Exploring Chaos Destruction with Niagara Particle System in Unreal Engine 4

Yanwen Xu
University of California, Santa Cruz
yxu83@ucsc.edu

Alejandro Henriquez
University of California, Santa Cruz
adlandav@ucsc.edu

Figure 1: Spaceship jumping into hyperspace.

ABSTRACT

Accurate and high-performance massive destruction simulations have become increasingly popular today. With the newly released Chaos Destruction Physics System from Epic, developers are capable of achieving cinematic-quality visuals in real-time in scenes with massive-scale levels of destruction.

In this work, we are interested in seeing how the Chaos physics system can be applied to traditional game development workflow in Unreal Engine. We created a space-setting scene that demonstrated the aesthetic and robustness of massive destruction using Chaos. Furthermore, we expanded the scene and integrated it with the new Niagara Particle System and created a visually exciting scenario cinematic clips using Unreal Engine 4.23.

KEYWORDS

Unreal Engine, Chaos Physics, Niagara

CMPM 164, Fall 2019, Santa Cruz

ACM Reference Format:
Yanwen Xu and Alejandro Henriquez. 2019. Exploring Chaos Destruction with Niagara Particle System in Unreal Engine 4. In *Proceedings of CMPM 164.* ACM, New York, NY, USA, 7 pages.

1 INTRODUCTION

For a long time, Unreal Engine 4 (UE4) uses the PhysX 3.3 physics engine [Nvidia 2008] to drive its physical simulation calculations and perform all collision calculations. PhysX provides the ability to calculate accurate collision detection and simulate the physical interaction of objects in the scene. With the newly released Chaos Physics System [Epic Games 2019], developers are capable of achieving cinematic-quality visuals in real-time in scenes with massive-scale levels of destruction. Before Niagara Particle systems [Epic Games 2017], the primary way to create and edit visual effects in UE4 was to use Cascade [Epic Games 2015]. While both Cascade and Niagara can be used to make visual effects inside of Unreal Engine 4, Niagara became a complete replacement for Cascade, because Niagara is much more adaptable and flexible.

The traditional workflow to make any massive destruction scenes relies heavily on the use of external third-party tools or plugins. However, Chaos allows developers to create and edit everything within the UE4 editor. We are interested in seeing how the Chaos physics system can be applied to traditional workflow, and we want to explore how Niagara can be integrated into the Chaos system.

Therefore, we created a space-war scene that demonstrates the potential of the Chaos physics system.

We describe our contribution as follows:

- We acquired and compiled UE 4.23 from the source code on multiple machines.
- We created some planets and space scenarios that exploit the features of Chaos physics in UE 4.23.
- We explored methods to integrate Niagara particle effects with the Chaos workflow, such as triggering particle spawns upon Chaos Break Event.
- We applied different property on different components of a single spaceship mesh.

Moreover, we have released our project to the public at www.github.com.

2 SETUP

Since the Chaos Physics system is still currently under Beta, we need to obtain a source build version of the Unreal Engine in order to use this feature. In this section, we describe what we have done to obtain the required version of UE4. We consider this as a big achievement because compiling UE4 is not trivial, and we spent more than 10 hours in it, with most of the time spent on compiling the engine and the projects.

2.1 Source Code Acquisition

Prerequisites:

- Epic account
- GitHub account
- git or GitHub desktop

According to the official document from Epic, we first need to link our GitHub account to our Epic account; this step is trivial, but it requires an evaluation from Epic. Upon approval, our GitHub account will be added to Epic's Unreal Engine Organization on GitHub and will be granted access to its private repository for Unreal Engine.

Fork the unreal engine repository and clone it to the local environment. Switch the branch to **Release** branch, which was initially Build 4.23.1. This step takes a while, but once it was finished, we will have complete access to UE4's source code on our local environment.

2.2 Build and Compile

Prerequisites:

- Visual Studio 2015/2017/2019

Install Environment: Once the sourced code is obtained locally, and Visual Studio was correctly installed. Be sure to include C++ support as part of the install as indicated in Figure 2, which is disabled by default.

- .NET desktop development
- Desktop development with C++
- Game development with C++

Due to the lack of documentation and tutorials on the internet, we have failed to compile the engine on the first try. After hours of diagnosis, we have determined the cause of failure was missing the a specific version of *.NET framework*.

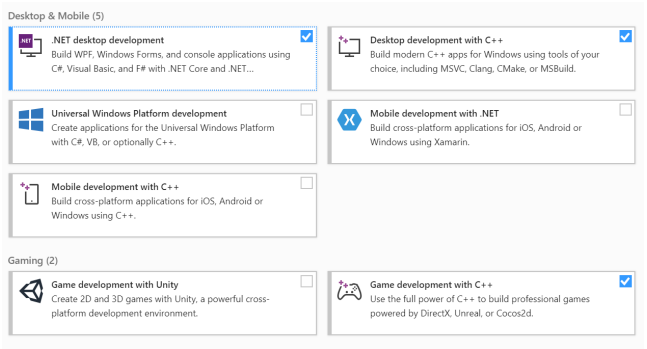


Figure 2: Figure showing the correct configuration for Visual Studio Installer for this project Note that the .NET desktop development is essential to success.

Although *.NET framework* was meant for C# desktop development, a substantial amount of Unreal Engine's component were depending on this framework. Moreover, different component was depending on different version of the *.NET framework*: installing only the latest version is not sufficient.

Build: By default, Chaos features are turned off. To enable the use of Chaos. We first add the statement *bCompileChaos = true;* into the **UE4Editor.Target.cs** file under the **Engine/Source** folder.

In the 4.23 root folder, run the **Setup** batch file, then upon completion, run the **GenerateProjectFiles** batch file. This step takes the most of the time. The average total time on our desktop machines was approximately $4 - 5$ hours. These two batch file will download additional files to the machines, and after the process, another $50GB$ of storage is used. Do not try this at home. We did this on a School Lab Computer.

Compile: In order to make the compilation process easier, we have installed additional Visual Studio Extension *UnrealVS* provided by Epic. *UnrealVS* provide various helper tools, such as the Batch Builder (figure 3), to make the compile task simpler to use.

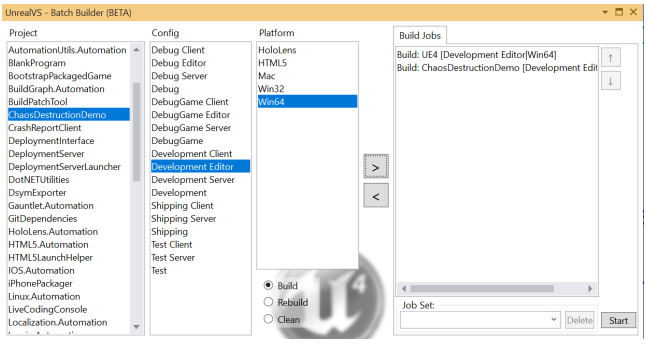


Figure 3: The correct Build Jobs used in this project. We first need to build the Development Editor of UE4 on Win64. Then build the Development Editor of ChaosDestruction-Demo on Win64.

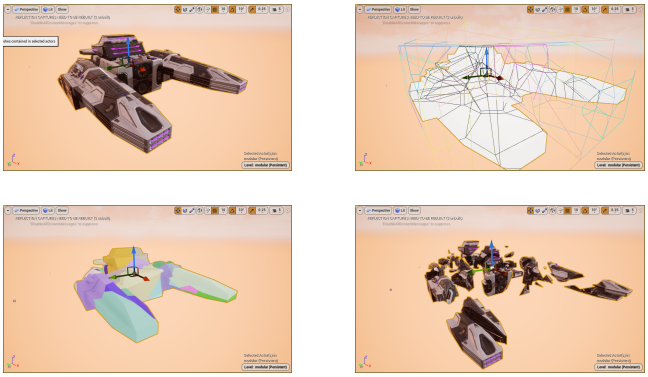


Figure 4: Example of a mesh being transformed into a geometry collection.

Following figure 3, we can start compile binary executable of Unreal Engine as well as the Chaos Demo project. This process takes another $2-3$ hours, as a result, additional $60GB$ of Binary and Intermediate files was generated.

2.3 Create project with Source Build

At this stage, we have successfully compiled Unreal Engine 4.23.1 on our own environment. We can now start the Chaos Demo Project using this source build version of UE4, simply by double clicking the *.uproject* files.

3 CHAOS PHYSICS

In this section, we are going to give an overview of the Chaos Physics System. However, we will not go over every detail of this system. We will only highlight the parts that are most used in our project.

3.1 New Features

- Chaos Editor
- Chaos Solver
- Chaos Niagara
- Planar Cut
- Editable Mesh
- Geometry
- Geometry Cache
- Field System

3.2 Geometry Collections

Geometry collections is a new type of asset that allows the engine to break a mesh apart using the fracture editor. The process of creating geometry collections is very simple. With a couple of clicks, you can turn any mesh or groups of meshes into geometry collections.

3.3 Fracturing and Clustering

In the fracture editor, click the generate geometry collection button under the select tab, adjust the settings until you are satisfied with the results, and then select fracture to finish creating your geometry collection. This process can be used in single or multiple meshes, including the ones inside blueprints.

Clustering is the process on which several meshes are fracture together, or when a smaller piece of an already existing geometry collection is fractured. A geometry collection can have several clusters (Figure 6), and each cluster can have a different strain force required for it to break apart.

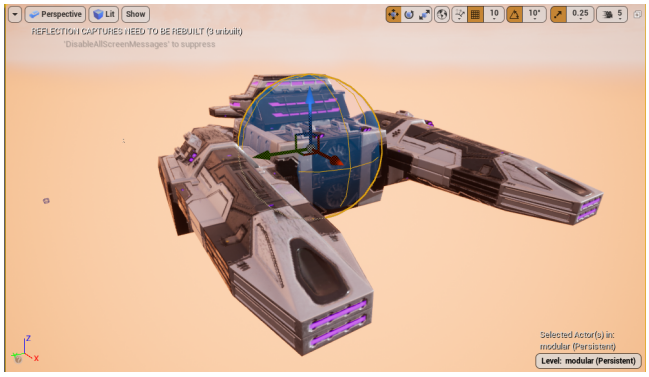


Figure 5: Figure shows a visual representation of the volume occupied by a force field.

3.4 Anchor Fields

Anchor fields are a special type of field because their logic works differently than most of the others. Its purpose is to hold portions of a geometry collection in place while the rest breaks. They must be established in the construction state so that a connection graph is made before the simulation runs. Therefore, the anchor fields must be initialized in the "initialization fields" tab of the geometry collection on which it will be used.

Anchored geometry will remain established until a strained is placed upon it. However, during our tests, we also found out that if the geometry collection is too big, you will need either a bigger anchor field or multiple anchor fields, to hold the structure in place.

3.5 Fields

Fields are a way to directly affect the physics simulation in a given volume, and are used to produce different behaviors or brake effects.

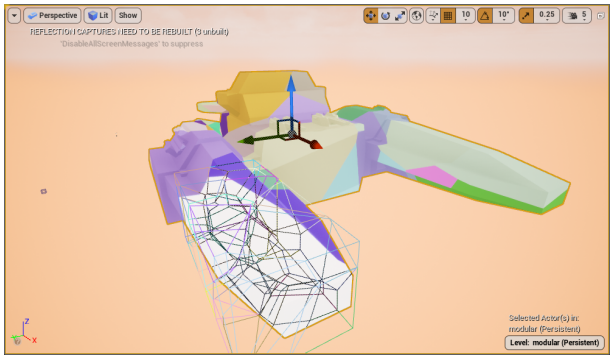
3.6 Field Types

- Cluster Strain
- Force
- Angular Torque
- Noise and Culling
- Sleep and Disable
- Internal Strain

3.7 Used Fields

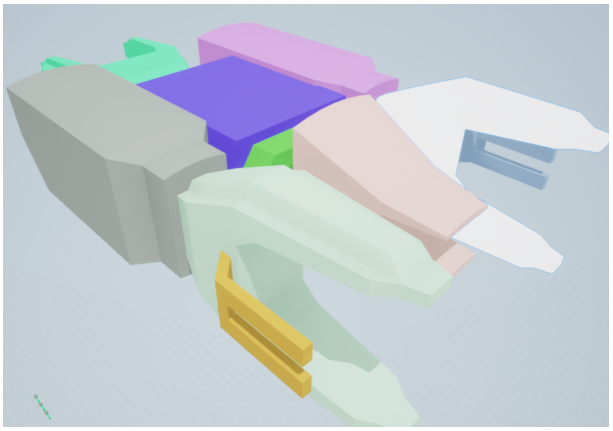
Although there are many types of force fields, we decided to make use of a simple force field (figure 5). These force fields apply a specified amount of force and strain to geometry collections within a given volume. Its basic function is first to apply enough strain to

(a)

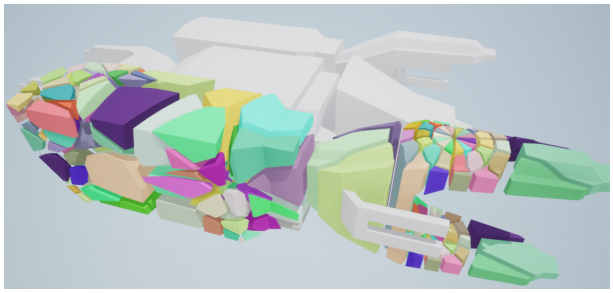


(b)

Figure 6: (a) Normal view of breaking a smaller piece of a geometry collection. (b) Clustering view of a sub-level fracturing. Note only the right wing of the spaceship is fractured. We demonstrate that we can hierarchically fracture geometry collection in Chaos.



(a)



(b)

Figure 7: (a) Decomposed spaceship mashes. (b) Example of showing different fracturing properties being applied to different parts of the ship. Note this is different from Figure 6, we have manually partitioned the original static mesh into pieces. This approach gives artist more control to produce desired visual effects.

break a geometry collection, and then apply a force with which it will push away the broken pieces of the structure. Moreover, we also made use of anchor fields in order to keep our meshes in place.

4 NIAGARA PARTICLE SYSTEM

Unreal Engine 4's Niagara visual effects system are used for creating and previewing particle effects in real time.

4.1 Overview

Niagara particle systems have been broken up into the following two parts:

Niagara Emitter This is a single visual effect that will be combined with other Niagara Emitters to make a Niagara System. For example, in making explosion effects for our project, we have multiple emitters represents fire, smokes, and embers particles.

Niagara System This contains everything needed to make up a single effect. Continuing with our example, this would be the completed explosion effect that contains the Niagara Emitters for the smoke, fire, distortion, and embers.

4.2 Chaos Integration

However, while Niagara sounds promising, the way that Niagara works in Chaos Projects is slightly different.

Chaos Data Interface: The most important thing to note is the Chaos Data, this is an abstract interface class. A Niagara system must implements the Chaos Data Interface, and will be active for all rigid bodies driven by the world Chaos Solver. Inside the Niagara Effects system, events listed below are listened for through the Chaos Data Interface.

Events: There are three types of events in Chaos, listed below,

- Break Event

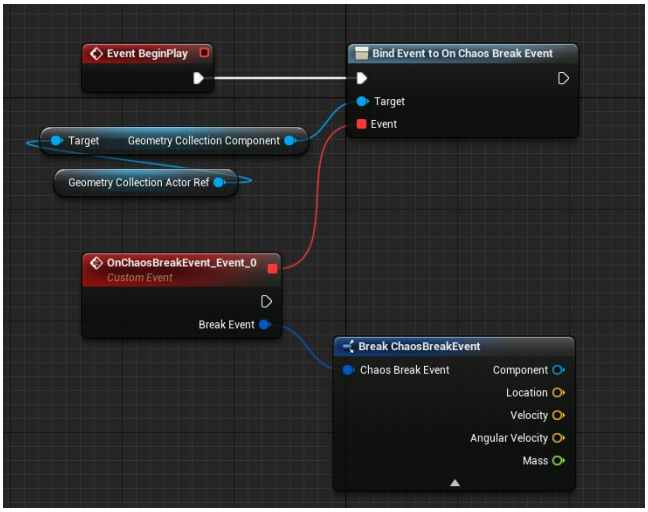


Figure 8: Blueprints in our project, showing how we can bind the Chaos break event of a Geometry Collection Actor to an custom Event.

- Collision Event
- Trail Event

Geometry Collections can send collision and break events that can be used to trigger gameplay. In our project, we are sending events to trigger the Niagara Particles to spawn the particles (Figure 8). For the Geometry Collection to send these events, the Notify Collisions (for collision events) or Notify Breaks (for break events) must be enabled on the Geometry Collection Actor.

As shown in Figure 8, we can bind any types of Chaos Events to a CustomEvent (named *OnChaosBreakEvent_Event_0*). Thus, when Chaos event is triggered, we can receive the information such as the following:

- Component: Component
- Location: FVector
- Velocity: FVector
- Angular Velocity: FVector
- Mass: float

With this information, we can make things like spawn a Niagara System Actor at *Location*, giving it some initial speed properties *Velocity* and more.

Chaos Solvers: It is vital to understand where the Chaos events are stored. By default, Chaos automatically creates a *Chaos World Solver* actor to manage Chaos related computation. It is an actor that will be placed on the Map. Chaos solvers within the Chaos Destruction system can store a list of the collision, breaking, and trailing events as the simulation runs. Inside the Niagara Effects system, those events are listened for through the Chaos Data Interface.

5 RESULTS

We describe our setup of our scene and present the results in this section.

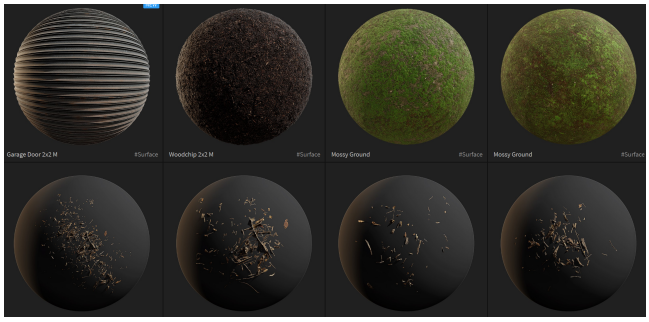


Figure 9: Megascans is a massive online scan library of high-resolution, consistent PBR calibrated surface, vegetation and 3D scans, also including desktop applications for managing, mixing and exporting your downloaded scan data.

	sm1	sm2	sm3	sm4	sm5	sm6
type	U	R	U	U	U	P
radius	-	50.0	-	-	-	-
min	20	-	15	10	5	-
max	20	-	25	30	35	-
cuts	-	-	-	-	-	3

Table 1: Settings of our composited spaceship. This setting turned out to have the best visual effect.

5.1 Our scene

We first created a Skybox Blueprint actor, using space textures from online assets store. The Skybox actor is internally represented as a sphere static mesh.

We created an over planet landscape with the Unreal terrain tools. We initially wanted to use the newly released terrain tools from 4.24. However, Chaos system is still not integrated into the Engine yet, and we were not able to compile it from source again. Thus we simply decided to use the traditional terrain editor. We imported textures from Quixel Megascans [Quixel 2015], which recently announced to join force with Epic.

We decomposed the static mesh asset of a spaceship obtained from assets store into multiple parts. The purpose of breaking the one single static mesh is to manually set different properties to different pieces of the spaceship. This step allows the Chaos physics engine to break the spaceship in a more realistic way; as a result, our experimental parameters presents a good final visual result. Table 1 shows a detailed setting of our decomposed ship.

Our demo space scene can run on Win64 Desktop with a frame rate of 60 fps, see Figure 10. Refers to Figure 11 and Figure 12 for updates.

6 CONCLUSIONS AND FUTURE WORK

In summary, we have successfully compiled UE 4.23 from the source code on multiple machines. We have successfully created an over planet space scenarios that exploit the features of Chaos physics. Most significantly, we proposed multiple methods to integrate Niagara particle effects with the Chaos workflow. Equally significant,

(a) (b)

Figure 10: (a) Chaos destruction event happening at timestamp t_0, the spaceship is exploding. (b) After the first explosion happens, parts of the spaceship crush on the ground at time t_1. This has triggered a Collision event, and illuminating particles spawned.

Figure 11: At timestamp t_0, a break even is triggered on a modified(decomposited) space mode. see Figure 2 12

we have successfully set up the different levels (materials) on the spaceship as promised.

After conducting this research and tried out Chaos, we see that Epic has ambitious plans to replace its original PhysX 3.3 physics engine with Chaos gradually. At the same time, Epic is also considering to substitute Cascade with Niagara completely.

Due to inexperience in UE4 and the shortage of time, we were incapable of creating the initially proposed plan: build an interactive star-war like an arcade shooting game. For future works, we are interested in seeing the following:

- Manipulating Gravity.
- Caching in Chaos.
- Interactive arcade shooting game with many spaceships.

Finally, the latest stable release of Unreal Engine 4.24 came out on December 9, 2019; a lot of new features added, such as the new Landscape tool. According to the development blog, and commit history, a few changes to the Niagara Particle system has been made. Although Chaos Physics System is still not integrated into the release version yet, we believe that once it completes its Beta test and is added to the Engine, Unreal Engine 4 will become the most powerful tool to create massive destruction.

ACKNOWLEDGMENTS

The authors would like to thank Montana Fowler for providing excellent lab tutorials on Niagara Particle systems. These tutorials made our project much easier to work with. The authors would also like to thank Professor Angus Forbes for providing valuable feedback on our work during office hours and class presentations.

REFERENCES

Inc Epic Games. 2015. Cascade Particle Editor Reference. https://docs.unrealengine.com/en-US/Engine/Rendering/ParticleSystems/Cascade/index.html

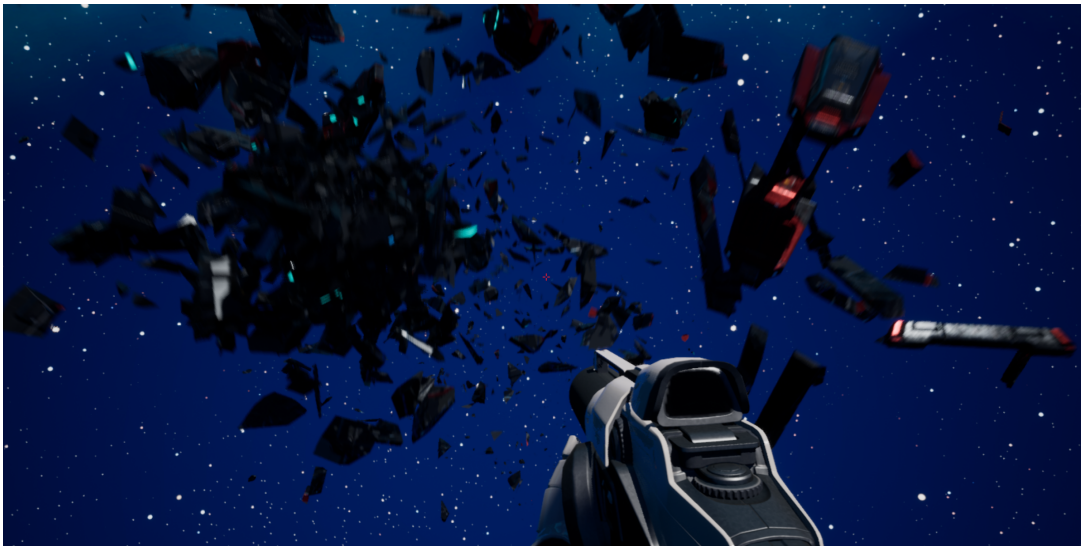


Figure 12: The most important figure, this figure shows the visual result of destructing a spaceship with setting from Table 1. As shown in the figure, part of the spaceship is broken into smaller pieces, where are others are larger and grouped together. This is because we have manually applied different material properties on different parts of the ship.

Inc Epic Games. 2017. Niagara Visual Effects. https://docs.unrealengine.com/en-US/Engine/Niagara/index.html

Inc Epic Games. 2019. Chaos Destruction Overview. https://docs.unrealengine.com/en-US/Engine/Chaos/ChaosDestruction/ChaosDestructionOverview/index.html

Corp Nvidia. 2008. NVIDIA PhysX SDK. https://github.com/NVIDIAGameWorks/PhysX

Quixel. 2015. Quixel Megascans. https://quixel.com/