Recreating Zarya's Particle Cannon

Thanut (Art) Parkeenvincha
Terence So
Alfred Lam

Figure 1: A demonstration of our right-click projectile particle.

ABSTRACT

The aim of our project was to recreate the functionality and visuals of Zarya's Particle Cannon from the game Overwatch (2016). The project would be done in the Unreal Engine, and involve a seamless combination of programming, visual effects, and custom materials and 3D meshes.

We initially set out to recreate all three functions of the Particle Cannon: the Primary Fire (a beam attack), the Secondary Fire (a bomb-like projectile), and the Ultimate Ability (which spawns a black hole that pulls objects into its center).

Unpublished working draft. Not for distribution.

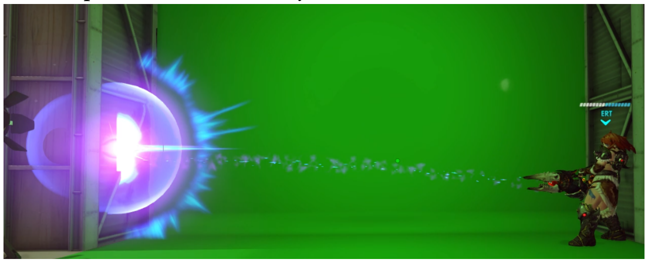
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
CMPM 164, 2019
© 2019 Association for Computing Machinery.
https://doi.org/N/A

ACM Reference Format:
Thanut (Art) Parkeenvincha, Terence So, and Alfred Lam. 2019. Recreating Zarya's Particle Cannon. In ,. ACM, New York, NY, USA, 4 pages. https://doi.org/N/A

1 INITIAL RESEARCH

Because our goal was to recreate the Particle Cannon to the best of our ability, our main source of research involved observing and deconstructing how the Particle Cannon works in-game.

Using our copies of the game, we created a private server in which we could collect reference footage of the Particle Cannon. We recorded footage of the Primary Fire, Secondary Fire, and Ultimate Ability from various angles which could be played back at either slower speeds or on a frame-by-frame basis.



Using this footage, our team was able to visually deconstruct the individual components that made up the Particle Cannon's look and feel. From here, we assigned the individual components amongst our team, and got to work on rebuilding these components in the Unreal Engine.

2 IMPLEMENTATION

After much trial and error with sprite sheets, particle trails, and flip book animations, we finally ended up implementing the left-click laser ability using Niagara's Beam particle. To do this, we first created an emissive material to replicate the effect of the laser's light producing glow. Next, we used this material as the "beam" in the Niagara emitter. The emitter itself consists of a single spawned beam, with a scaled up width and length to imitate the long ranging laser. In the Particle Update section, we then use a Uniform Range to change the beam width over time, resulting in a wavy beam that looks like it's fluctuating over time. We then create a second, larger width, purple colored emitter in the same manner, and stack it on top as the "outer" glow of the laser. Finally, we create a third trails emitter, which uses Niagara's ribbon renderer to send small trails of light outwards along the laser beam. This last part gives the laser a more lifelike feeling, especially when viewed from the side.

On the functional side of the laser, we had to attach the laser to the player, and have it fire whenever the left mouse button was held down, along with loop a laser sound while firing. This was accomplished in Blueprints by spawning a Particle System on click with the "SpawnSystemAttached" function, which automatically placed it where the player location was as well. The looping sound was done by adding an AudioComponent to the player character blueprint, and executing the "Play" and "Stop" nodes whenever the mouse is pressed or released.

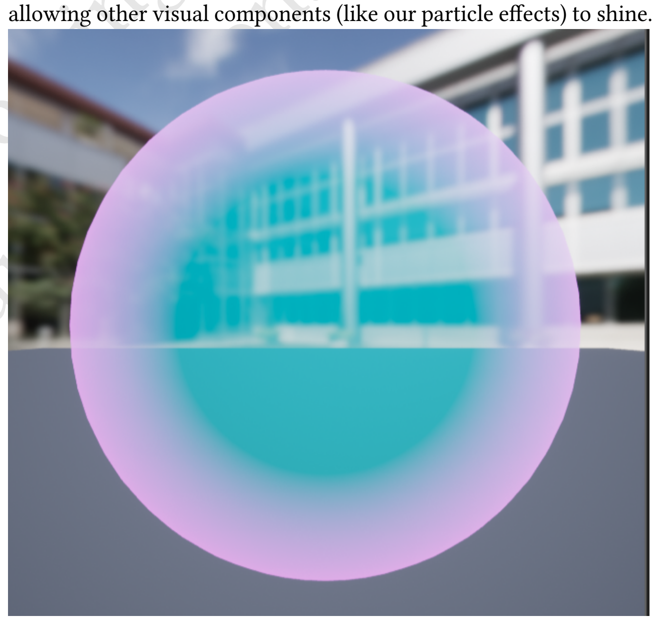
Some technicalities we were not able to finish in time were making the laser stop or create a force on hitting an object or fine tuning the beams to make them more realistic. For the former, we may have needed to ray cast or otherwise find the closest object in a straight line, in order to tell the Niagara system where to stop with a passed-in variable - otherwise, there is no way for the particle system to know how long to make the beam, since it has no connection to the scene as a whole. For the latter, we could have adjusted the materials and colors to make the effect seem more cohesive, mostly through trial and error.

2.1 Materials

Custom materials were made with Unreal Engine's Material Editor; most of the work in this department involved learning how to use the different nodes available in tandem with different PBR-based channels.

Learning how to work with the Material Editor was relatively painless after going through a couple online resources and tutorials (plus a cursory background with Unity's Shader Graph). However, this part of the process still proved time-consuming as a fair amount of fine-tuning was required to achieve the specific visual target that we were aiming for.

After overviewing our reference footage, we found that the look of the projectile itself and the explosion spawned from the projectile were both made up of an "energy orb" effect, which featured varied color and emission values depending on how close a point was to the center of the sphere (in viewport space). By using the Material Editor's Fresnel node, we could differentiate how the edges of objects were rendered compared to their "interiors", since the surface normals of a sphere always grow more perpendicular to the viewing angle as you get closer to the edges in viewport space. In addition, we learned that we could multiply the effects of our Fresnel node with texture-sampled Normal maps to get a more non-uniform look. Finally, swapping the Blend Mode of our materials to "Translucent" allowed us to emphasize the "energy orb" effect while allowing other visual components (like our particle effects) to shine.



In the game, the Particle Cannon also features a muzzle flash upon firing. The muzzle flash contains a subtle distortion effect which we also chose to implement via custom materials. By creating a material with 0% Opacity and toying with the Refraction

Unpublished working draft. Not for distribution.

channel, we could bend how the portion of the scene behind a material was rendered without drawing an explicit, noticeable mesh.

The amount that the background scene is distorted by our material varies depending on how close the rendered portion is to the edge of the mesh; once again, this was achieved with Fresnel math. Non-uniform levels of distortion makes the underlying mesh even harder to detect, which granted meshes spawned with this material a "felt-but-not-seen" quality.

Finally, we experimented with different meshes to apply our distortion material to. Because distortion levels are based on the Fresnel effect, we eventually found that a torus object created a more visually interesting look as there were more surface normals running perpendicular to the view angle.

Unpublished working draft. Not for distribution.

2.2 Niagara

Many particle system were created to create the look of the projectile. The smoke trail, omnidirectional burst, explosion, and flare systems just to name a few. The one we will focus on today is the flare particle system.

To create this effect, a particle sprite is needed. We tried desperately to scout out a sprite that was similar to the shape we desired. However, we were unable to find it. We then decided it was easier to create it ourselves.

It was time to hop into Niagara. We used the omnidirectional burst template and tweaked it to our desired look. One of the most important things was to make sure the alignment mode was velocity aligned. This makes it so that the billboard's axis is in the direction the particle is headed. This allows us to get a "stretched" looking effect.

Because this was the omnidirectional burst template, it was spawning from a reference sphere. We set the surface only band-thickness to zero to guarantee that the particles spawn only at the surface of the sphere.

We then create a slab with a thin thickness. All the particles that spawn within this slab are to be kept, and the rest of the particles that spawn outside are to be killed. This creates a rim-effect.

The result is a flare effect.

2.3 Scene

Our scene was created using assets from the Advanced Village Pack from Unreal's Marketplace. We utilized the 2 house meshes that were given to create a small town scene, making more varied looking buildings by combining the house meshes in different ways. In addition, we added streetlights, but with our own added point lights, since the actual meshes did not actually emit light in the scene. Finally, we added interact able objects (the boxes and watermelon) by using the physics system and changing the mass and other properties of the objects. Creating the scene was perhaps the simplest part of the project, involving mainly learning editor controls, navigation, and importing and editing assets.

3 LESSONS LEARNED

3.1 Blueprints

Working with Unreal was new to all of us, and learning its Blueprint system, with its thousands of nodes and commands, was definitely a challenge. Due to our unfamiliarity with the system, everything from simply looping a soundtrack to spawning a particle system on click required a lot of looking up tutorials and documentation to accomplish. One of the key things we had trouble with was understanding the difference between Actors and Components in the system - the difference was subtle, but important. In the end, "Actors" were simply a term used to describe anything in the scene, which was made up of Components. This was significant in our use of Blueprints, since a good amount of work was put into making sure things were spawned at the correct location, with the correct parent actors - done using nodes like "GetActorTransform" or "GetActorFrom."

Another thing we learned was how difficult it was to make clear, understandable Blueprints - since nodes are executed in a synchronous, linear order, adding functionality quickly resulted in a mess of nodes and inter-crossing lines in the blueprint that was difficult to understand. Given more time and for future reference, we would definitely use more comment boxes and organize the nodes and lines for better readability.

3.2 Version Control

One of our biggest regrets is the fact that we did not notice we could integrate version control directly inside the editor. Instead, we used the Git the traditional way. At first, we had a lot of trouble adding things in due to the fact that many unnecessary things were being committed. One such example is the Saved and Backups folder. But after sorting through that, version control was easily implemented.

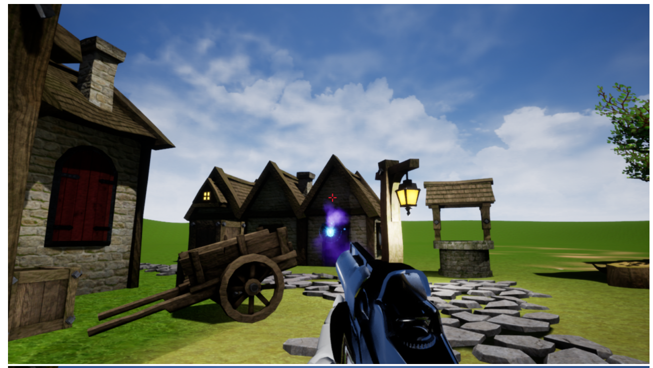
Unpublished working draft.
Not for distribution.

3.3 Niagara

Niagara was the biggest challenge of this project by far. The system is still fresh, with barely any tutorials to be found online. In fact, even the regular particle system that Unreal has barely has any tutorials either! A lot of trial and error was needed to understand how the system worked. After spending time working on this project, we are now more confident to work with it.

4 RESULTS

Our results are shown below:



5 REFERENCES

(1) Actors and Components - Unreal Engine - https://docs.unrealengine.com/en-US/Programming/UnrealArchitecture/Actors/index.html
(2) Niagara Documentation - Unreal Engine - https://docs.unrealengine.com/en-US/Engine/Niagara/index.html

5.1 Assets used

(1) Village Assets - https://www.unrealengine.com/marketplace/en-US/slug/advanced-village-pack
(2) Laser Sound - BMacZero - August 28, 2012 - https://freesound.org/people/BMacZero/sounds/164102/