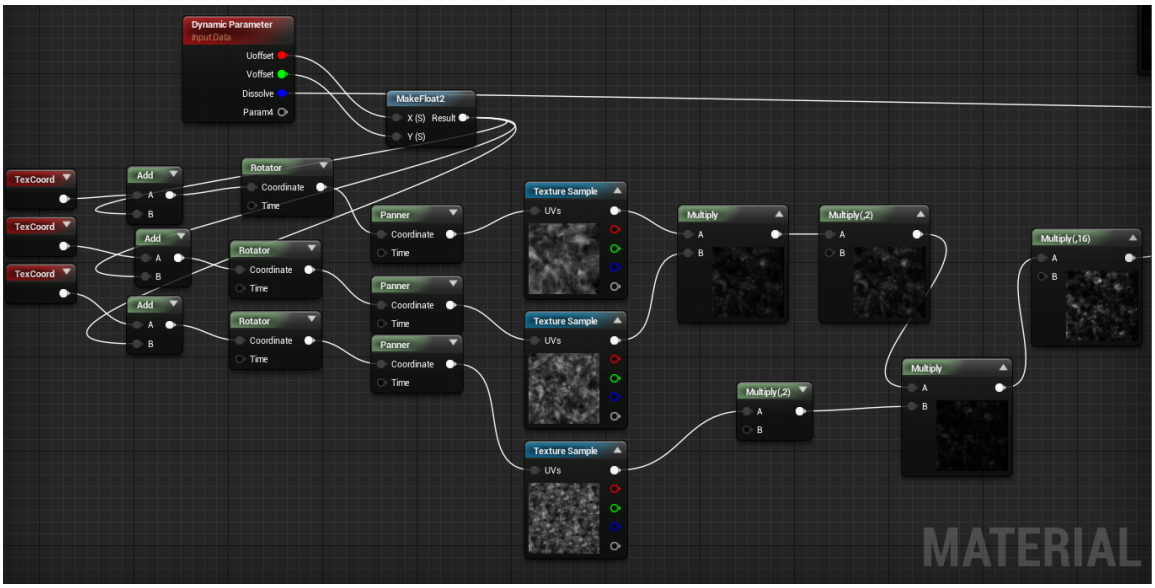

Figure 27. Fire texture multiplication and panning

The same fire is used both in the explosion and in the trail of the meteor. It should be noted as the textures are being multiplied together the overall brightness decreases. In order to compensate for this, the multiplied texture can be simply multiplied with a scalar value to bring up the values to desired level. The meteor mesh, as seen below (See figure 28), was generated using Debrismaker2 in 3ds max. Normal maps were baked from a high poly version of the mesh to give the low poly mesh some additional detail.

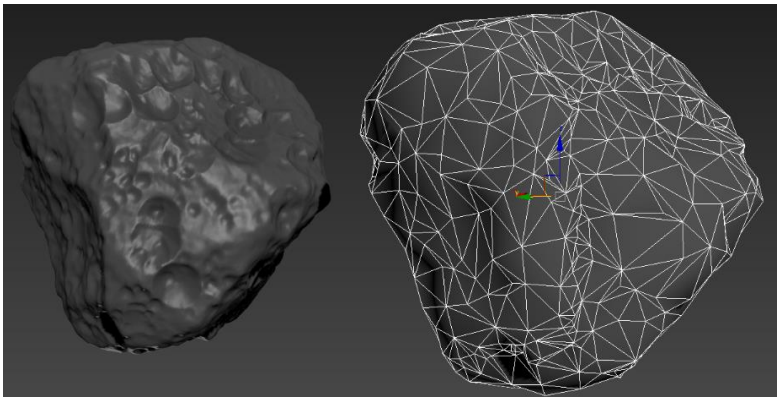

Figure 28. Meteor mesh, low poly on the right, high poly on the left

Figure 29. Meteor falling down

The meteor, while falling down as seen in the above image (See figure 29), is glowing orange to make it seem like that a great amount heat is being produced as the meteor cuts through the air. This is done in order to make the whole effect seem more powerful. When it hits the ground the mesh is destroyed and the explosion particle emitters are triggered. These emitters include the explosion, rocks that are flung from the center of the explosion at great speed as shrapnel, and the impact mark on the ground itself. The impact texture is again generated from processed noise texture and further processed within the material editor of Unreal Engine 4.

5.4 Lightning

Lightning effects can be tricky to create. Luckily I had some experience in creating some lightning effects in Unreal Engine 4 from working on Ancestory. Looking at reference is a great help when creating lightning, as it gives some clear ideas how the effect should behave. The main characteristics of lightning are its short amount time it’s visible, brightness and shape. The following examples of lightning were created using the beam type data module within Cascade. The beam type data module enables the emitter to have beam source and target location modules and noise modules, as seen in the following image (See figure 30).

Figure 30. Beam emitter in Cascade

As the name suggests, beam emitters are used to create different kinds of beam effects. They are very useful when creating lightning effects, as the noise module can be used to produce the desired noise behavior and visual appearance.

The material used for the following lightning effects is quite simple, as seen in the following image (See figure 31). The texture being used is generated with some simple math inside the material editor, which is then plugged into the Particle Color node to control the color and opacity of the particles.

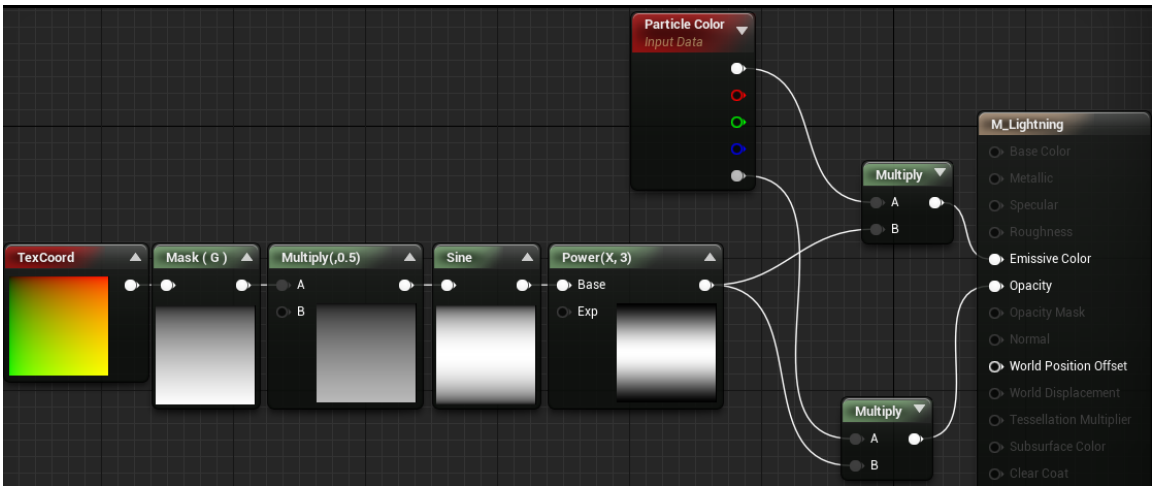

Figure 31. Lightning beam particle material.

The sheer brightness of lightning and electricity is done by using the Scale Color/Life module in Cascade. This module simply scales up existing color values that have been set in the emitter over particle lifetime. For example, in the following thunder bolt visual effect, the color values are being scaled as demonstrated by the following image (See figure 32).

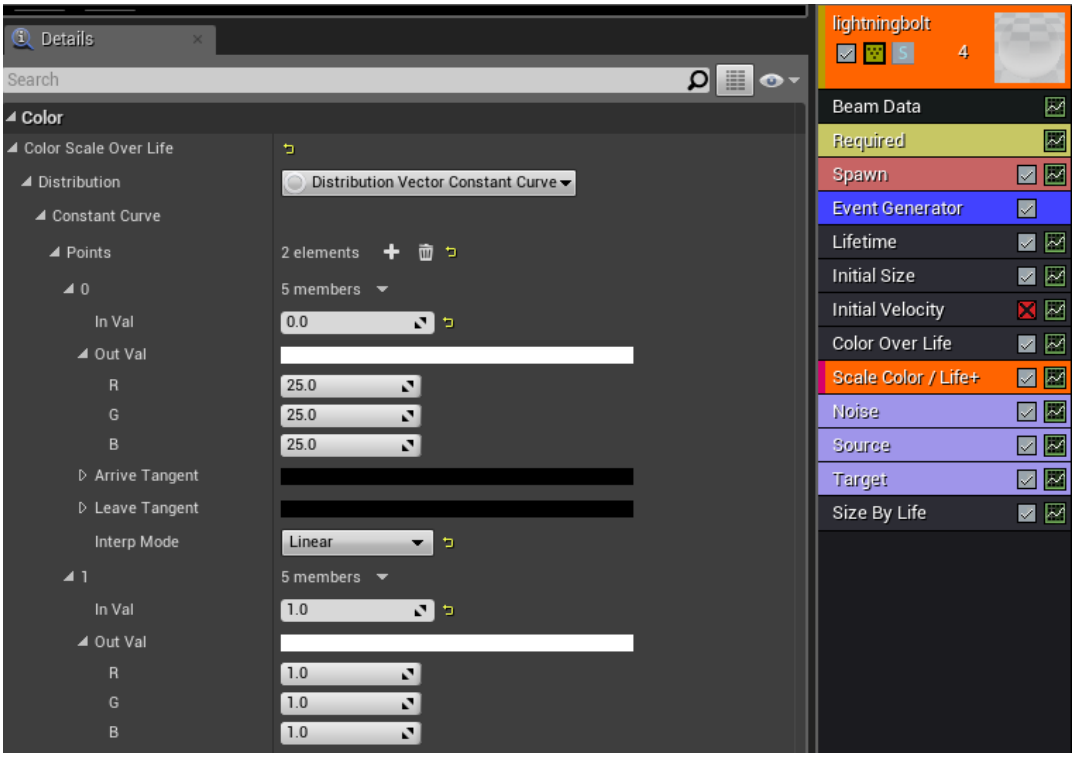

Figure 32. Using Scale Color / Life module to make the colors brighter

As seen in the image, as the lightning beam is spawned the color values are being multiplied by a factor of 25. The color multiplication is reduced to a factor of one linearly throughout the lifetime of the beam particles. This produces a clear flash in the beam of lightning. The color scaling also creates some bloom in the vicinity of the lightning beam, which fits the overall effect. This can be seen in the following image (See figure 33).

Figure 33 Thunder Bolt. See video here.

When creating a large thunder bolt that strikes down from the skies, it is good to have the noise of the beam locked in place in order to mimic behavior of real thunder bolts. It gives the thunder bolt a real sense of power and improves the readability of the effect as a whole. Secondary beam emitters are visible as they

strike down in the general vicinity of the main beam. These secondary beams add a bit of flavor to the effect while mimicking the branching nature of thunder bolts. As the thunder bolt hits the ground, a simple radial gradient particle is spawned with lights to create the flash of the lightning bolt. Additionally, sparks are flung out with great speed to demonstrate the force of the impact.

Figure 34. Sustained electrocution. See video here.

Smaller lightning works well with the noise not being completely locked. In this case the noise is locked to 0.08 seconds, meaning the noise is recalculated every 0.08 seconds. This creates interesting movement in the beam while keeping it under control, lower values would easily result in something that is too chaotic and unstable. Again, radial gradient particles are spawned as the beam emitter are destroyed to give the lightning its flash. Though in this case, the radial gradient particles are being spawned by an Event Receiver module. This module works by

receiving events generated by an Event Generator module in the lightning beam emitter. In this case, the Event Generator is set to generate an event each time a lightning beam is killed, as the lifetime of the beam runs out. The Event Receiver module receives the event of particle death and spawns one radial gradient particle with the light module, as seen in the following image (See figure 35). This produces a flash of light on the ground each time an arc of lightning hits it

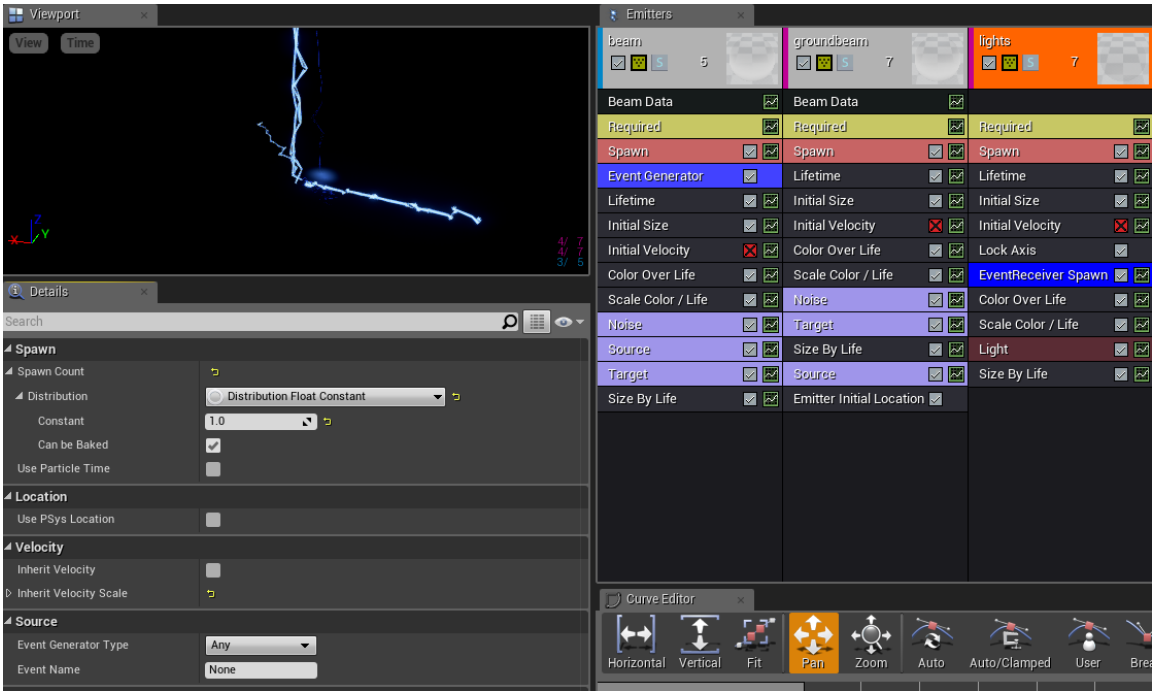

Figure 35. Using Event Receiver module to spawn particles as the lightning hits the ground.

In this effect I wanted to create the feel of more sustained electricity, making the ground electrified as well as the electricity keeps hitting the ground. This is achieved by spawning secondary arcs of lightning that target the surrounding area of the main beam of lightning.

Figure 36. Lightning variation. See video here.

This effect is a variation on the previous effect, it has more lightning beams to give the effect more power, and a defined ending. The ending is achieved by changing the timing of the lightning and reducing the amount of beams being generated drastically. This gives the impression of the source of the lightning running out of power as it discharges the last few arcs of lightning.

Figure 37. Lightning targeting bones of an animated skeletal mesh. See video here.

Another variation on lightning, this time the goal was to make the lightning target points on a skeletal mesh. Inspiration was drawn from Star Wars and Sith Lightning. It is fairly easy to create something similar using particle parameter functionality within Unreal Engine 4 and simply passing the bone world space transforms to the particles via a blueprint, as seen in the following image (See figure 38). On each frame, a random bone is chosen from the character mesh used in the effect. This mesh has 80 bones, and their identification numbers run from 0 to 79. The location of the randomly selected bone is retrieved using the Transform from Bone Space node. This returns a 3-dimensional vector value, which is then passed over to the particle vector parameter node. This node passes the vector value to the Cascade module that has the particle parameter “TargetLocation”. There are some secondary arcs of lightning spawned as each beam of lightning hits its target. These secondary beams have their source and target location handled by the last two nodes in the blueprint. The source location is the same as

the target location of the main beam of lightning, while the target location of the secondary beams is randomized into a space 10x10x10 unit cube around the source location.

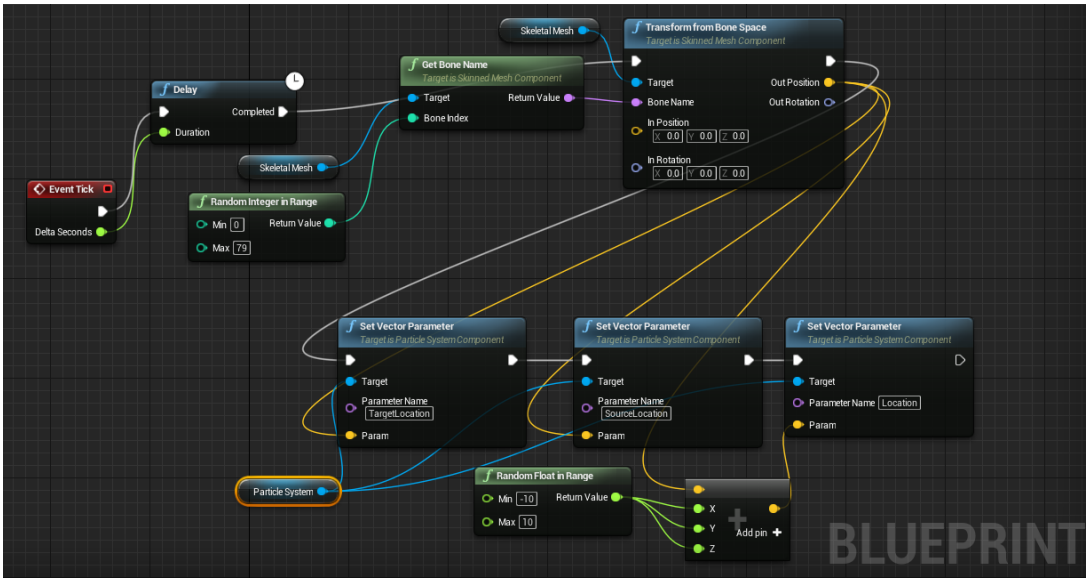

Figure 38. Using a blueprint to pass over bone locations to beam emitter in Cascade

5.5 Arcane explosion

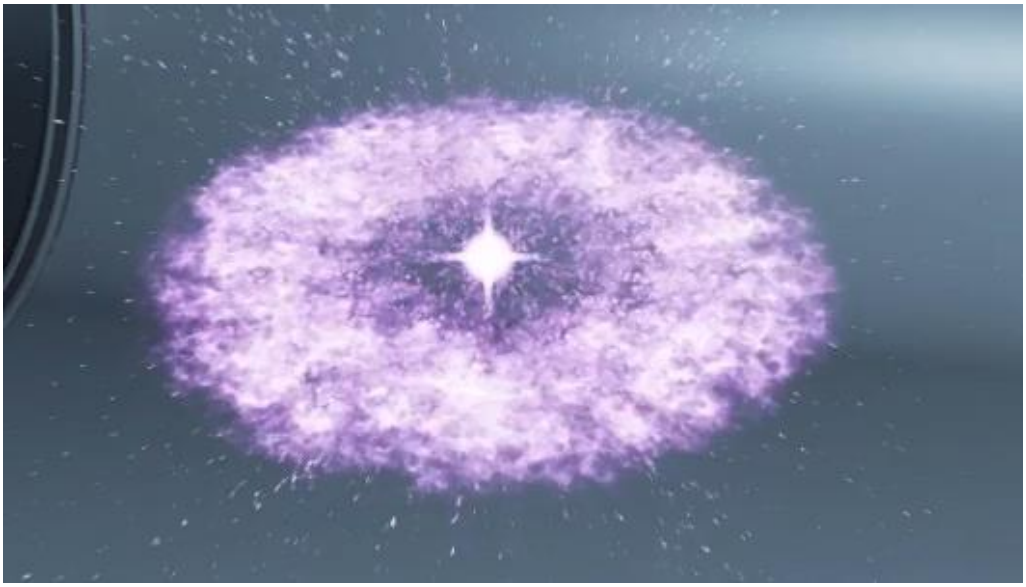

Figure 39. Arcane explosion effect. See video here.

This particular effect was an exercise in anticipation. The goal in mind was to create an effect that is easily anticipated in a game setting. The anticipation of the effect is mainly achieved by animating the scale of the sphere of energy that the whole effect is centered around. The scale of the sphere is animated according to the following curve, as seen in the image below (See figure 40).

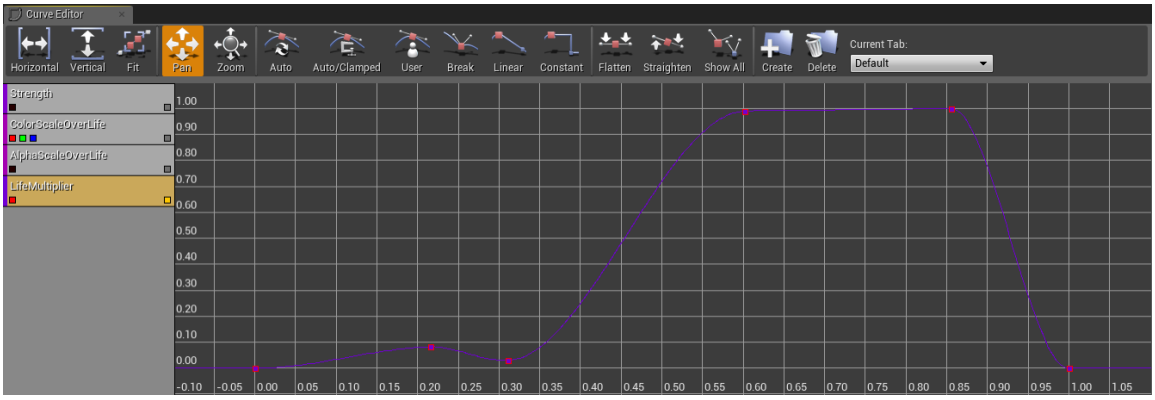

Figure 40. Curve inside Cascade used to scale the sphere mesh.

First it starts out small, drawing energy as shown by the stretched out radial gradient particles, then expanding to its full size. The following increase in brightness is key part of creating anticipation for the effect, as it is very clear that something is about to happen. Afterwards the sphere simply collapses and the whole thing explodes, triggering explosion particle, flash particle and shockwave particle emitters.

Another important part that needed some iteration on this effect was how the small particles behave as the sphere is being animated. It brings nice detail to the effect as the small particles are simply slightly attracted to the sphere of energy at first as it grows to its full size, then being sucked in to the center effect as the sphere collapses, and then being flung out as the explosion occurs. This is achieved simply through using separate emitters for each desired particle behavior and timing when each emitter should be emitting particles and for how long.

5.6 Arcane beam

Figurine 41. Arcane beam. See video here.

This effect was directly inspired by Heroes of the Storm. After seeing the effect in question I simply wanted to try and create something similar, as it is a great looking effect, as seen below (See figure 42).

Figure 42. Disintegration beam from Heroes of the Storm. See video here. [15]

Creating the effect took quite a bit iteration and experimenting, especially how to create the pulses in the beam. At first I tried to make the pulses part of the beam itself but that solution did not behave as I wanted. Instead, simply creating separate particles with a diamond gradient shape while stretching them along the direction of the beam resulted in the desired pulses in the beam. These particles that form the pulses can be seen in the image below (See figure 43).

Figure 43. Pulse particles used in the beam effect

By changing the draw order making the beam on top of the pulse particles it seems like the pulses are actually a part of the beam itself while they are not. The motion of the beam is simply generated by panning the textures that are being used. Draw order of particles in Cascade is determined by the position of emitters in the particle system. Draw order is sorted from bottom to top, as the emitters are stacked on top of each other from left to right in Cascade. As seen in figure 44, the beams are on the bottom of the draw order. This allows for the end and beginning points to be occluded.

Figure 44. Setting up draw order of the particle emitters within the particle system.

The ending and starting points of the beam must be occluded somehow, as it would not look very appealing with the sharp edges of them beam clearly visible, as seen below (See figure 45).

Figure 45. Occlusion is needed.

The starting point is occluded by creating a tear drop shape that tapers off along the direction the beam. This shape is created by spawning radial gradient particles and particles using the same material as pulse particles in the beam. These particles add some spikiness to the otherwise smooth shape to give it some variety as it is hard to see the movement of the overall shape from the smooth radial gradient particles alone. This shape is placed on top of the beam in regards to draw order in order to occlude the starting point.

For the ending point occlusion I wanted to create something closely resembling the reference. Again, radial gradients and the same pulse particles were used to create the necessary occlusion, the radial gradient define the general shape and the pulse particles add to the details, definition and motion of the shape. The pulse particles are simply scaled from zero to their full size, and back to zero as they are destroyed to give them motion through the particles lifetime. Variation is achieved

by randomizing the rotation of the particles when they are spawned. To achieve the same colors as in the reference using duplicate particle emitters with the desired color was fairly successful. This took some iteration but through trial and error the end result is fairly similar to the reference.

5.7 Magic portal

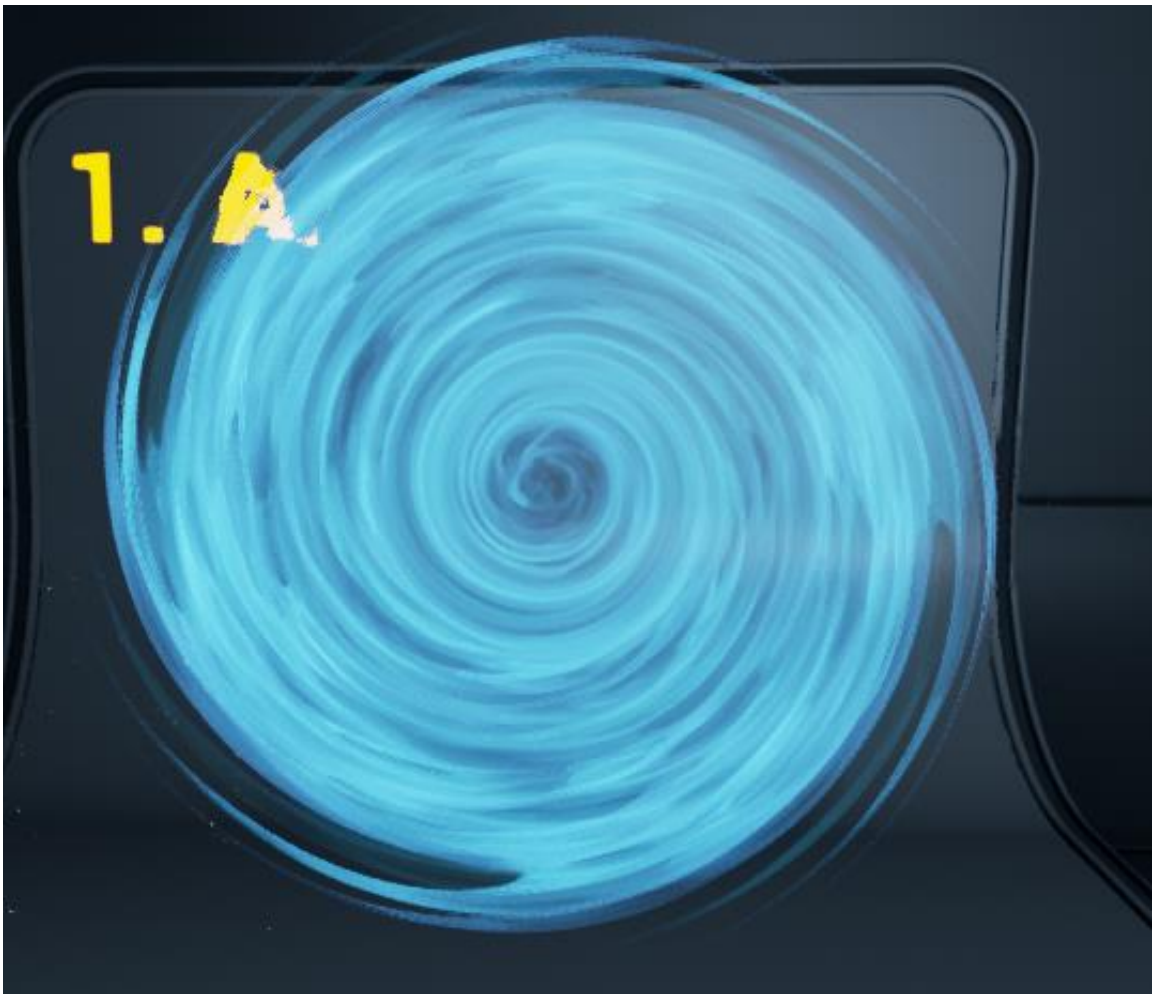

Figure 46. Magic portal. See video here.

This effect was created fairly quickly simply by using a noise texture that was processed into the texture featured in the particle effect. Giving this texture color as seen above and simply randomizing the initial rotation of the particle and

rotating the particles over time makes a fairly good looking portal. Adding some additional particles with refraction behind these particles makes it appear as if the portal is bending space itself, which is quite fitting to a magical portal. The effect could be further improved by creating additional geometry or structure to support the portal effect, though it works fine as shown (See figure 46).

Using different colors schemes with the portal is a great way to give an idea about what kind of place it might lead to. Using neutral or positive colors such as blue or green would appear not that threatening to the player. If colors like dark red or such are used then it would be quite easy to understand that the place the portal leads to could be quite dangerous (See figure 47).

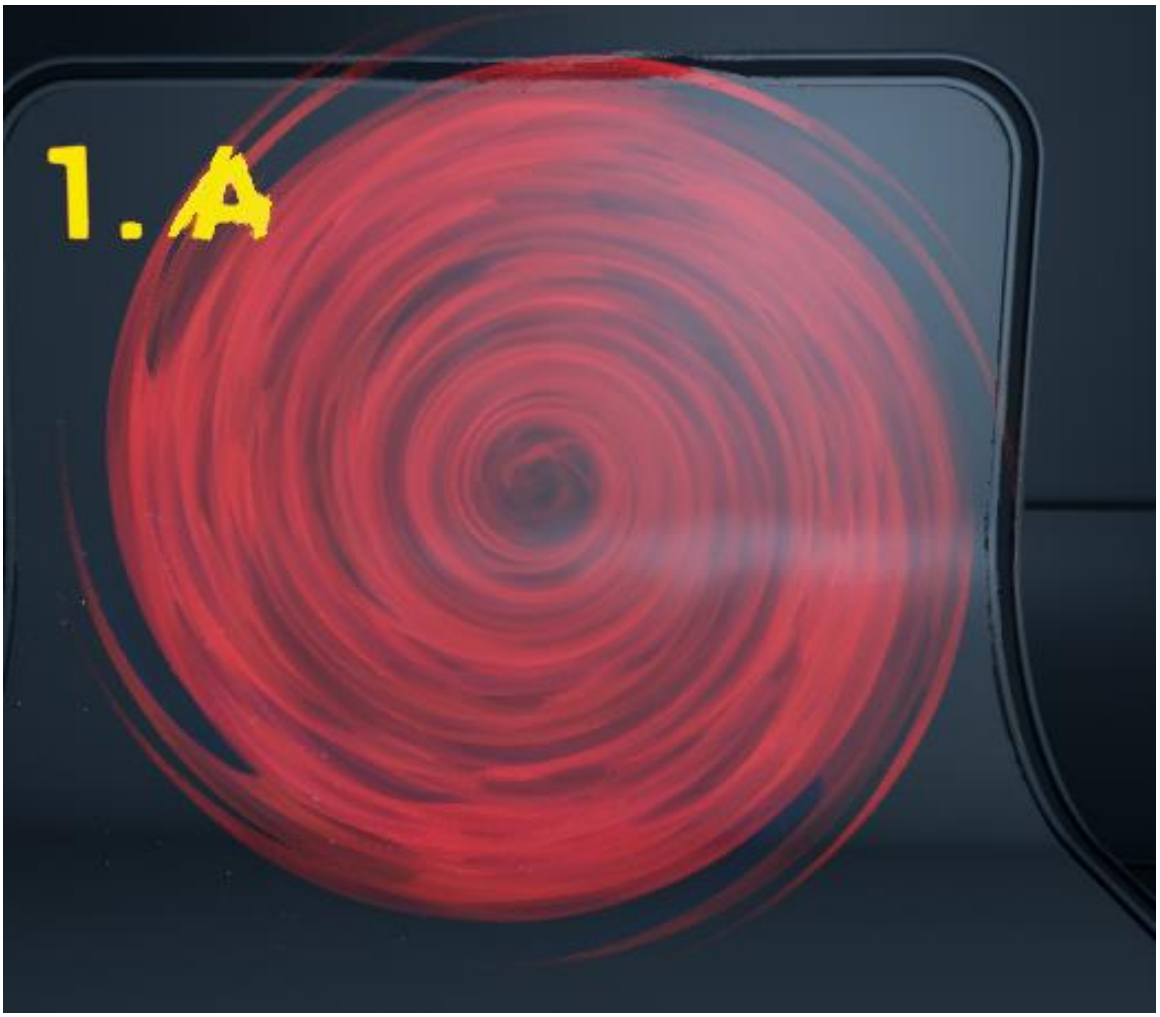

Figure 47. Portal dark red color scheme

5.8 Holy shield

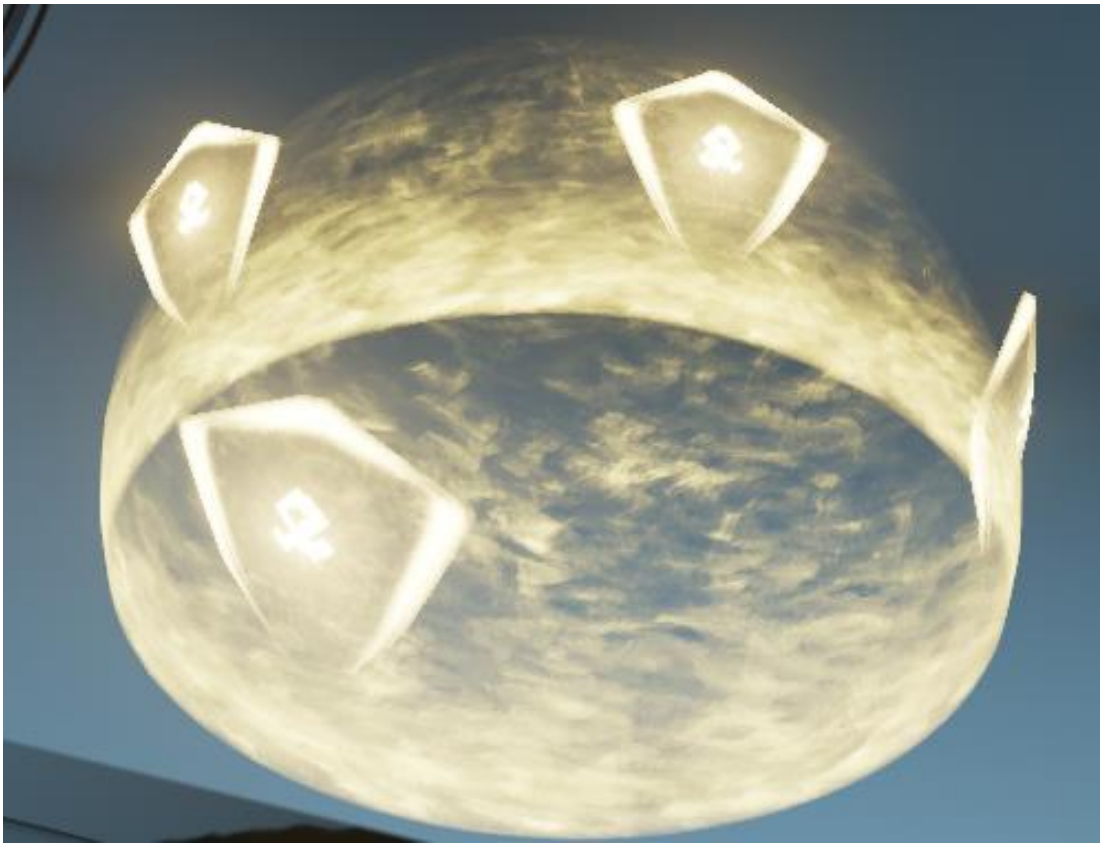

Figure 48. Holy shield magic effect. See video here.

There was no direct visual reference for this effect, though a game mechanic was in mind. The purpose of this magic spell would be to protect allies from harm by providing cover to them in a form of a defensive bubble. A holy warrior, like a paladin, could call down this protective blessing from the heavens onto his friends.

In order to make the visuals match the game mechanics and setting of this spell, there are a few things to consider. First, it makes sense that some visual element actually comes from above considering the narrative background of the spell. It also gives a chance to have anticipation with the overall effect. Secondly, the shielding effect needs to be fairly translucent as the things being shielded should be visible from the outside for obvious reasons. Thirdly, it should be fairly clear

that the gameplay effect that the spell provides is positive, this is achieved with the colors and the rotating shields around the bubble itself.

The bubble mesh uses a panning processed noise texture to give the effect motion and details, additional motion is achieved by simply rotating the bubble meshes over their lifetime. The initial sweep from top to the bottom of the bubble is achieved by controlling the panning of the textures in the material by using dynamic parameters to control the values directly. This can be seen in the image below (See figure 49). Using a curve inside Cascade to change the values of the “PannerTime” parameter results in the top to bottom sweep as seen in the effect.

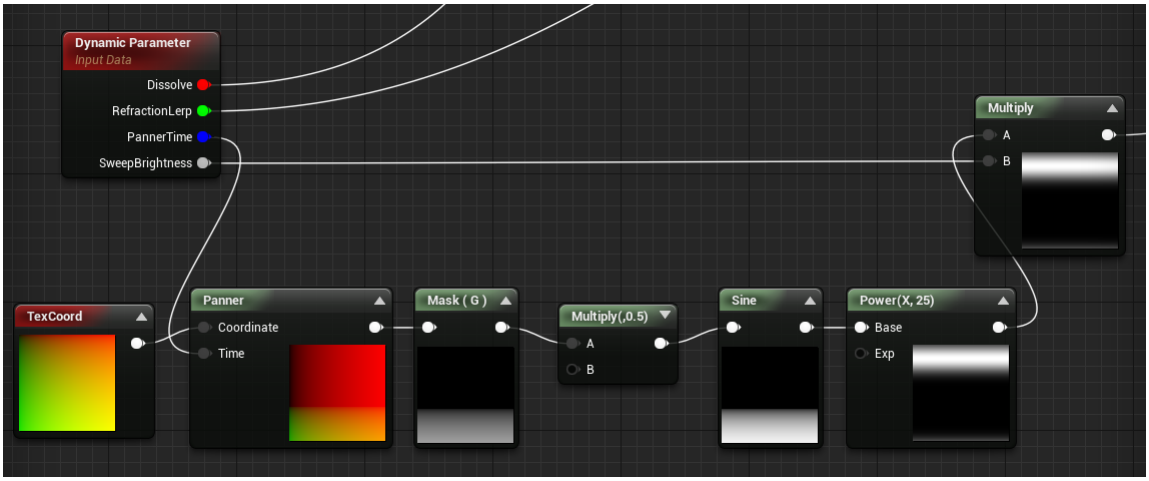

Figure 49. Controlling the time component of a Panner node using Dynamic Parameters

A slight amount of refraction is used on the bubble to add small detail to it, it would mainly be useful in a situation where the camera is close to the top of or sides of the bubble, as in the screenshot below (See figure 50). The refraction would only slightly distort any objects or characters if they were inside the shield.

Figure 50. Refraction brings subtle details to the protective bubble

6 CONCLUSION

The main goal and purpose of this thesis was to use it as an opportunity to learn. Of course, there are still many things to learn about. Some things that I wanted to try to create in this thesis, such as creating hand animated visual effects, had to be dropped due to time. Though there's still much to learn and I strongly feel that I am still a mere novice at this, I would say this thesis has been a successful and fulfilling project for me. I am pretty happy with most of the effects presented in this thesis, some could use some more polish but overall the results are satisfying.

Having the basic skills of 3D modelling is essential when working with visual effects. For me, making this thesis was a great opportunity to strengthen those skills. I only had limited experience working with 3D modeling software prior to working on this thesis, as I am coming from a sound design background. In terms of acquiring skills, the next logical step for me would be to start learning how to create animated visual effects and how to create more stylized textures.

I have been focusing on and working on visual effects for roughly a year now. I am fairly sure at this point that the best way to learn how to create visual effects in games is by doing. Which is probably part of the reason why there seems to be no books written on video game visual effects, as the industry and its practices are changing so quickly. Though the value of having a solid foundation of skills 3D modelling, 2D graphics and animation cannot be stressed enough. Having skills in scripting or math should not be disregarded either, though this is largely dependent on the game engine that you happen to be using.

7 REFERENCES

Books

2 Gilland. Elemental Magic: The Art of Special Effects Animation. 2013th ed. 2 Park Square, Milton Park, Abingdon, Oxon OX14 4RN: Focal Press; 2009.

3 Gilland J. Liquids. Elemental Magic: The Art of Special Effects Animation. 2013th ed. 2 Park Square, Milton Park, Abingdon, Oxon OX14 4RN: Focal Press; 2009. p. 102-103.

Websites

1 [Legion] Unholy Death Knight Spell Animations. 2016; Available at: https://www.youtube.com/watch?v=p04IdFnXgzw. Accessed 8.4.2016, 2016.

4 Kern M. The Limits of Magic. 2003; Available at: http://www.victorianweb.org/courses/fiction/65/tolkien/kern14.html. Accessed 2.4.2016, 2016.

5 Epic Games. GPUSprites Type Data. 2015; Available at: https://docs.unrealengine.com/latest/INT/Engine/Rendering/ParticleSystems/Reference/TypeData/GPUSprites/index.html. Accessed April 17th, 2016.

6 Epic Games. Vector Fields. 2015; Available at: https://docs.unrealengine.com/latest/INT/Engine/Rendering/ParticleSystems/VectorFields/index.html. Accessed 17.4, 2016.

7 Epic Games. Vertex Animation Tool. 2015; Available at: https://docs.unrealengine.com/latest/INT/Engine/Animation/Tools/VertexAnimationTool/index.html. Accessed April 18th, 2016.

8 Epic Games. Creating a Beam Emitter. 2014; Available at: https://www.youtube.com/watch?v=ywd3lFOuMV8. Accessed April 18th, 2016.

9 Kajak Games. Ancestory Launch Trailer. 2015; Available at: https://www.youtube.com/watch?v=QHW4r6OSqjA. Accessed April 18th, 2016.

10 Epic Games. Materials. 2015; Available at: https://docs.unrealengine.com/latest/INT/Engine/Rendering/Materials/index.html. Accessed April 18th, 2016.

11 Epic Games. Essential Material Concepts. 2015; Available at: https://docs.unrealengine.com/latest/INT/Engine/Rendering/Materials/IntroductionToMaterials/index.html. Accessed April 18th, 2016.

12 Ice block visual effect in World of Warcraft. 2012; Available at: http://wow.zamimg.com/uploads/screenshots/normal/319376-ice-block.jpg. Accessed 5.8.2016, 2016.

13 Epic Games. Using Fresnel in your Materials. 2015; Available at: https://docs.unrealengine.com/latest/INT/Engine/Rendering/Materials/HowTo/Fresnel/index.html. Accessed April 19th, 2016.

14 Love J. Technical Artist Bootcamp: The VFX of Diablo. 2013; Available at: http://www.gdcvault.com/play/1017660/Technical-Artist-Bootcamp-The-VFX. Accessed 8.4.2016, 2016.

15 Blizzard Entertainment. Li-Ming Spotlight - Heroes of the Storm. 2016; Available at: https://www.youtube.com/watch?v=6SqmKqFUwnM. Accessed 1.3.2016, 2016.