# Comparative Study of Challenges Creating FX for Real-time and Off-Line Workflows

A Thesis Submitted to the Faculty of the Visual Effects Department in Partial Fulfillment of the Requirements for the Degree of Master of Fine Arts in Visual Effects

At

The Savannah College of Art and Design

Luke D. Vuilliomenet

Savannah, GA



*Deborah Fowler,* Committee Chair

*Stuart Robertson,* Committee Member

*Jennifer McSpadden*, Committee Member

## Dedication

I want to dedicate this work to Ms. Rita Black.

Thank you for always believing in me and everything I do.

May your kindness be an inspiration to everyone to walk to the beat of their own drum.

**Acknowledgments**

To my mother and father for their unwavering love and support.

To Professor Deborah Fowler, for inspiring me to learn Houdini and FX.

To Professor Stuart Robertson, Professor Bridget Gaynor, and Professor Joe Pasquale, for constantly pushing me to strive for my best and never settle for less.

To Professor Charles Shami and Professor Aram Cookson for offering help and advice as I explored this entirely new technology for me.

Especially to Jennifer McSpadden, who offered guidance, support, and encouragement. I cannot express my thanks enough.

**Table of Contents**

## List of Figures

## Abstract

Comparative Study of Challenges Creating FX for Real-time and Off-Line Workflows

Luke Vuilliomenet

November 2021

Virtual Production (VP) is a fast-growing multi-production pipeline that draws upon the strengths of Virtual Reality (VR) and Augmented Reality (AR) to create a hybrid workflow environment known as XR. The use of XR Stages, equipment and other VP techniques, including LED wall sets, offers a great deal of control and freedom over background elements when shooting live-action actors. However, productions using virtual production are dependent upon real-time render engines, such as Unreal, and the limitations with their implementation.

A study to illustrate the advantages and disadvantages of creating FX within Side Effects' Houdini for virtual production, including XR productions, is required for Visual Effects artists to design their work effectively for real-time and established production environments, termed off-line. One example of an FX that could be explored is a cluster of rocks impacting the lunar surface (moon), aiming to be a hero element within a Production. The designed elements will be created within Houdini, including Rigid Body Dynamic (RBD) destruction assets that lead to incorporating Particle Operators (POP) for dust and Pyro elements for smoke. All assets are then imported into the Unreal Engine and set up to run in a VP/XR environment. The study will highlight the differences between the two workflows and what challenges an FX artist may face when working on a virtual production.

Understanding these challenges and the methodology of Houdini FX for XR and virtual production will greatly benefit Houdini users and future Productions hoping to utilize this technology.

Key Words/Phrases:
Houdini, Unreal, FX, Simulation, Virtual Production, Real-time, Off-line.

## 1. Introduction

When working in the Film and Television industries, creating content has been, to date, a linear (off-line) workflow. As its name suggests, a linear workflow moves in a particular direction. Each step in the production has a clearly defined role dependent upon what comes before it in the pipeline. An oversimplified example is the fact that pre-visualization occurs before principal photography, which occurs before computer-generated (CG) model creation, followed by layout and animation. The limitation of a linear pipeline is the dependencies it creates. Different production stages are often reliant upon the completion of previous steps. Animation cannot occur without a model to animate or camera tracking without a plate to track. If a sudden change is needed or a mistake is found, an entire team's work can be rendered unusable.

In comparison, the game industry runs on an entirely different concept, which is a non-linear, real-time workflow. Instead of having one element created at a time, multiple production stages co-occur. FX and compositing are designed to run simultaneously in real-time, rendering them less dependent on one another. The lighter amount of dependency between production stages allows a more rapid back and forth during development. However, a faster iteration cycle comes at the cost of hardware and software limitations. A real-time production's final visual look/fidelity is ultimately based on consumers' hardware, leading to varying experiences.

Each of these pipelines has a set of problems when creating FX in Production. The off-line (linear) workflow is entirely camera-oriented; if the renderable camera does not see the element, it is an unnecessary addition. Thus each shot in an off-line workflow has the maximum amount of detail afforded for the render frame. However, the high visual fidelity comes at a lengthy render time and compositing cost. On the other hand, the camera in the

real-time pipeline is not constrained and has more freedom to move within the scene. The newfound camera freedom in real-time comes at a cost that FX can no longer be camera-oriented and instead must be optimized to be viewed from most angles while running at a minimum of 30 frames per second (fps). A new methodology is given form, virtual production (VP), by combining the real-time rendering strength of a game engine and the FX capability of an off-line Digital Content Creator (DCC).

### 1.1 Houdini and Unreal

SideFX Houdini has been chosen as the primary off-line DCC FX creation tool for this examination, known for its robust FX suites and procedural production capability. Houdini utilizes a dynamic operator system (DOPs) that offers FX types ranging from dense particle simulations to vast oceans (fluids).[1] In recent years SideFX has also begun investing heavily in the research and application of Houdini FX within game engines. In particular, SideFX has partnered with Epic Games' Unreal Engine, designing an extensive toolset, SideFX Labs, to export assets from Houdini and plug-ins for the game to import Houdini simulations.

Unreal Engine offers an in-depth framework to build a wide range of real-time experiences. Noticeably, Unreal has been adopted into the film and episodic (Television) industries for its support of virtual production and non-game-based projects.

### 1.2 Goal: Illustration of Advantages and Disadvantages

Real-time production capabilities have given rise to new design challenges when

---

[1] Craig Zerouni, "Particles," in *The VES Handbook of Visual Effects Industry Standard VFX Practices and Procedures* 3rd ed., eds. Jeffrey A. Okun and Susan Zwerman (New York: Routledge, 2021), 571.

producing Visual Effects. Using current simulation FX techniques and exploring multiple FX, a comparative examination will illustrate the new real-time production workflow's advantages and disadvantages compared to the traditional pipeline, referred to as off-line from this point. The examination will demonstrate that real-time offers substantial benefits to projects using virtual production. However, this comes at the cost of FX still dependent on the off-line workflow.

The FX case study will focus on an impact/explosion on the lunar (moon) surface dealing with particles, volumes, destruction in the absence of an atmosphere, and low gravity to document the technical and creative solution/challenges by comparing the two workflows of real-time and off-line FX. The FX will be designed for use within an LED Volume virtual production project, thus not requiring an entirely open-world experience.

## 2. How is Virtual Production Defined in Industry?

A term that has recently exploded in popularity within the Film and Television industries is Virtual Production (VP). However, what exactly is virtual production? There is no clear-cut answer to this question, as the technology is still evolving and varies for each discipline and studio, depending on intent and implementation. How a previsualization studio defines and utilizes virtual production varies significantly compared to a post-production or production studio.

Epic Games Virtual Production Supervisor Kevin Cushing speaks on the topic of virtual production and the impact it has had on the industry in a personal interview. Kevin has had over twenty years of experience within the visual effects industry, working on titles such as *Avatar (2009)*, *Avengers Infinity War (2018)*, *Avengers Endgame (2019)*, *The Mandalorian*

*(2019)*, and several others. When discussing how to define and approach virtual production, Kevin comments about the wide range virtual production covers and the confusion the term has caused:

> "It is a pretty broad term, so that is probably the issue. It covers everything from motion capture, to previs, to postvis, to visual effects. It is hard to define. It is more defining the space. I'm currently working at Epic Games as part of the LED wall technology that was used for *The Mandalorian*, which is an aspect of virtual production. You could say all the people prepping the artwork and the art director looking at assets on the wall are using virtual production. Instead of a lot of post-visual effects, you have to preplan and get a lot done upfront. It has moved a lot of the visual effect workspace to the front end rather than the back end."[2]

Virtual production is one of the technologies that has created this paradigm shift within the production pipeline. In order to help define virtual production as a possible solution, a review of its development history is needed to clarify the problems that it aims to solve.

### 2.1 Virtual Production and the Problems it Hopes to Address

The traditional production pipeline has a multitude of problems and limitations that virtual production hopes to help alleviate, if not solve. *The Virtual Production Field Guide V1* by Noah Kadner and presented by Epic Games discusses the problems with the traditional production pipeline.

The most common issue artists found was the feeling of disconnection between

---

[2] Kevin Cushing, Interview with Author, Zoom, October 20, 2021.

departments. Iteration is a critical process in any form of production, in which a project is refined toward the final product. However, the iteration process in a traditional production workflow can take weeks or longer to see results. Kadner aptly described the process as "learning how to play an instrument that you can't hear."[3] In place of universal understanding and a common goal, uncertainty is formed. Directors and actors alike had to make informed guesses on what and where a CG/green-screen element would exist in the shot when filming. This leads to a more cautious approach being adopted to ensure an element can be integrated correctly due to cost or the impossibility of making post-production changes.[4]

Another issue presented within the traditional pipeline is the complexity of the projects. There are many moving parts within a film's production that all have to operate on a tight schedule to be completed on time. Complex dynamic simulations that utilize millions of particles and geometry fractures must be run through a complicated compositing network to begin looking at the final result. Previs and Techvis must happen along with primary photography, principle CG modeling, animation, VFX, and much more.[5]

---

[3] Noah Kadner, *The Virtual Production Field Guide*, Ver. 1, (Epic Games, 2019), https://cdn2.unrealengine.com/vp-field-guide-v1-3-01-f0bce45b6319.pdf, 8.
[4] Kadner, 8.
[5] Kadner, 7.

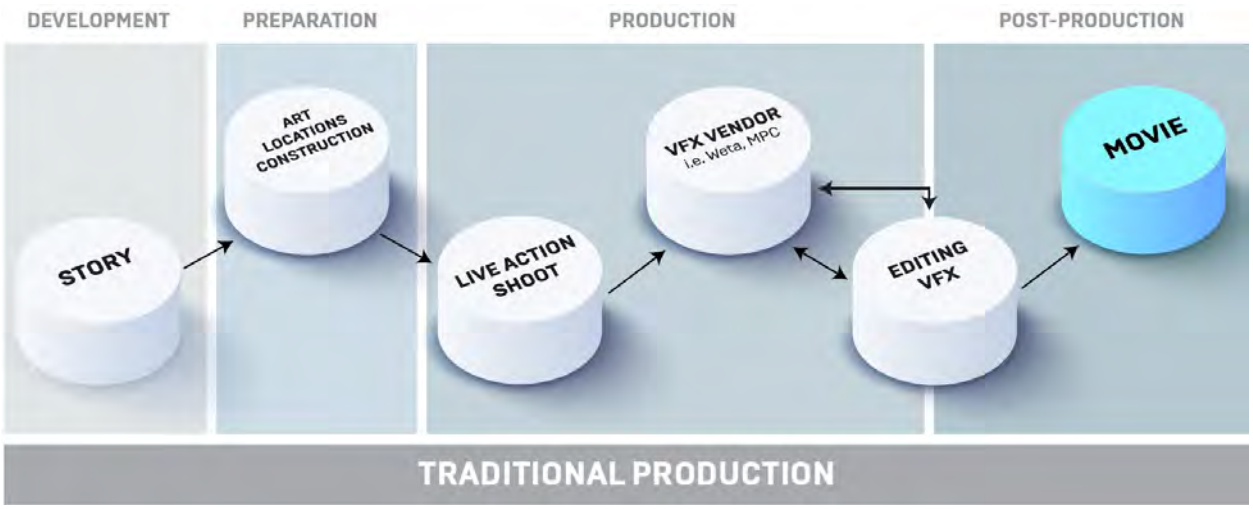



*Fig. 2.1 Noah Kadner and Epic Games, Traditional Production Pipeline Graphic 2019 © 2019 Epic Games (Kadner, Image, The Virtual Production Field Guide, https://cdn2.unrealengine.com/vp-field-guide-v1-3-01-f0bce45b6319.pdf)*

The linear structure of the traditional production pipeline leads to many issues. Content is created in each stage of production to be passed on to the next team. The traditional linear workflow is illustrated in Fig. 2.1 from *The Virtual Production Field Guide*. Kadner compares the workings of the traditional production pipeline to that of a "linear affair, one that resembles an assembly line, encompassing development, pre-production, production, and post."[6] With each part of production generating the elements required for its portion of the project, a majority of those elements will become incompatible with later parts of the production.[7]

Virtual production was created to resolve the problems mentioned above within the traditional pipeline to reduce iteration time and cost while removing the uncertainty when working on a green-screen stage.

While speaking with Kevin Cushing, a question arose regarding virtual production and its

---

[6] Kadner, 7.
[7] Kadner, 7.

benefits to offer a project and the visual effects industry. Cushing's thoughts of the benefits offered in the virtual production pipeline are directly correlated to the problems initially within the traditional pipeline:

> "There are several things. The first is in terms of creativity, being able to be there on set rather than on a green screen. On *Avengers*, we shot three-hundred sixty-degree of green screen massive sets where you couldn't see the CG elements in the background. Often we would argue, where is the moon, where is Thanos coming from, now you are looking at boards in different directions to figure out what will happen. When you shoot through a camera using real-time lighting, the DP and art directors can see what is going on and adjust composition. There is much more interactivity by heads of departments and anyone else on the stage."[8]

The development of virtual production took years of testing, application, and retesting across multiple productions and companies to culminate into the toolsets used on modern shows. There are multiple forms of virtual production which can include but are not limited to visualization, performance capture, hybrid green screen live, and a full live LED wall.[9]

[8] Cushing.
[9] Kadner, 14.

## 2.2 The Historical Rise of Virtual Production and Real-Time FX

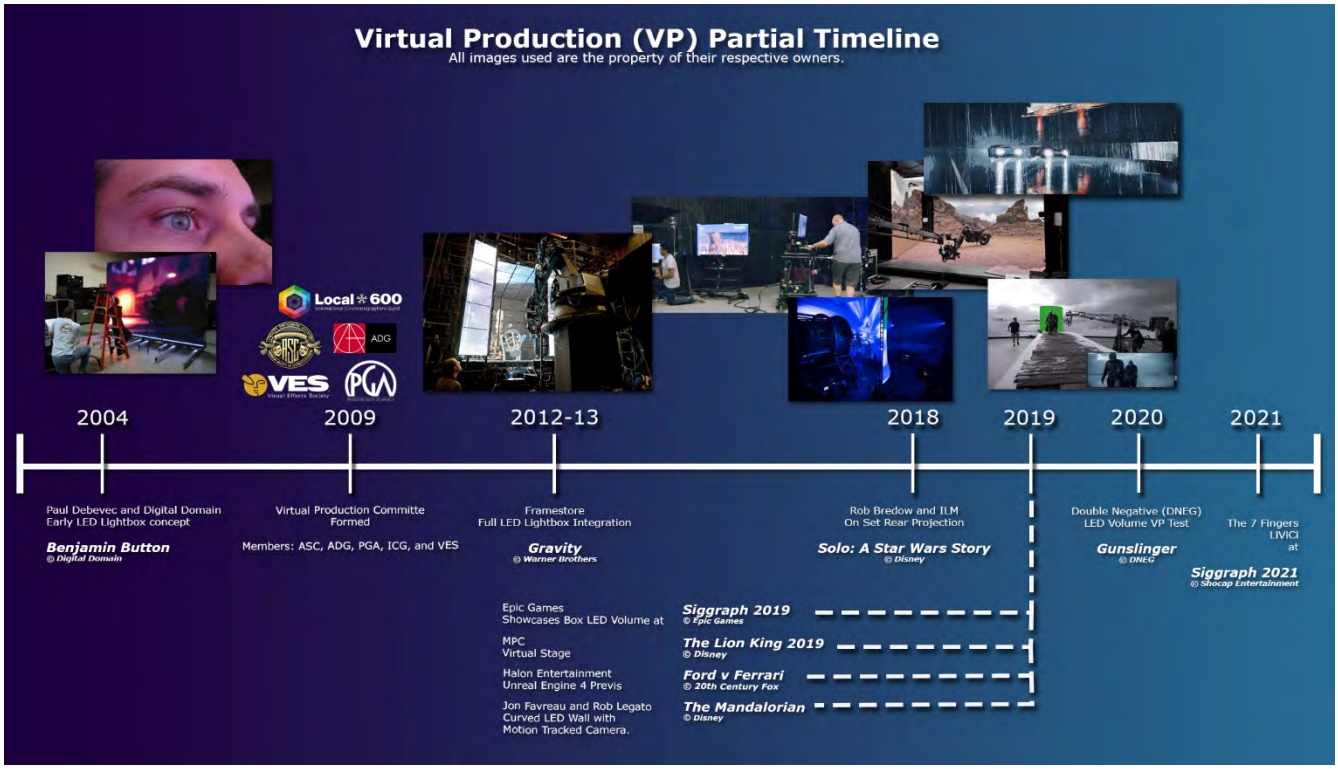



*Fig. 2.2 Virtual Production (VP) Partial Timeline (All images used are the property of their respective owners.)*

Current production workflow problems encourage those within the film and television industries to create new technology to solve or adapt these problems. However, new techniques can be slow to create, especially if the necessary computer hardware has not been invented. Virtual production is a perfect case example of this problem-centered growth. A partial timeline is seen in Fig. 2.2.

An early example of virtual production testing was done by Paul Debevec in an early version of the *Benjamin Button* film. In the film, the plan was to have the character's eyes be a live actor and the face as a digital double (Digi double). However, this required the live actor's eye to have the correct reflections to match the CG environment. The idea was to surround the actor in a wall of LED panels placed upon rotating robotic arms, with the first test occurring in 2004, shown in Fig. 2.3. The test involved having a person sit on a ladder

with a single panel placed in front of them to create the reflections. The test was successful in creating accurate reflections on the actor's eye. However, the director decided to use a full digi-double for the character, and the idea never moved past the initial test phase.[10]

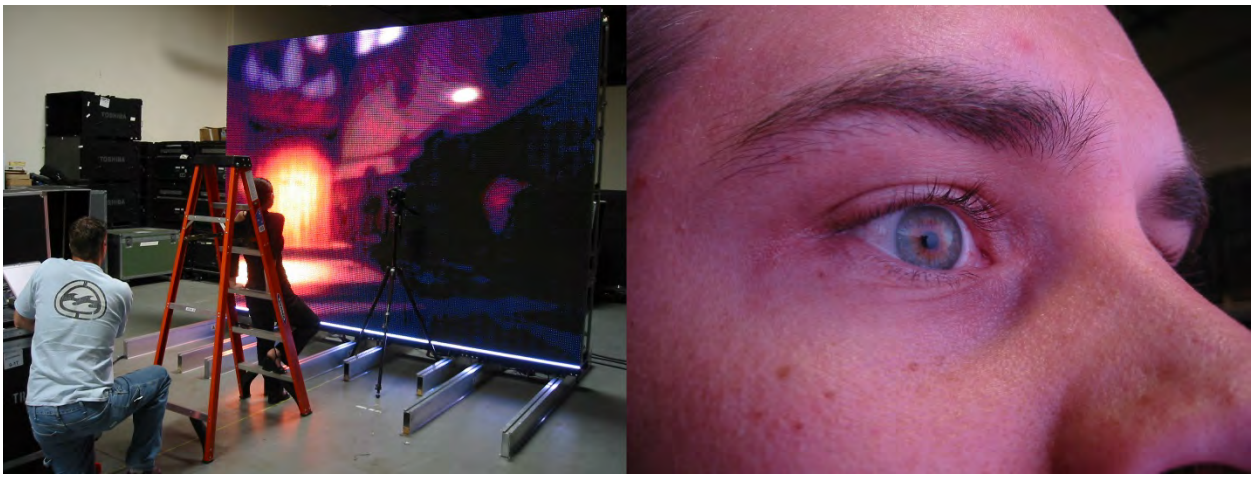


*Fig. 2.3 Digital Domain and Paul Debevec, Early LED Lightbox Concept Test for Benjamin Button 2004 © Digital Domain (Seymour, Image, Art of (LED Wall) Virtual Production Sets, Part Two: 'How you make one,' https://www.fxguide.com/fxfeatured/art-of-led-wall-virtual-production-sets-part-two-how-you-make-one/)*

The Virtual Production Committee was officially formed in 2009, with individuals from the American Society of Cinematographers (ASC), the Art Directors Guild (ADG), the Producers Guild of America (PGA), the International Cinematographers Guild (ICG), and the Visual Effects Society (VES). The committee's purpose was to share information and case studies regarding films and television Production that utilized virtual production and to help define this new technology. The initial test case data includes George Lucas's *Star Wars: Episode I*, Peter Jackson's *Lord of the Rings*, James Cameron's *Avatar,* and many others.[11]

---

[10] Mike Seymour, "Art of (LED Wall) Virtual Production Sets, Part Two: 'How You Make One'," *FX Guide*, March 9, 2020, accessed October 9, 2021, https://www.fxguide.com/fxfeatured/art-of-led-wall-virtual-production-sets-part-two-how-you-make-one/?highlight=LED%20Wall%20Virtual%20Production.

[11] Kadner, 5.

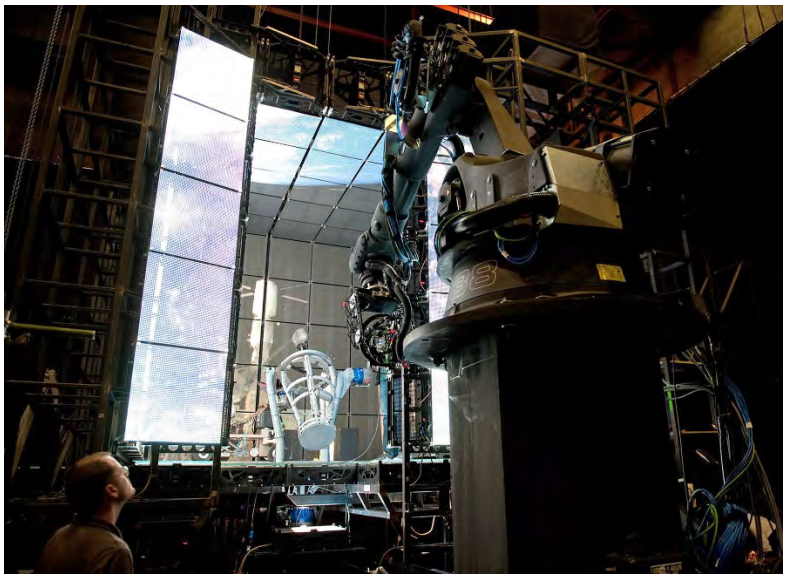


*Fig. 2.4 Framestore and Tim Webber, LED Lightbox Integration for Gravity 2013 © Warner Brothers (Seymour, Image, Art of (LED Wall) Virtual Production Sets, Part Two: 'How you make one,' https://www.fxguide.com/fxfeatured/art-of-led-wall-virtual-production-sets-part-two-how-you-make-one/)*

In 2012-2013, the film *Gravity* took the next step from the 2004 *Benjamin Button* test concept and implemented a full LED lightbox. The lightbox stood 20 feet tall and 10 feet wide, consisting of 196 LED panels, in essence, inward-facing giant televisions, with 4,096 LED lights per panel. An image taken from the LED Lightbox stage of *Gravity* can be seen in Fig. 2.4. The lightbox was an early predecessor to the massive LED walls of modern Production, the panels were coarse and did not match perfectly in alignment, and the colors shifted depending on what angle the LEDs were viewed.[12]

The next large-scale advancement of virtual production was during the production of *The Lion King* remake in 2019, where the team at Technicolor and MPC created an entirely virtual shooting stage for the CG film while giving the director controls similar to that of a live set. The advancement allowed directors and artists to work collaboratively in a virtual

[12] Mike Seymour, "Art of (LED Wall) Virtual Production Sets, Part Two: 'How You Make One'."

environment with results viewable immediately.[13]

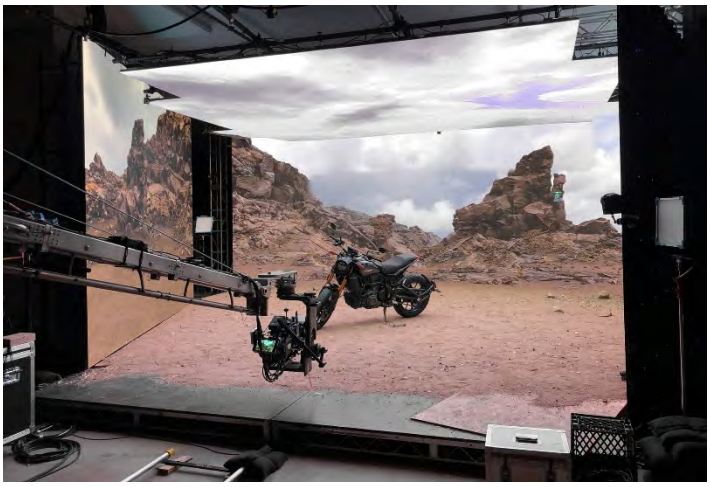


*Fig. 2.5 Epic Games, Siggraph 2019 Epic Games Showcases LED Volume Setup © Epic Games (Seymour, Image, Art of (LED Wall) Virtual Production Sets, Part Two: 'How you make one,' https://www.fxguide.com/fxfeatured/art-of-led-wall-virtual-production-sets-part-two-how-you-make-one/)*

In 2019, Epic Games presented at Siggraph a box-shaped LED Wall volume, Fig. 2.5, in collaboration with Lux Machina, Magnopus, Profile Studios, Quixel, ARRI, and Matt Workman to publicly show the new technology.[14] After Epic's presentation, virtual production had reached a state close to what is available in 2021, and improvements continued to be released covering a wide variety of situations.

As the production of *The Lion King* wrapped up, a new series based in the Star Wars universe was discussed, titled *The Mandalorian*. Jon Favreau, director of *The Lion King* remake, and Rob Legato, the VFX Supervisor, began discussing the possibility of using the technology developed for *The Lion King* on this new Production. However, instead of using an entirely virtual environment, the stage would be real and use a giant LED wall, advancing

---

[13] The Focus, "Virtual Production 101: How Does It Work, How Can It Revolutionise VFX?" *Medium*, October 18, 2019, accessed September 28, 2021, https://medium.com/@thefocus/virtual-production-101-how-does-it-work-how-can-it-revolutionise-vfx-1c7e80ade0f2.

[14] Mike Seymour, "Art of (LED Wall) Virtual Production Sets, Part Two: 'How You Make One'."

upon *Gravity* and *Benjamin Button*'s original tests while modifying the design to be similar to the rear-screen project technique, going back to the 1960s.[15] Partnering with Epic Games, ILM, Magnopus, and others, Favreau and Legato began putting together a test project for the technology, based on the rear-screen project used on *Solo: A Star Wars Story* (Fig. 2.6).

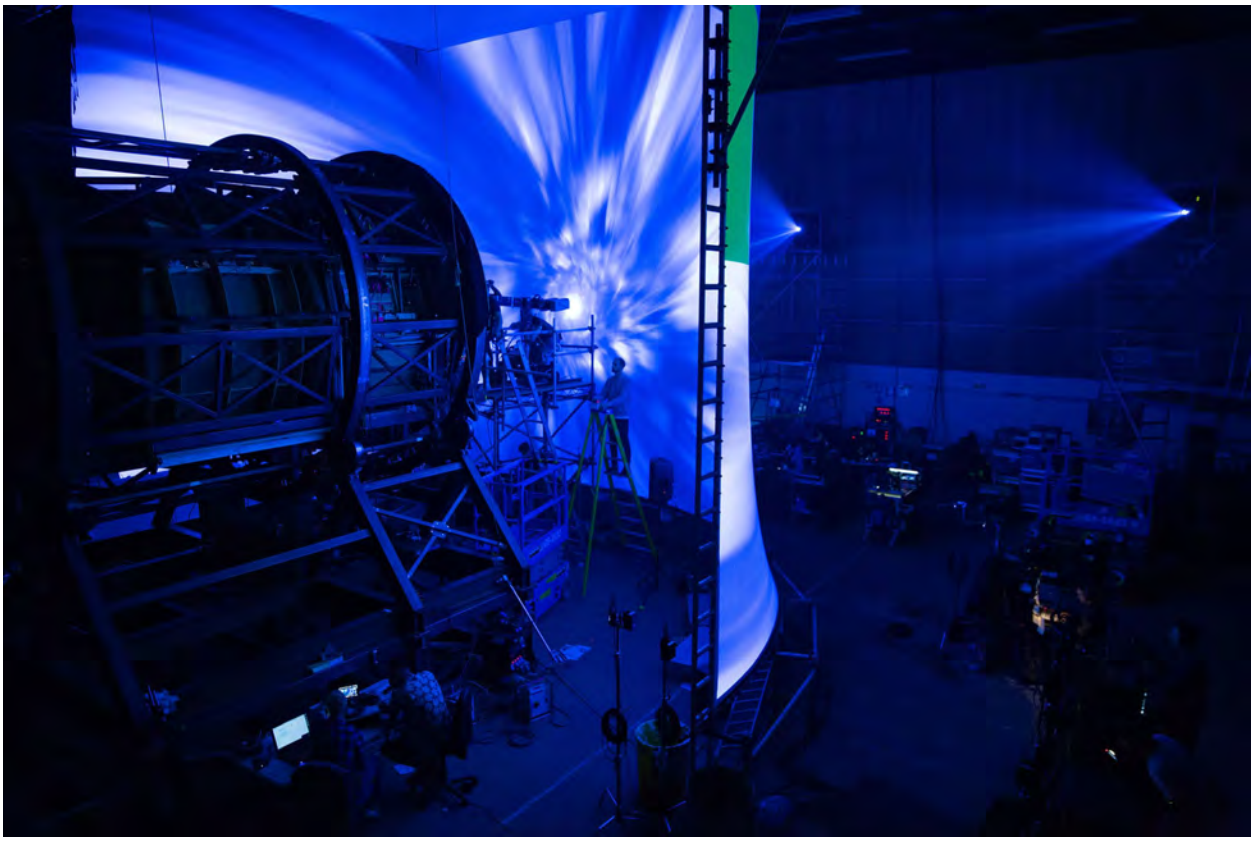


*Fig. 2.6 John Knoll and Rob Bredow, Rear Projection for Solo: A Star War Story © 2018 Lucasfilm Ltd. & Tm, Disney (Seymour, Image, Art of (LED Wall) Virtual Production Sets, Part Two: 'How you make one,' https://www.fxguide.com/fxfeatured/art-of-led-wall-virtual-production-sets-part-two-how-you-make-one/)*

Jon Favreau and Rob Legato's test idea of using a large LED wall was not only possible but could be achieved. The new stage used a large-scale curved LED wall to display dynamic digital sets live while filming. The stage sat at a diameter of 75`with a curved LED wall 20`

[15] Mike Seymour, "Art of (LED Wall) Virtual Production Sets, Part Two: 'How You Make One'."

tall and spanning 270-degrees.[16] The digital environments were rendered and streamed to the LED wall using the Unreal Engine and four synchronized PCs. [17] At the same time, a live camera was tracked within the volume using motion capture and mapped to the view of the virtual scene displayed on the wall. Not only were the digital sets viewable while filming, but thanks to the Unreal Engine, they were also editable, allowing the digital elements to be adjusted in real-time. Necessary changes to the digital environment were directed from within the LED volume via iPad input and could be as specific as per asset. The stage used for *The Mandalorian* can be seen in Fig. 2.7.

*Fig. 2.7 Jon Favreau, Rob Legato, ILM and Disney, Live LED Wall Stage for The Mandalorian Season 1 2019 © 2020 Lucasfilm Ltd. & Tm, Disney (Seymour, Image, Art of (LED Wall) Virtual Production Sets, Part Two: 'How you make one,' https://www.fxguide.com/fxfeatured/art-of-led-wall-virtual-production-sets-part-two-how-you-make-one/)*

Favreau and Legato initially believed that the new technology would only be helpful for the show's more straightforward shots but would still require a VFX crew to be present

---

[16] Mike Seymour, "Art of LED Wall Virtual Production, Part One: 'Lessons from the Mandalorian'," *FX Guide*, March 4, 2020, accessed October 9, 2021, https://www.fxguide.com/fxfeatured/art-of-led-wall-virtual-production-part-one-lessons-from-the-mandalorian.

[17] Jeff Farris, "Forging New Paths for Filmmakers on 'The Mandalorian'," *Epic Games*, February 20, 2020, accessed October 3, 2021, https://www.unrealengine.com/en-US/blog/forging-new-paths-for-filmmakers-on-the-mandalorian.

afterward to add complexity. In an astonishing turn of events, over fifty percent of *The Mandalorian* season one was filmed using the new LED stage and made it into the final edit.[18] The introduction of LED wall technology on *The Mandalorian* sparked a wildfire of interest within all areas of media creation. It invisibly changed the entire way film and television were planned and filmed. When asked about his time spent on the production of *The Mandalorian,* Kevin Cushing presented how working within the LED volume was a learning experience for everyone involved:

> "For me on Mandalorian, I came off Avengers and was part of Profile Studios, supervising camera tracking. I have been doing camera tracking for years, and on Avengers, we were tracking the hero camera so we could have actors in mocap suits and capture characters like Hulk or Thanos. In this case (*The Mandalorian)*, we were focusing on tracking the camera, which is fairly straightforward. However, it must be kept stable to pass into the Unreal Engine and display the content on the LED wall. Unreal was new to virtual production at the time, so I think it was a learning experience between all of us. It ended up being a lot of back-and-forth trial and error in the pre-production phase."[19]

This new technology and workflow would receive its greatest trial by fire in the short months after its realization. As viewers worldwide enjoyed *The Mandalorian* at the end of 2019 into early 2020, the Coronavirus 2019 (Covid-19) was spreading across the globe. By mid-March of 2020, countries around the world began going into lockdown, leading to delays and complete shutdowns of projects. When working on set was allowed again, health

---

[18] Mike Seymour, "Art of LED Wall Virtual Production, Part One: 'Lessons from the Mandalorian'."
[19] Cushing.

protocols demanded fewer people to be present and shooting to be stopped if someone contracted Covid-19. The extent of delays has led to the reschedule of at least two major films, *The Batman* by Warner Brothers and the third *Jurassic World* film, for release in 2022 instead of 2021, with countless others being delayed or canceled entirely.[20] The lockdown restrictions required companies and workforces to be adaptable and, for the first time, work from home in shared virtual workspaces compared to the traditional in-office: "Interest turned into necessity when the coronavirus pandemic restricted the ability to shoot global locations. If you cannot go out into the world then the next best thing is to create a photorealistic, computer-generated environment that can be adjusted in real-time."[21]

### 2.3 Virtual Production in Action

Previsualization (Previs) studios are a critical part of pre-production, responsible for helping to define the cinematic vision, provide an early form of the film cut, and create a roadmap for Production.[22] Previs is typically presented in the form of Computer Generated (CG) rough animations known as animatics. The animatics need to be updated as changes are made to a film or episodic scripts. The turnaround time in a previs studio is short, requiring a streamlined workflow. The growth of real-time render engines has dramatically benefited previs studios by reducing the amount of work required to update the previous animatic and allows a director to provide immediate feedback during planning.

---

[20] Kaare Eriksen, "Why Virtual Production Benefits Filmmaking in a Pandemic," *Variety*, May 13, 2021, accessed September 3, 2021, https://variety.com/vip/why-virtual-production-benefits-filmmaking-in-a-pandemic-1234971940/.
[21] Trevor Hogg, "Virtual Production Takes a Big Step Forward," *VFX Voice*, January 28, 2021, accessed September 4, 2021, https://www.vfxvoice.com/virtual-production-takes-a-big-step-forward/.
[22] Hamilton Lewis, "What Is Previs and Other Forms of Visualization," in *The VES Handbook of Visual Effects Industry Standard VFX Practices and Procedures* 3rd ed., eds. Jeffrey A. Okun and Susan Zwerman (New York: Routledge, 2021), 37.

*Fig. 2.8 Halon Entertainment, Previs for Ford v Ferrari 2019 © 2019 20th Century Fox (Failes, Image, How Previs Has Gone Real-Time, https://www.vfxvoice.com/how-previs-has-gone-real-time/)*

Halon Entertainment, responsible for the *Ford v Ferrari* and *War for the Planet of the Apes* previs, seen in Fig. 2.8 and Fig. 2.9, is one studio to adopt a real-time-based workflow for creating previs animatics. Compared to their previous workflow of using the Maya viewport to render out previs, the new real-time involves modeling and keyframe animation within a DCC, Maya, and then imported into Unreal.[23] In an article published by VFX Voice, Senior Visualization Supervisor Ryan McCoy comments about the use of Unreal Engine and previs at Halon Entertainment: "We also have virtual production capabilities where multiple actors can be motion-captured and shot through a virtual camera … allowing the director to run the actors through a scene and improvise different actions and camera moves, just as they would on a real set." [24]

[23] Ian Failes,"How Previs Has Gone Real-Time," *VFX Voice*, April 1, 2020, accessed September 27, 2021, https://www.vfxvoice.com/how-previs-has-gone-real-time/.
[24] Ian Failes,"How Previs Has Gone Real-Time."


*Fig. 2.9 Halon Entertainment, Previs of War for the Planet of the Apes 2017 © 20th Century Fox (Halon Entertainment, Video, War for the Planet of the Apes Previs Reel, https://vimeo.com/228733117)*

Another previs studio to implement virtual production/real-time into their animatic pipeline is The Third Floor. Responsible for the *Game of Thrones* final season previs work, The Third Floor utilized a Maya/Unreal hybrid system to plan out special FX and to do VR set scouting for the King's Landing scenes.[25] Having the ability to scout out locations and ensure CG environment integrity in real-time is invaluable, allowing for informed decisions prior to production and post-production. See Fig. 2.10 for an example of the previs done by The Third Floor. Virtual production Supervisor Kaya Jabar from The Third Floor discusses in an article by FX Guide the use of virtual production in previs. When questioned about the overlap between previs/pre-production and post-production with the real-time workflow, Kaya said the following: "We would previs everything, get that approved quickly, and then

[25] Ian Failes,"How Previs Has Gone Real-Time."

send our scenes to the finals house … It empowered us to pull quite a bit of animation work forward in production."[26]

*Fig. 2.10 The Third Floor, Previs for Game of Thrones Season 8 2019 © 2019 HBO (Failes, Image, How Previs Has Gone Real-Time, https://www.vfxvoice.com/how-previs-has-gone-real-time/)*

Once budgeting, previs, and many other elements are completed, a project can move into the production phase. Production is the stage in which a large portion of a film/episodic core content is created, whether that be principal photography for a live-action work using actors or primary layout and asset creation of an entirely CG project. Virtual production techniques have also been implemented within the production stages to augment the workflow and facilitate collaboration.

One example of virtual production for production is the Moving Picture Company (MPC) and their work on the 2019 release of *The Lion King*. The *Technicolor Quick Reference Virtual Production Glossary* defines Virtual Production as a "filmmaking technique that allows filmmakers to merge traditional practices with current and ongoing advances in real- time technology." [27] MPC was able to achieve something astounding through the use of real-time technology, the ability for filmmakers and crew members to collaborate in real-time from

---

[26] Mike Seymour, "Virtual Production Guide: Kaya Jabar, Third Floor," *FX Guide*, July 25, 2019, accessed August 29, 2021, https://www.fxguide.com/quicktakes/virtual-production-guide-kaya-jabar-third-floor/.
[27] *Technicolor Quick Reference Virtual Production Glossary*, Ver. 1, (Technicolor, January 1, 2019), https://www.technicolor.com/sites/default/files/2019-07/20190726-Virtual-Production-Glossary-v1-web.pdf.

anywhere in the world on a project that has no actual stage to shoot on. The new collaborative environment allowed people to actively design, set-dress rough elements, and plan a camera move as necessary for each film shot.[28] In a video breakdown provided by Technicolor, VFX Supervisor Robert Legato discusses the intent behind implementing virtual production techniques within an animated film: "In CG Production, you're not seeing something in real-time. You're not reacting real-time, and you have too much time to think about it. So everything becomes more intellectual, and what we're trying to do is tap into your gut-level response to things, your instantaneous art choice."[29]

*Fig. 2.11 Technicolor and Moving Picture Company, Virtual Stage Production for The Lion King 2019 © 2021 Technicolor (Technicolor, Video, The Lion King – Breaking Ground with Virtual Production, https://www.technicolor.com/thelionking#virtual-production)*

In essence, the use of virtual production turned *The Lion King* from a CG film

[28] "Technicolor's MPC Film and the filmmakers of The Lion King bring the Beloved Characters from a Disney Classic Back to the Screen – Like You've Never Seen Before," *Technicolor*, 2019, accessed October 1, 2021, https://www.technicolor.com/thelionking#virtual-production.

[29] "The Lion King – Breaking Ground with Virtual Production," Technicolor, August 7, 2019, video, 4:50, accessed October 1, 2021, https://youtu.be/shD-dQ1yPdk.

environment into a live-action shooting stage done virtually (Fig.2.11).[30]

On the other end of production is 2020 Double Negative (DNEG) with *Gunslinger,* an LED volume virtual production test.[31] Compared to MPC creating an entirely virtual stage that acts as a real environment using Virtual Reality (VR) headsets, DNEG is creating a live stage that utilizes virtual environments and LED walls to create a volume. The most significant benefit of using the LED wall volume is that it offers the ability to create multiple sets and environments that can be interchanged without physically building the set or moving locations.[32] Making the background image a live part of the stage when shooting allows for greater accuracy in color and lighting since the LED wall can serve as a form of global illumination, spilling light onto actors. Creative Director/Founder of DNEG Paul Franklin discusses *Gunslinger*'s creation and how virtual production had an impact: "Unlike a regular scenic backdrop or just a rear projection. This environment is a live 3D world. So we're creating a virtual stage but then filming it all in-camera. Virtual sets have existed for a long time in Visual Effects, using green-screen where we have to then replace the green later on in post-production. So it [Virtual Production] gives you the best of both worlds—the flexibility of a visual effects project but then the immediate realism of actual photography."[33]

---

[30] "The Lion King – Breaking Ground with Virtual Production."
[31] "Virtual Production at DNEG," *DNEG*, accessed September 23, 2021, https://www.dneg.com/virtual-production/.
[32] "Virtual Production: LED Stage Creative Test at Dimension Studios," Dimension Studio, August 6, 2020, video, 7:54, accessed September 23, 2021, https://youtu.be/SKTg83Pgfas.
[33] "Virtual Production: LED Stage Creative Test at Dimension Studios."

*Fig. 2.12 The 7 Fingers, Live Virtual Acrobatic Performance at the Performance & XR Symposium 2020 © 2020 Shocap Entertainment Ltd. (Shocap Entertainment Ltd., Video, LiViCi Presentation at Performance & XR Symposium, https://www.shocap.com/projects/livicipresentation)*

While virtual production has been adopted primarily as a production toolset, it has also given rise to an entirely new medium: Live VP performances. Showcased at Real-Time Live Siggraph 2021, The Montreal-based Circus Collective, The 7 Fingers, presented a live VP circus performance titled *LiViCi*. During the performance, Sidonie Adamson, Cameron Fraser, and Kasha Konaka performed acrobatics typically found within a Cirque du Soleil show inside a virtual world as fictional characters using motion capture technology and a real-time engine. The show was broadcast live to all attendees of Siggraph 2021, with The 7 Fingers expressing a desire/ability to make *LiViCi* or a similar performance viewable to audiences anywhere in the world in VR or on desktop computers. A recorded version of *LiViCi*'s performance can be found from the 2020 Performance and XR Symposium (See Fig.

2.12) on YouTube[34] or Shocap Entertainment's[35] website.

*The Visual Effects Society (VES) Handbook Third Edition* defines Virtual Production as the following:

> "Virtual production is the technique that uses technology to join the digital world with the physical world in real time. It can be an incredibly useful tool to help explore, create, plan and make decisions during the production process. A few examples of technology that connects the real world to the virtual world are motion capture, virtual camera, encoders, and light probes. In contrast, technologies that bring the virtual world to the physical are motion control bases, programmable cranes, LEDs, and projectors. Often the best solution may require technology from both of these categories."[36]

Under the VES definition, each of the aforementioned companies' implementations is virtual production. This once again raises the original question. What is virtual production? The definition will continue to change as production and workflows change. It will be centered around real-time technology being utilized.

## 2.4 The Current State of Virtual Production and Real-time FX

Virtual production was initially a tool used to add another dimension to a project but instead became a near requirement for completing a project in the era of Covid.

As time goes on, more and more companies are investing in the real-time technology

---

[34] "LiViCi Presentation at Performance & XR Symposium," Shocap Entertainment, Ltd., October 6, 2020, video, 6:29, https://youtu.be/Od61KOfQkFE.
[35] "LiViCi Presentation," *Shocap Entertainment*, October 7, 2020, https://www.shocap.com/projects/livicipresentation.
[36] Addison Bath, "Virtual Production," in *The VES Handbook of Visual Effects Industry Standard VFX Practices and Procedures* 3rd ed., eds. Jeffrey A. Okun and Susan Zwerman (New York: Routledge, 2021), 57.

boom. Epic Games with Unreal Engine 4/5 has worked to make the inclusion of virtual production all the easier. Magnopus has created tools that allow multi-user VR scouting within a virtual environment, while Quixel, now a part of Epic/Unreal, creates hi-resolution scans using photogrammetry.[37] VFX has already began transitioning from an entire post-production stage to a mixture of pre-production, on stage production, and post-production.

In an article by FX Guide titled "Virtual Production Supervisor at The Third Floor," Kaya Jabar said the following about virtual production and final pixel imagery: "Much attention is being drawn to virtual production lately. I've been working on it for over four years, and other people have since Avatar, but it was never conceivable that you would get final pixel in-camera. It was always a prototyping tool. With improving GPUs, especially the NVIDIA RTX cards and global illumination, I can see it being brought more on set."[38] The potential for final pixel imagery in-camera is revolutionary for film, television and visual effects, allowing elements to be capture live on stage instead of created and applied in post-production.

The realization of more advanced real-time technology has forever changed the future of film and television creation. So the question becomes, what does the future hold for virtual production and the issues that still exist? Kevin Cushing covers some of the issues that still exist, specifically for visual effects and how they are being addressed within Unreal: "Epic is focusing on real-time interactive lighting, so you do not have to wait for the lighting to re-bake out if there is a lighting change. Particles systems are being worked on so that you can have more particles with collision. Things that are inherently within the visual effects

---

[37] Debra Kaufman, "New Virtual Technologies Remake VFX's Future Pipeline," *VFX Voice*, January 2, 2020, accessed October 6, 2021, https://www.vfxvoice.com/new-virtual-technologies-remake-vfxs-future-pipeline/.
[38] Mike Seymour, "Virtual Production Guide: Kaya Jabar, Third Floor."

pipeline because they are hard to do in real-time. So the push is to take more and more of that pipeline that could actually be on screen."[39] In order to address the issues that exist within the visual effects to real-time pipeline, the first step is to identify the challenges and differences between the two systems.

## 3 The Challenges and Differences in FX Design for Real-Time

The easiest method to quantify the advantages and disadvantages of creating FX for real-time and off-line workflows is to design the same FX and test its implementation in the different pipelines, thus studying the differences and limitations of each method. A limited amount of documentation/tutorials, which is growing daily, is available discussing the creation of Houdini FX and importing it into Unreal. Most are game-based single simulations and rarely deal with types being combined for the intention of film or television. The FX creation process utilizing multiple simulation types will be presented and compared to understand better the benefits and detriments of FX for real-time and off-line. For each discussion point, three aspects will be presented: each type's role in the final FX, the real-time method, and finally, the noticeable differences and challenges faced.

### 3.1 FX Design for Off-line Methodology

Houdini Rigid-Body Dynamics (RBD) will be the lunar impact and driving force behind the other FX design and creations. Instead of simulating over a highly detailed model, RBD creates a shell that envelopes each piece of the simulation to handle interactions, allowing it

---

[39] Cushing.

to benefit from quickly solving and storing simulation information on packed geometry (points). The most significant drawback of the simulation type is that objects have to be pre-fractured, broken into smaller pieces, and re-attached to form the beginning whole. [40] Once ready, constraints will be generated from the fractured models. All the prepared elements will be packed and passed to a rigid body simulation to generate the first FX element. The simulation will then be cached to prevent data loss or recook errors. The cached sim will be assigned the desired shaders, rendered, and composited with the other FX elements.

Houdini particles will generate the small fragment pieces and substrate that are kicked up by an impact. An interesting behavior can be created inside the POP simulation due to the lack of atmosphere and low gravity. Since POP simulations only run over points, this simulation type is relatively quick to compute but requires a great deal of attention in the sourcing. The work begins by taking the prepared RBD simulations and creating point sources for the simulation. A custom velocity force is created on the points. Once the sources are ready, the particle simulation is run and cached out. After the simulation is cached out, any number of post-processing effects can be done, including the assignment of shaders to the particles or particle scale randomization and final renders.

The last simulation type is Houdini volumes (Pyro). Volumes will be used to represent the large amounts of lunar material that will be kicked up from the surface on impact. Houdini volumes are built upon a unit called a voxel, or a three-dimensional pixel. Voxels allow Houdini to simulate real-world-based volumes and render depth. The volume sourcing

---

[40] Craig Zerouni, "Rigid-Body Dynamics," in *The VES Handbook of Visual Effects Industry Standard VFX Practices and Procedures* 3rd ed., eds. Jeffrey A. Okun and Susan Zwerman (New York: Routledge, 2021), 577.

is dependent upon the RBD simulation to be complete and accessible. Once ready, the RBD debris is used to create the simulation source volumes. Custom forces are then created to affect the motion of the volumes. The source volumes and custom forces are passed into a pyro simulation, along with the RBD elements as colliders.

### 3.2 [Methodology] Real-time FX Environment Creation and Setup

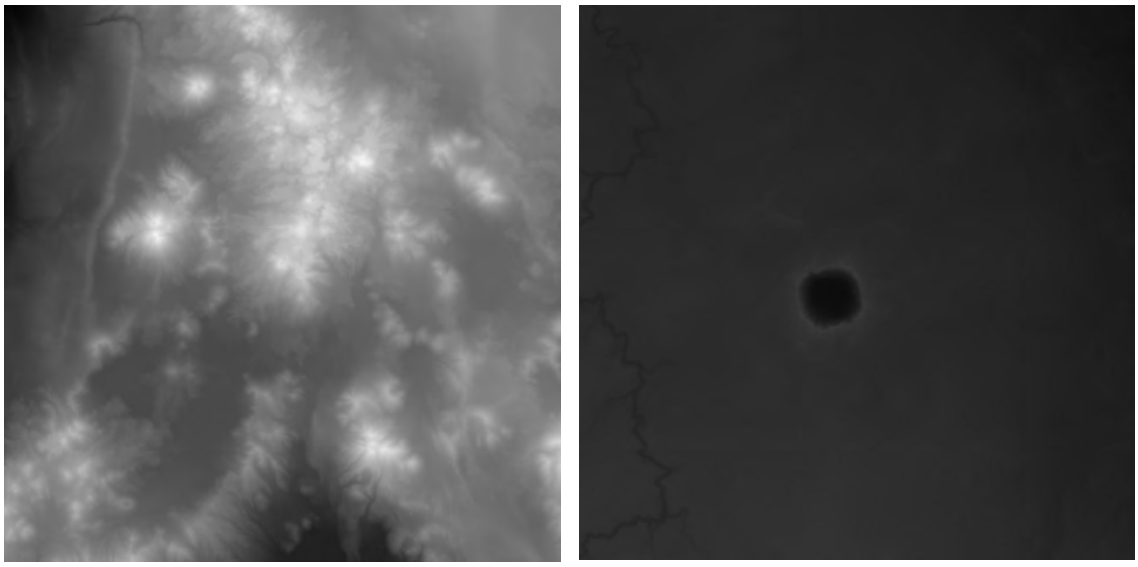


*Fig. 3.1 Scan Data of Atacama Desert, Chile (Terrain Party, Image, http://terrain.party/api/export?name=Atacama+Desert%2C+Chile+terrain_20x20&box=-68.188207,-22.851705,-68.383431,-23.031367)*

*Fig. 3.2 Scan Data Meteor Crater, AZ (Terrain Party, Image, http://terrain.party/api/export?name=Meteor+Crater%2C+AZ&box=-110.962603,35.074682,-111.072367,34.984851)*

Before designing the core FX within the scene, the environment must be created to define the setting and any elements that may need to be accounted for in the FX preparation and execution. Houdini Heightfields were chosen to create the terrain of the lunar surface using real-world scan data of the Atacama Desert, Chile, and Meteor Crate,

Arizona (Figs. 3.1 and 3.2 gathered from *Terrain Party*[41]). Heightfields can use map elevation data to generate a detailed two-dimensional (2D) volume that utilizes an attribute called @height. Depending on the scene requirements, the 2D volume can then be converted to polygonal geometry or remain as a heightfield. Fig. 3.3 shows the result of combining the two scan data using Houdini's heightfield tools.

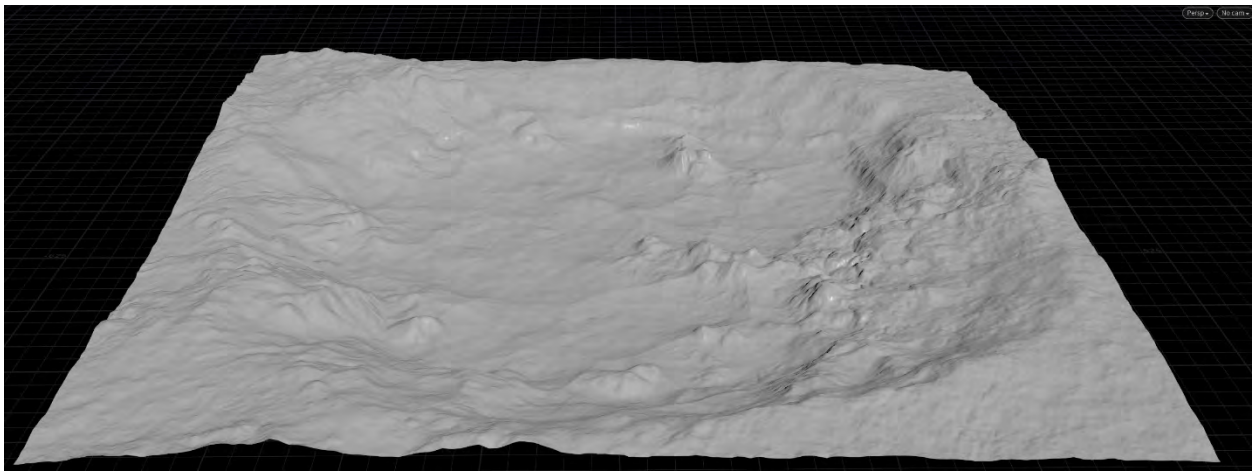


*Fig. 3.3 Houdini HeightField from Scan Data*

The largest of the challenges is transferring the Houdini heightfield to Unreal since Houdini's native format is based on a two-dimensional volume that Unreal does not utilize. Instead, Unreal displays a heightfield as a Landscape/Terrain Actor, which is divided into flat grid components. The component is roughly moved to represent the surface, while height data is stored in a texture file on a per-component basis to be rendered by the engine.[42] Unreal does accept height data in a greyscale image exported from Houdini using a COP network. However, the landscape tool in Unreal is particular in how the image is formatted when exported or stepping issues occur in the landscape. The image must be exported as a

[41] "Terrain Party," last updated 2021, accessed March 2, 2021, https://terrain.party/.
[42] "Landscape Technical Guide," *Unreal Engine Docs*, ver. 4.26, last updated 2021, accessed March 3, 2021, https://docs.unrealengine.com/4.27/en-US/BuildingWorlds/Landscape/TechnicalGuide/.

sixteen-bit greyscale png file. Upon importing into Unreal, a new issue appeared in the mismatching scale between the heightfield and the exported Houdini buildings and FX elements. The differences in heightfield format and world-scale requires an instance of Houdini to run in Unreal through the Houdini Engine plug-in to construct the environment and then output assets to Unreal. Please see Appendix A.1 for Creating Houdini Heightfields.

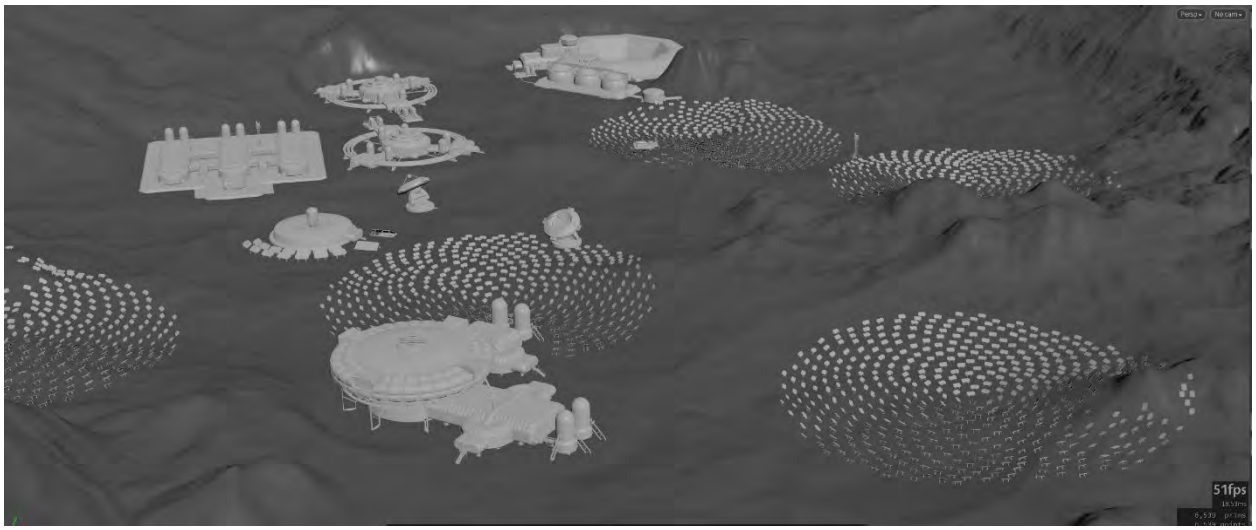



*Fig. 3.4 Houdini Base Layout*

After the terrain, the next element is a lunar base complex that introduces inorganic elements to the environment and FX. Models were selected from the Kitbash3D kit “Lunar Base” to implement a procedural layout utilizing Houdini VEX code to create the desired placement seen in Fig 4.4. Points were scattered onto the previously created heightfield with varying orientation, scale, and variant attributes to determine the model used. Reference Appendix A.2 for Base Layout Using Variant Attribute.

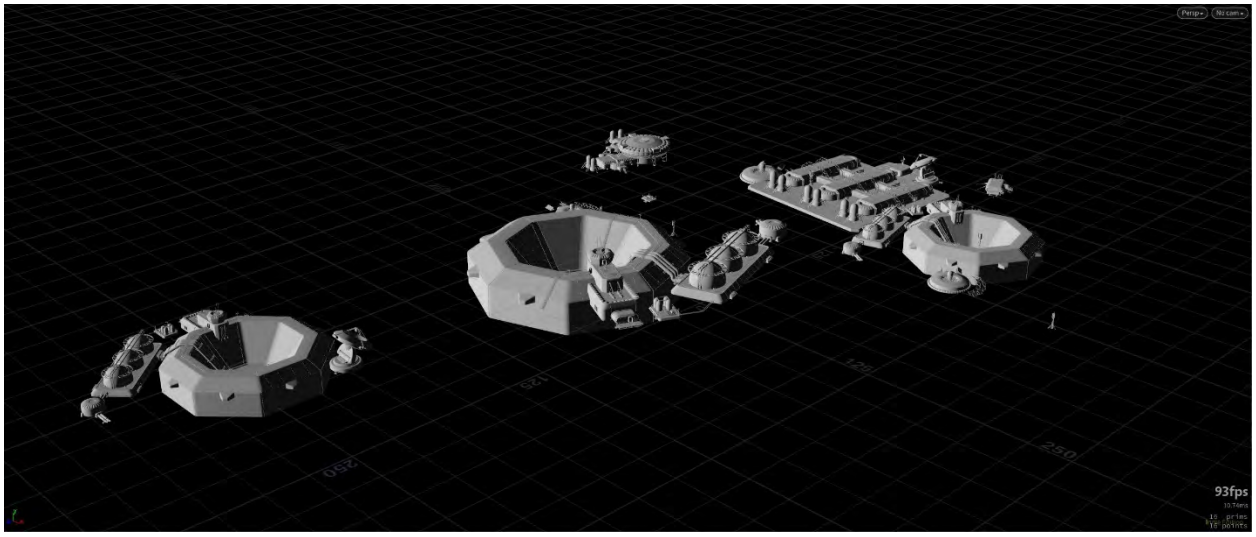



*Fig. 3.5 Building distribution not working without being self-aware*

With the buildings laid out in a desirable location, a new problem became apparent. Since the packed models are instanced onto scattered points, the varying height of the heightfield is not considered, causing certain parts of the models to either interest or float above the heightfield. A ForEach SOP was used to update the height attribute of the lunar surface for each newly placed building model to represent the flattening of the surface for construction. Fig. 3.6 and 3.7 demonstrate the method and final result used to address this problem. See Appendix A.3 Updating Heightfield for Base Location.

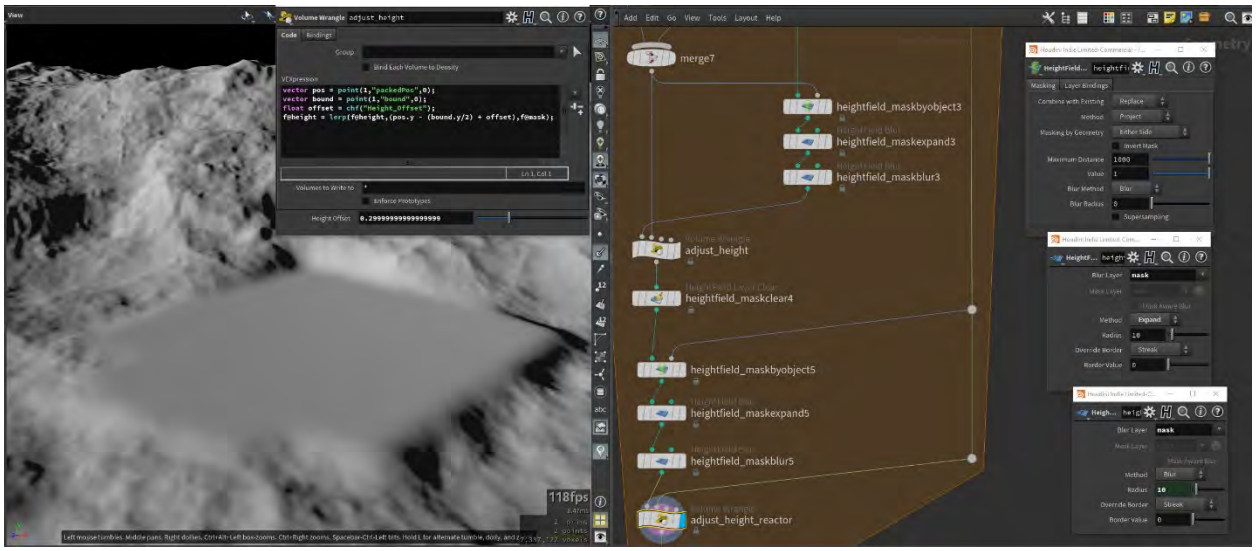


*Fig. 3.6 Adjusting the heightfield for the buildings*

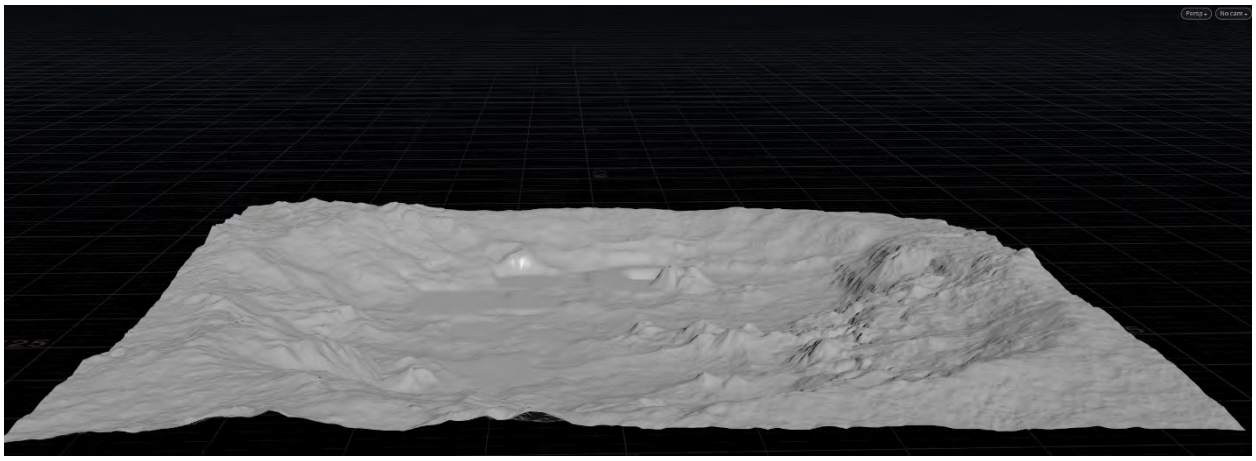


*Fig. 3.7 Houdini HeightField Updated*

The last required element of the layout is the need for an inorganic asset included in the core FX. Many of the techniques used in laying out the main buildings are reused in the solar panel array placement. The only difference between the two systems is that the solar panel mask removes any area overlapping a building before scattering points using the mask. Reference Appendix A.4 for Solar Panel Phyllotactics Based on Lunar Surface.

Once the lunar surface, building layout, and solar panel placement were complete, the elements were imported into Unreal. Similar to Houdini, Unreal also has a tool named, Landscapes, designed to read in height data and create an environment. In order to avoid any stepping issues (Fig. 3.8) when using a greyscale png file (Fig. 3.8) the heightfield_output node was used. The heightfield_output node built into Houdini writes out a 16bit greyscale image that reads correctly into Unreal. View Appendix A.5 for the method for Exporting Heightfield Data from Houdini to Unreal.

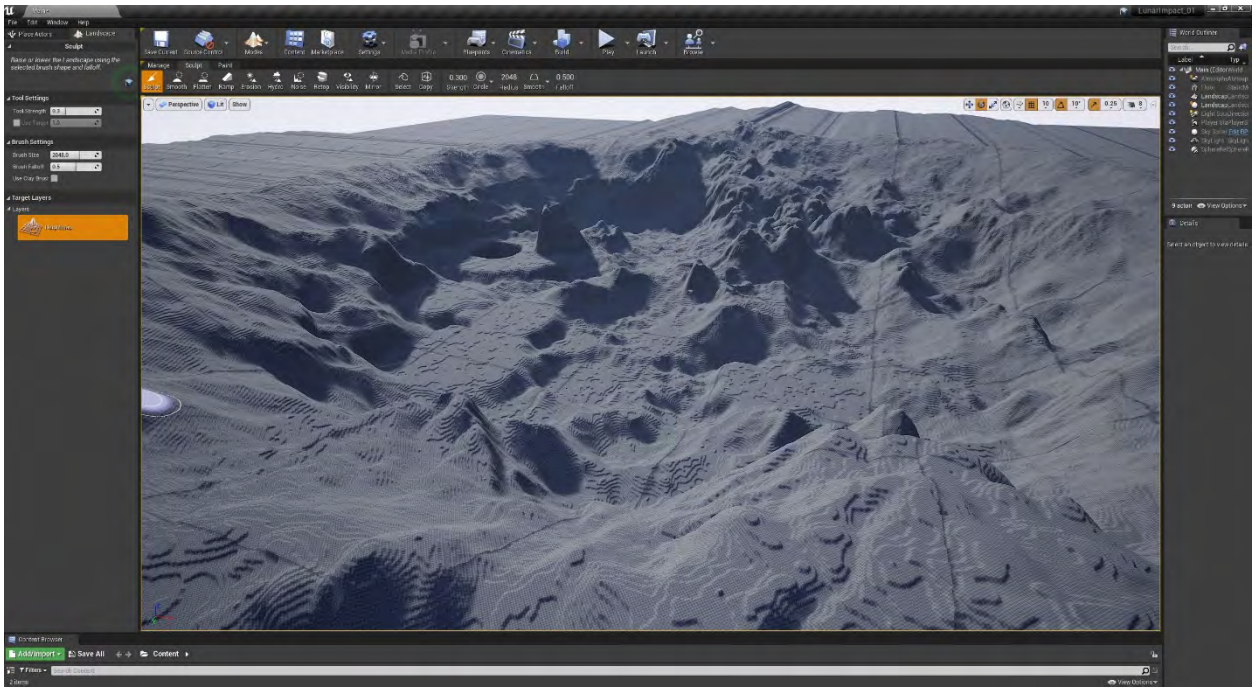


*Fig. 3.8 Houdini HeightField to Unreal Landscape not working*

The landscape and exported Houdini FX elements do not share the same world scale or position. The difference in world scale and orientation creates the need for alignment by hand, causing inaccuracy between the landscape, built environment, and FX elements. The Houdini Engine plug-in for Unreal was used to eliminate the need for hand alignment.[43] The Houdini Engine uses a Houdini Digital Asset (HDA) to run an instance of Houdini within Unreal and output Unreal actors and assets.[44] The final result of the assets created in Unreal using the HDA is seen in Fig. 3.9 with the breakdown available in Appendix A.6 Houdini Engine HDA Setup.

[43] "Houdini Engine, Unreal Plug-In," *SideFX Houdini*, accessed April 2, 2021, https://www.sidefx.com/products/houdini-engine/plug-ins/unreal-plug-in/.
[44] "Assets," *Houdini Engine for Unreal*, ver. 18.5, last updated 2021, accessed April 2, 2021, https://www.sidefx.com/docs/unreal/_assets.html.

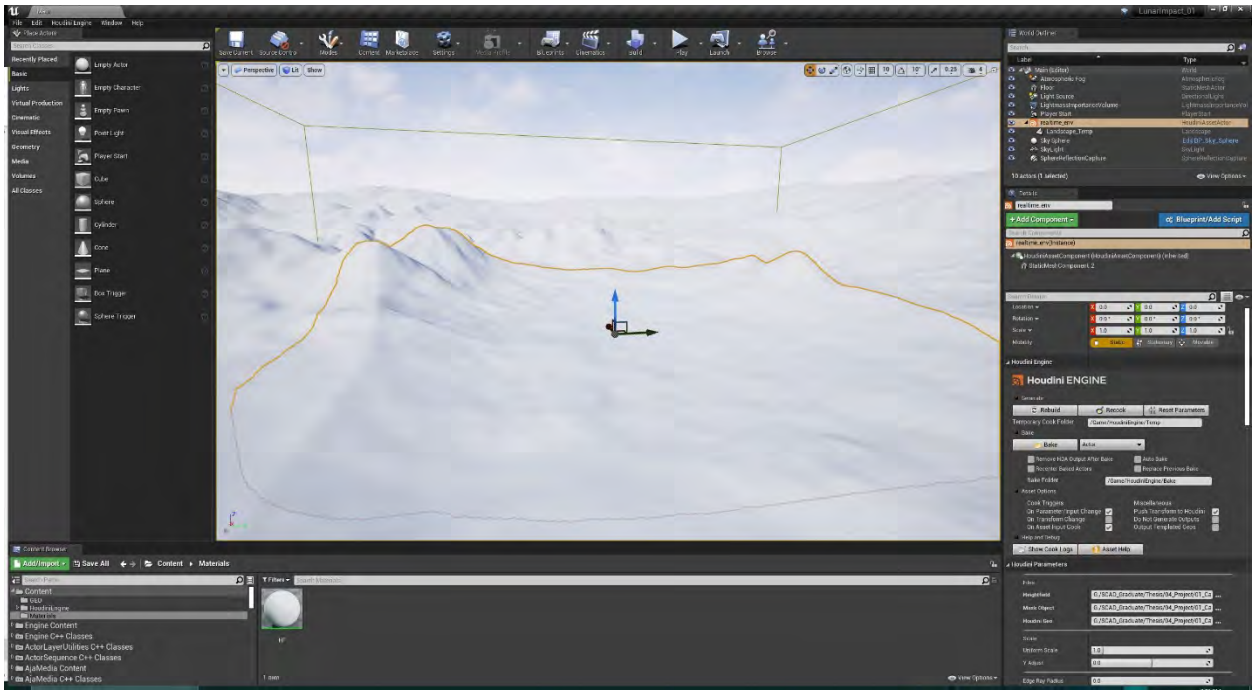



*Fig. 3.9 Houdini Engine HDA creates seamless Unreal Landscape and Static Mesh Actor*

### 3.2.1 [Methodology] Real-time Rigid Body Dynamics (RBD) Destruction

*Fig. 3.10 Houdini Lunar Impact RBD Fracture Sim Proof of Concept*

The leading FX must be designed based on the intended use within Unreal using the lunar surface now defined by Houdini heightfields. Rigid Body Dynamics (RBD) were chosen as the driving simulation of the rocks impacting the moon's surface, as illustrated by the

proof of concept shown in Fig. 3.10. RBD benefits from its ability to quickly solve and store simulation information on packed geometry (points).

The main challenge was designing the fracturing system for importing and optimization since Houdini RBD achieves complexity through high-poly meshes with numerous fractures, whereas Unreal, utilizes materials/shaders. The key was to find a balance of enough geometry to create an exciting simulation within Houdini that is light enough to run within Unreal.

Unreal accepts animated geometry in two forms, FBX or the experimental Alembic (ABC) cache.[45] Each file format has benefits and drawbacks within the Unreal Engine. The FBX format, when exported from Houdini, will return without animation when imported into Unreal unless exported packed, with a non-overlapping name attribute, using the Houdini custom ROP node "RBD to FBX." In contrast, the alembic cache can be exported as packed or unpacked geometry, with a non-overlapping name attribute to import with the animation attached.[46] However, it loses access to the static mesh attribute within Unreal. The process used for preparing the RBD mesh is available in Appendix B.1 Preparing RBD FX Mesh.

---

[45] "Alembic File Importer," *Unreal Engine Docs*, ver. 4.26, last updated 2021, accessed April 12, 2021, https://docs.unrealengine.com/4.26/en-US/WorkingWithContent/Importing/AlembicImporter/.
[46] "Introduction to Alembic | Live Training | Unreal," Unreal Engine, September 21, 2017, video, 55:45, https://youtu.be/rDHxxk0y2D4.

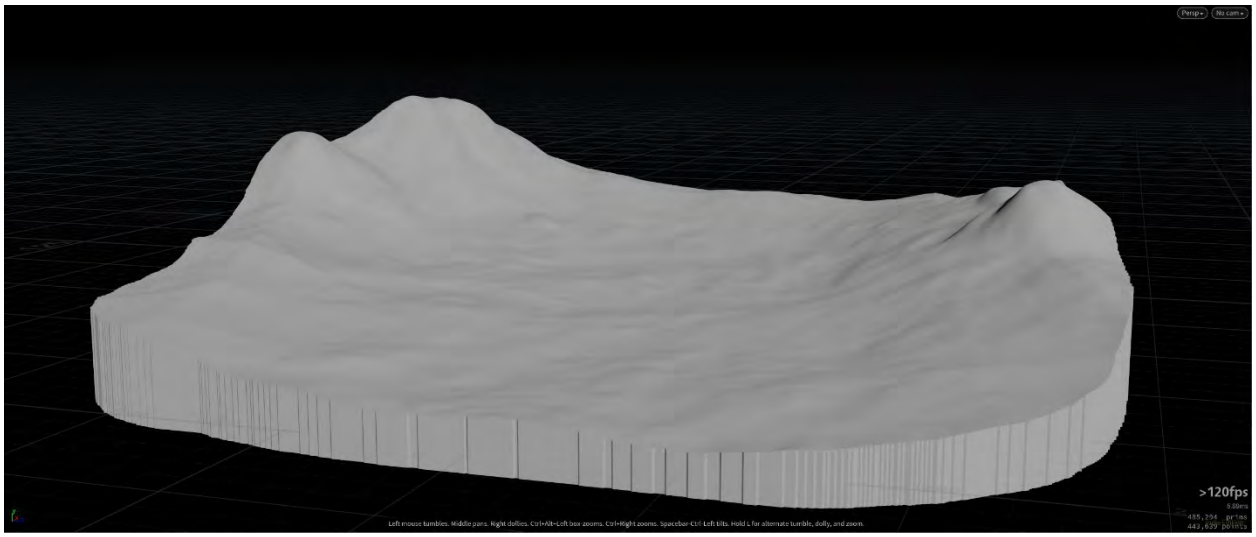



*Fig. 3.11 Houdini RBD Fracture Prepped Geo*

With the FX mesh complete, seen in Fig 3.11. The process of fracturing the FX model presents a new challenge when designing for Unreal. Polycount is especially crucial in maintaining specific Frames Per Second (FPS) when running in real-time. The off-line practices would create fractures and simulate over those connected pieces. However, for real-time, a multi-level fracture system creates a bare minimum number of fractures and then refractures those pieces based on intended impact areas was created. View Appendix B.2 Preparing Fracture RBD FX Mesh for the model prep.

The difference in polycount between the original FX mesh (left) and the prepared fracture mesh (right) is shown in Fig 4.12.

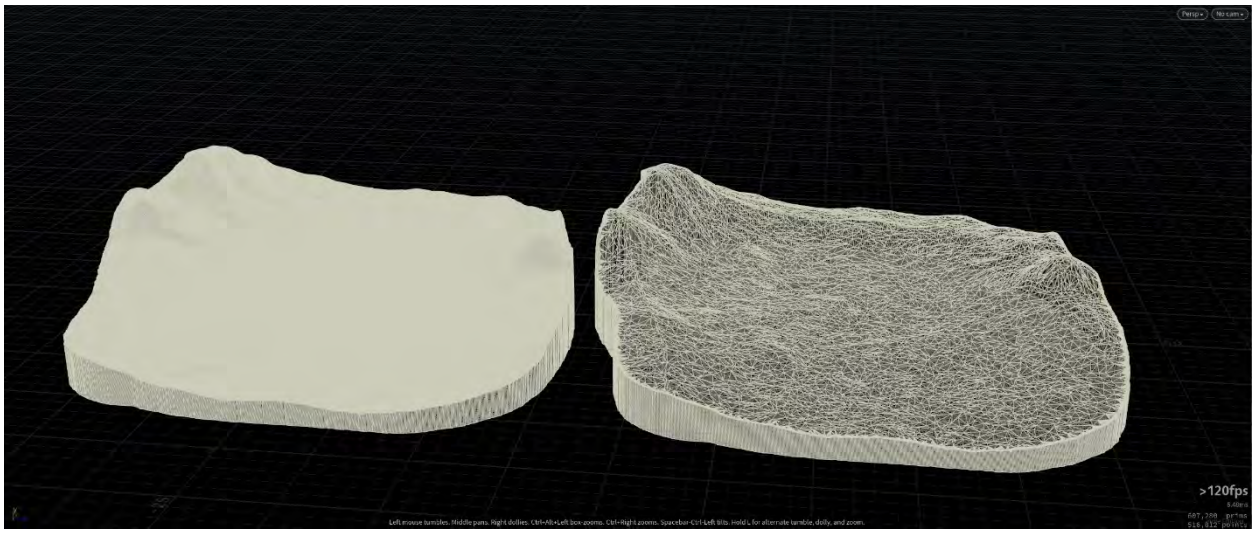



*Fig. 3.12 Original FX Mesh vs. Fracture Ready Mesh*

Once the model is cleaned and prepared, the next step in fracturing would be to cut (fracture) the model using one of two methods, Voronoi fracture or boolean fracture. Voronoi fracture utilizes points to cut the model into the desired number of pieces but runs into the issue of quickly producing a repeating pattern. Boolean fracture, which can be used in the RBD Material fracture node, can produce a much more organic geometry but can be unstable. The produced geometry pieces often intersect while also having a large polycount. A modification of the Voronoi fracture method was used to focus the fractures in specific areas and add noise to the fracture edge. Shown in Fig. 3.13 are the beginning fracture points of the FX mesh. The point scatter breakdown is seen in Appendix B.3 Fracture Point Creation.

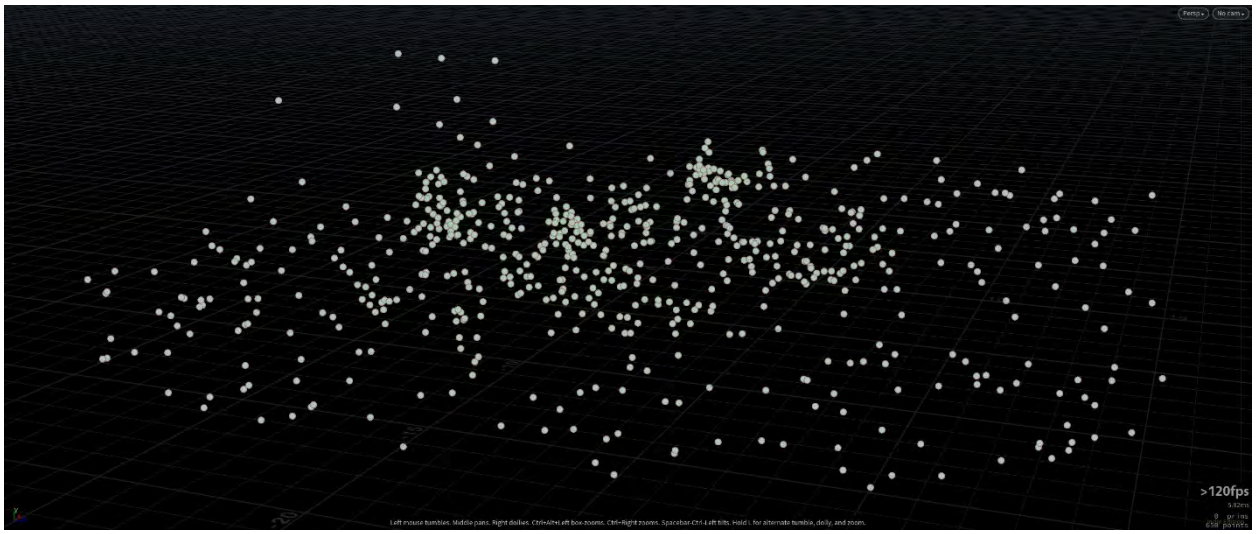



*Fig. 3.13 Fracture Points*

After the points are created, all necessary elements are ready to fracture the RBD model. The multi-level fracture system is outlined in Appendix B.4 Fracture FX Mesh.

Interior face detail was omitted to reduce polycount. A clean node was used to address the problem of any unwanted points or degenerate primitives that could result in crashes in Unreal when loading the animated simulation. A divide SOP was needed to eliminate an error by setting the “Maximum Edges” to three. Finally, Unreal would complain unless the UV’s were very carefully laid out. Reference Appendix B.5 RBD Mesh Cleanup and UV Layout.

Once repacked, the fractured mesh, seen on the right of Fig. 3.14, is ready for simulation. The fracture method used above is applied to all RBD elements, including the impacting space rocks and solar panels.

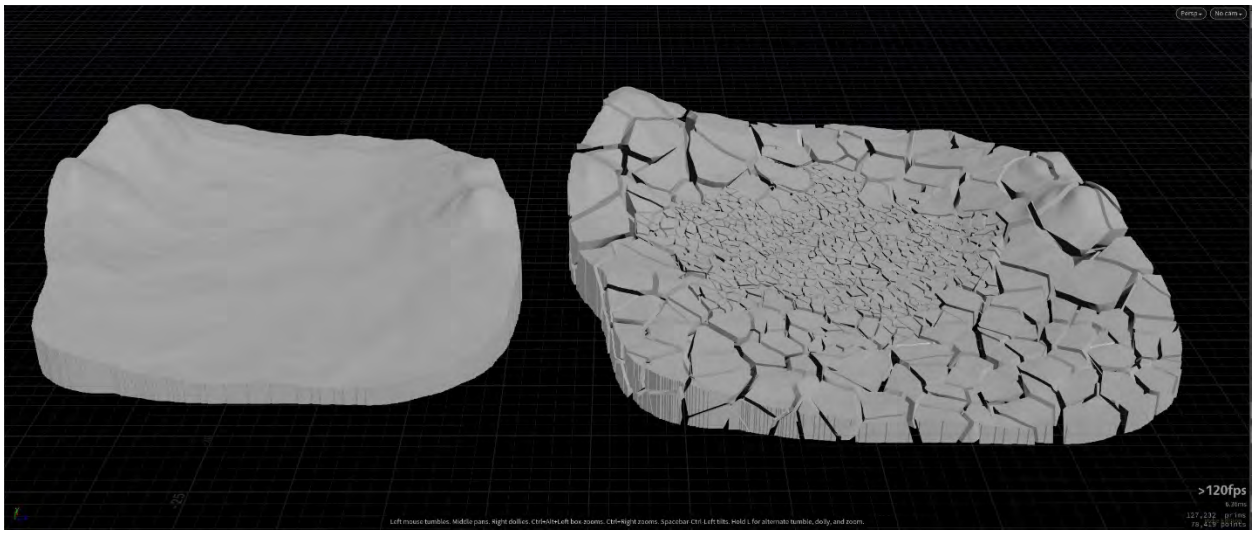



*Fig. 3.14 RBD Fractured Geo*

With the fractured geometry ready, the next step was to set up the RBD simulation attributes that are passed to the Houdini Dynamic Operator (DOP) network. Attributes needed to be created to control the simulation include i@active, which tells the solver whether to consider or ignore an object in its calculation. When necessary, ignoring a piece in the solver calculation increases the sim's efficiency and reduces run time. However, creating a blanket @active attribute would cause all pieces to be affected by sim forces, including gravity, causing the outer edge pieces to fall away from the environment heightfield edge created in *section 4.1 FX Environment Creation*. Instead, all pieces are turned inactive and then reactivated based on the expected impact sites grouped. See Appendix B.6 Set Active and Inactive Simulation Geometry for a complete walkthrough of the techniques used.

Constraint geometry is used to hold the pieces together and provide more interest to the simulation behavior (Fig. 3.15). Constraints behave by acting as a force that attempts to keep the attached pieces together. The process outlined in Appendix B.7 Create Constraint Geometry is identical for all simulation elements, including the lunar surface, impacting

rocks, and solar panels, minus a few input values.

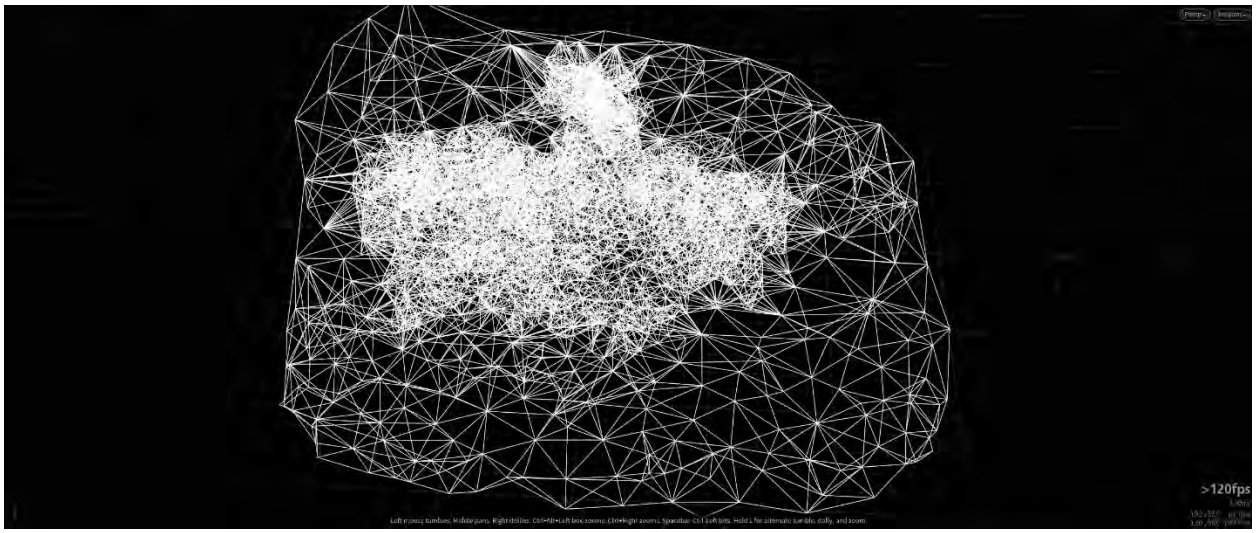



*Fig. 3.15 Lunar Surface RBD Constraint Geometry*

After completing the simulation preparations, a DOP network needs to be created, allowing Houdini to solve the object interactions. DOPs work in a specific way regarding the importing of information and what element takes precedence. Data flows from left to right, top to bottom; therefore, the sim collects all the required information and then applies it in that order. All DOP networks follow this structure regardless of the simulation type. The DOP network breakdown can be found in Appendix B.8 RBD Simulation DOP Network Import or in Fig. 3.16.

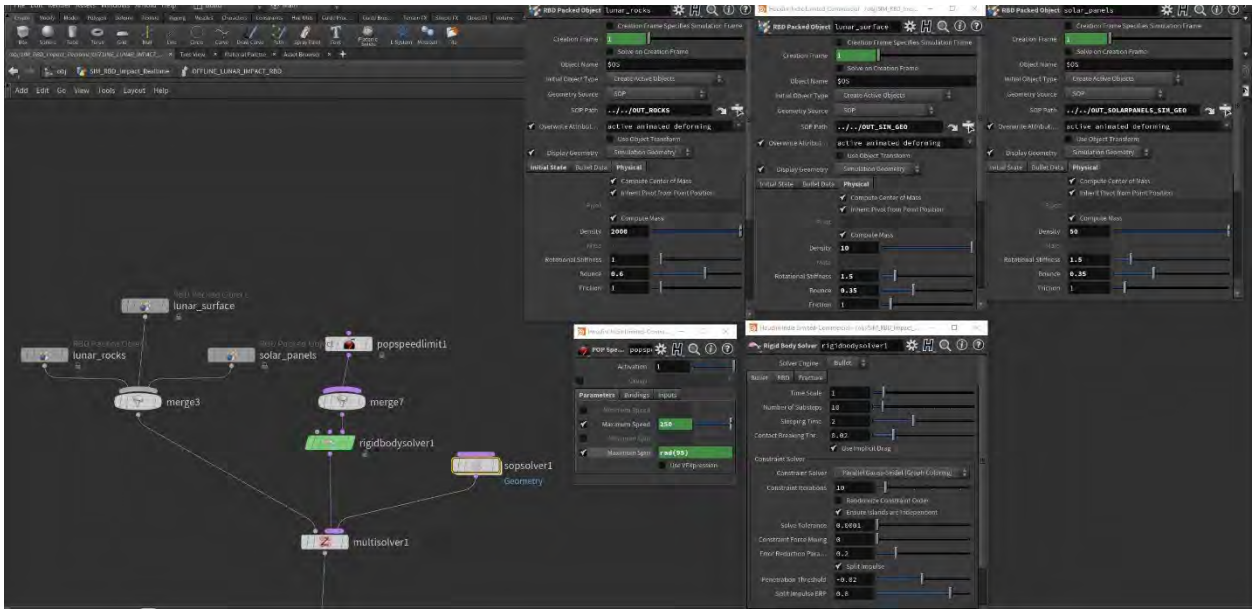



Fig. 3.16 RBD Simulation Import Geo

Collision geometry includes the heightfield environment created earlier or any animated object that would be needed from outside the DOP network, seen in Fig. 3.17. Collision specifics are referenced in Appendix B.9 RBD Simulation DOP Network Collisions.

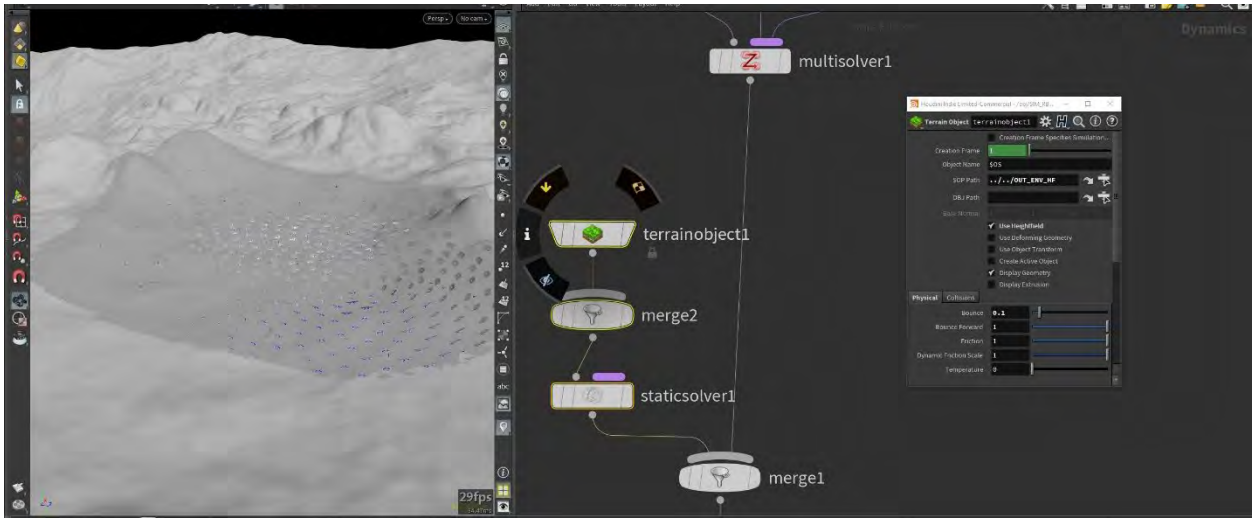



*Fig 4.17 RBD Simulation Collisions*

Constraints applied to the simulation geometry and any final force applied to the overall scene is seen in Fig. 3.18. Each constraint set network created for a specific object must

have its own constraint system. For example, the lunar surface, solar panels, and impacting rocks have their own constraint systems within the DOP network to apply their constraints. See Appendix B.10 RBD Simulation DOP Network Constraints and Forces.

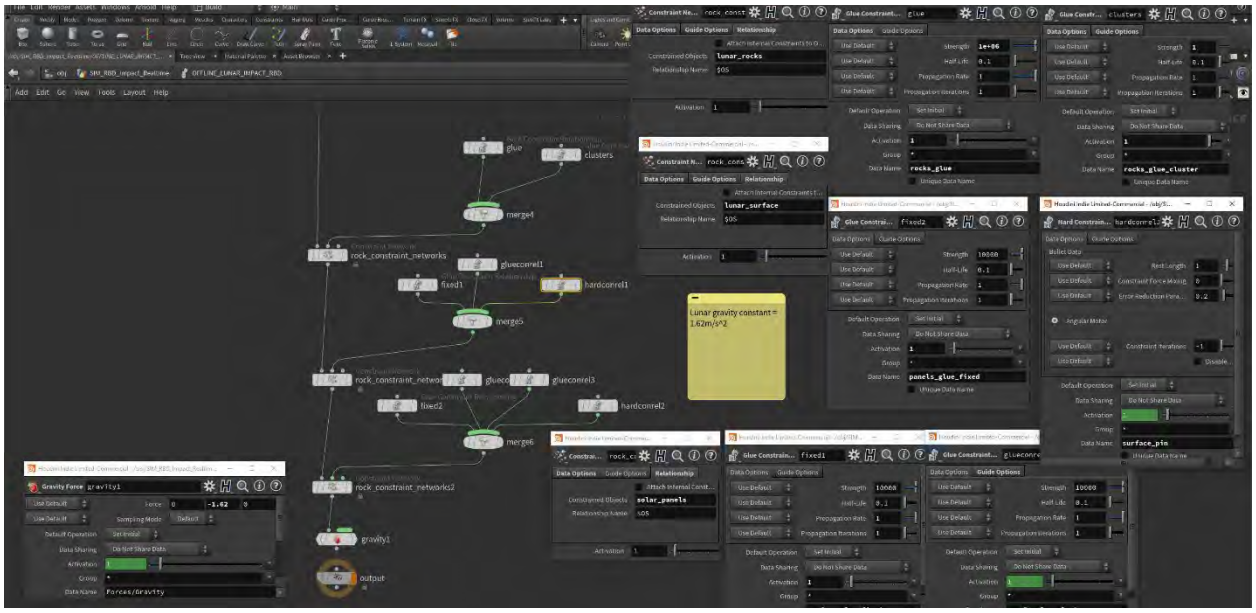



*Fig. 3.18 RBD Simulation Constraints and Forces*

The results of the Houdini RBD simulation can be seen in Fig. 3.19. The simulation was cached out and stored as points in a Houdini bgeo.sc file sequence to prevent data loss or recook errors.

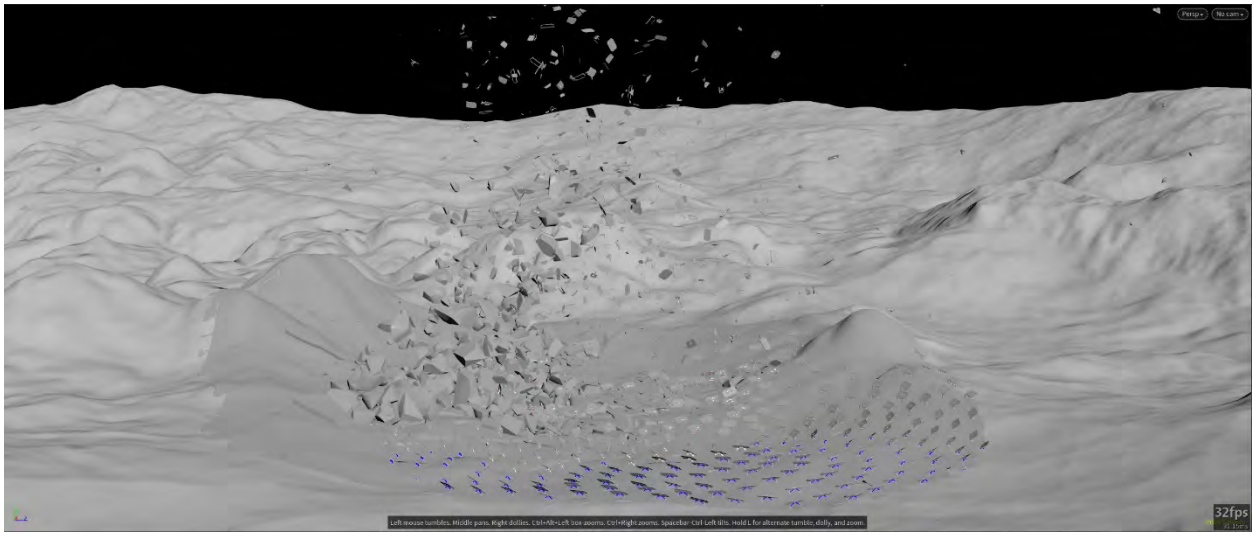


*Fig. 3.19 Houdini RBD Simulation*

A standard file format must be chosen to begin the process of importing into Unreal. The experimental ABC file was chosen due to one crucial weakness with the FBX file.[47] In Unreal, instead of importing one clean file, each component of the FBX, static mesh, animation, and bones, are imported as separate pieces, requiring time spent on reattaching.

Once cached, the high-res pieces can be substituted into the low-res animation. Please reference Appendix B.11 Houdini RBD Caching.

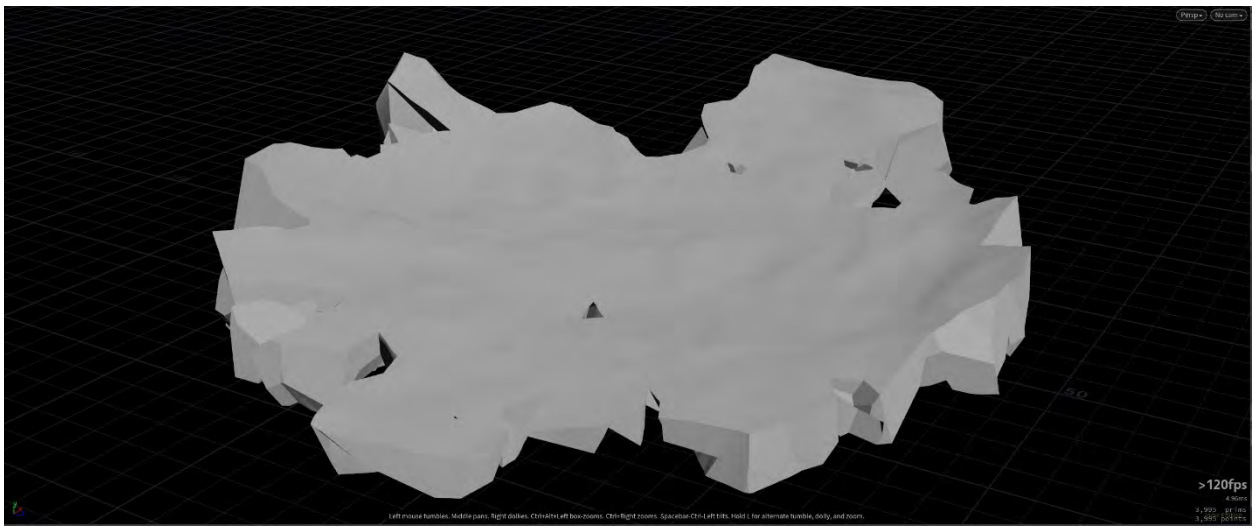



*Fig. 3.20 RBD Export Prep Active Geometry*

Once the animation is reapplied to the high-rez mesh, the following process breaks apart the model based on the pieces (Fig. 3.20) and perform final export preparations using the SideFX Labs tools. SideFX Labs is a collection of common workflows, including real-time FX, that is extremely useful for any Houdini artist.[48] However, it was decided to merge the inactive and active pieces into a single imported alembic file to solve an issue with normals

[47] "FBX Content Pipeline," *Unreal Engine Docs*, ver. 4.26, last updated 2021, accessed March 3, 2021, https://docs.unrealengine.com/4.26/en-US/WorkingWithContent/Importing/FBX/.
[48] "SideFX Labs," *SideFX Houdini*, accessed April 5, 2021, https://www.sidefx.com/products/sidefx-labs/.

inside Unreal.

Prior to exporting, the geometry needs to be prepared and optimized for real-time. The active and inactive pieces are separated and any unneeded attributes removed, shown in Appendix B.12 RBD Export Preparations, Active and Inactive.

The resulting inactive geometry now has all of its interior coplanar faces removed and fused into one single piece of geometry using the Labs Destruction Cleanup SOP[49], shown in Fig. 3.21. Once the inactive geometry is turned into a single mesh, the base of the model needs to have its poly counts reduced due to the process of fracturing, causing a significant increase. However, this did cause an issue upon import into Unreal. When the mesh is reduced below an unknown threshold, the model loses too much information, and Unreal crashes. The poly reduction amount must be handled on a case-by-case basis. The reduction method is available in Appendix B.13 Poly Reduction of Inactive Geometry.

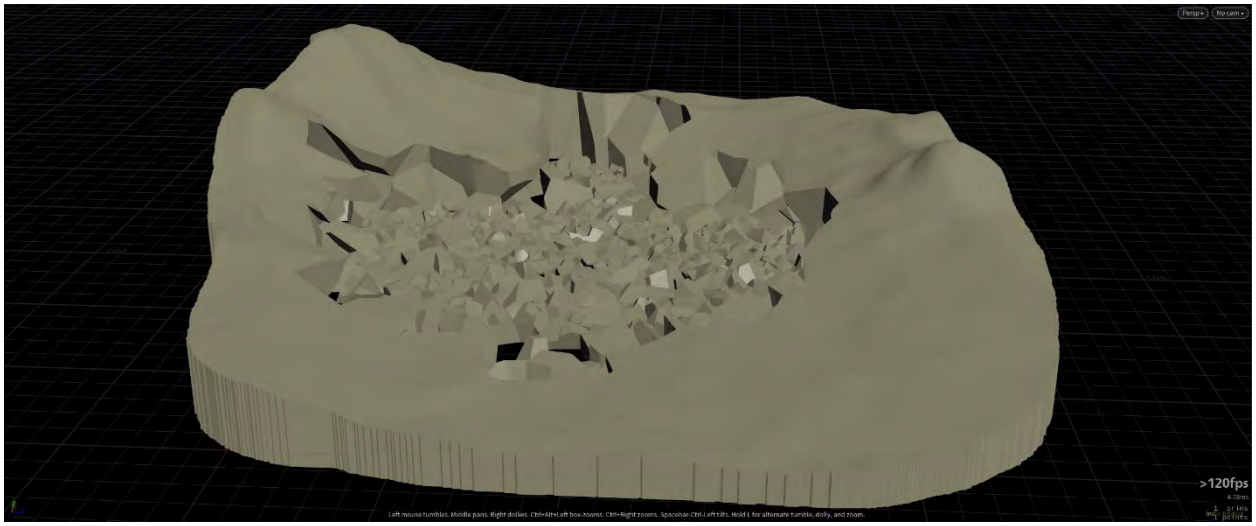



*Fig. 3.21 RBD Export Prep Inactive Geometry*

[49] Paul Ambrosiussen, "Optimizing Destruction Simulations for Real-Time Solutions," *SideFX Houdini*, ver. 16, last updated April 6, 2017, accessed April 7, 2021, https://www.sidefx.com/tutorials/optimizing-destruction-simulations-for-real-time-solutions/.

After successfully creating the landscape and importing static mesh elements into Unreal using the Houdini Engine, the RBD cache was imported and placed within the scene, and a texture was applied. When a normal map was fed into the shader, it was discovered that the normal between the inactive and active elements of the simulation did not line up, causing a near inversion of color, normal maps, and emission. Note the difference between the center (active) and outer (inactive) pieces of the model seen in Fig. 3.22. Through extensive troubleshooting, it has been determined that the Houdini Engine performs an alteration of the static mesh vertex normal upon the creation of the Unreal actors. It is uncertain what exact alteration method has been applied, but it is suspected to deal with the difference between Houdini and Unreal's coordinate systems. Houdini utilizes a right-handed Y up coordinate system, while Unreal uses a left-handed Z up system.[50] The exporting of the active and inactive pieces is further explained in Appendix B.14 Export RBD Alembic (ABC) File.

[50] "Coordinate Systems," *Houdini Engine for Unreal*, ver. 18.5, last updated 2021, accessed July 25, 2021, https://www.sidefx.com/docs/unreal/_coordinate_system.html.

*Fig. 3.22 Houdini Engine Normals Issue*

The new alembic file was exported and brought into Unreal. When importing the experimental alembic cache, specific settings must be chosen to be successful. If the settings are incorrect, either the animation will fail to be correctly attached to the mesh, or the mesh will not be created. At the same time, detail needs to be brought back to the model using texture baking from Houdini[51]. The steps needed to import an alembic file into Unreal and sequencer, along with the beginning of Houdini texture baking, are in Appendix B.15 RBD Unreal Import/Setup and Texture Baking. The final results are shown in Fig. 3.23 and Fig. 3.24.

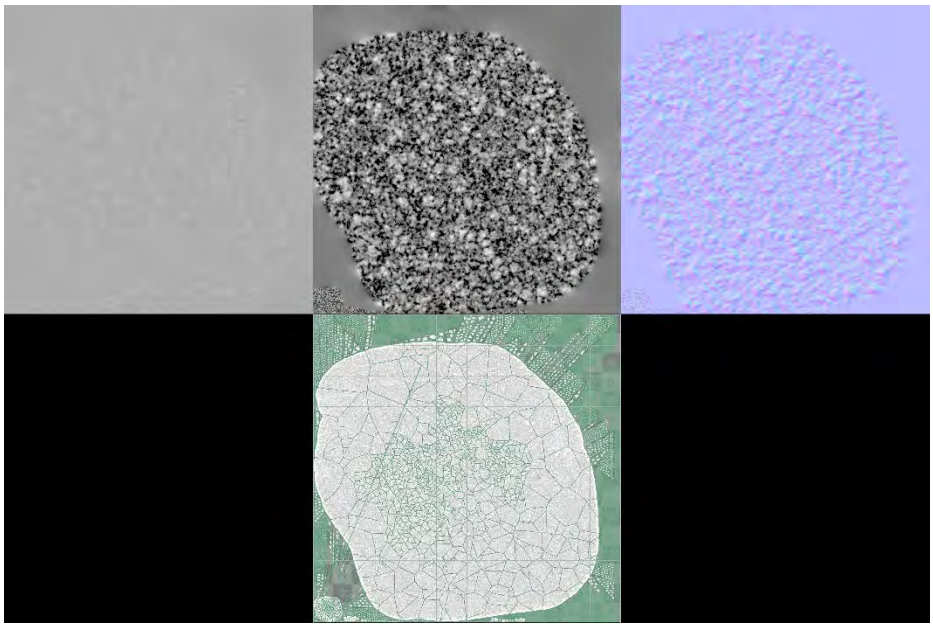


*Fig. 3.23 Houdini Bake Texture Example*

[51] "Bake Texture," *SideFX Houdini*, ver. 18.5, last updated 2021, accessed July 22, 2021, https://www.sidefx.com/docs/houdini/nodes/out/baketexture.html.

*Fig. 3.24 Unreal RBD Import*

Optimization and final setup of the RBD alembic files revealed a few issues. The most significant problem for setup and optimization came in the form of lighting and polycount. The easiest way to identify where exactly the problem exists is to use component profiling[52] and the "stat unit" line command in Unreal, accessed using the "`" symbol. Once it was determined that the GPU was where the bottleneck was occurring, value above 30 milliseconds (ms), the next step was to profile the GPU using the "profilegpu" command and establish what component of the scene was draining resources.[53] By using Unreal's profiling tools, it was clear that the two elements with the highest millisecond (ms) reading were the lighting and BasePass.

The alembic cache is an animated geometry; thus, Unreal's default, static lighting build

---

[52] "Performance and Profiling," *Unreal Engine Docs*, ver. 4.26, last updated 2021, accessed July 30, 2021, https://docs.unrealengine.com/4.26/en-US/TestingAndOptimization/PerformanceAndProfiling/.
[53] "GPU Profiling," *Unreal Engine Docs*, ver. 4.26, last updated 2021, accessed July 31, 2021, https://docs.unrealengine.com/4.26/en-US/TestingAndOptimization/PerformanceAndProfiling/GPU/.

will not work once the FX begins. Instead, the cache has its Cast Shadow turned on, and Dynamic shadow is set to True, meaning the scene light had to be converted from a static light to a stationary light for the entire scene. The changing of the light type had a drastic impact on overall scene performance. The change in lighting had such an impact that the Unreal Landscape had to have 'receiving dynamic shadows' turned off to improve the frame rate.

When building static lighting within Unreal, the process of building would begin to cause heat issues within the user's computer. The problem would continue to grow as FX, and the project as a whole, grew. An application called SWARM is responsible for building the lighting from Unreal.[54] Adjusting the number of resources allocated to SWARM can alleviate/prevent any heating issues during the lighting build. The instructions on adjusting SWARM are available in Appendix B.16 Unreal Light Building and SWARM Application.

BasePass, on the other hand, does not provide specifics when identifying what exactly is draining the system. The BasePass is responsible for multiple parts of the final rendering inside Unreal, but the most important is saving static lighting to the G-Buffer before it is passed to the GPU and forward rendering of the scene dynamic lighting.[55] The elements that have the highest cost of the BasePass are rendered resolution, shader complexity, and the number of objects/triangle count.[56] The amount of fracture on the solar panels and the lunar surface was reduced to lower the total number of triangles and limit the cost on the BasePass. At the same time, the background base elements were merged using the merge

---

[54] "Unreal Swarm," *Unreal Engine Docs*, ver. 4.26, last updated 2021, accessed July 30, 2021, https://docs.unrealengine.com/4.26/en-US/RenderingAndGraphics/Lightmass/UnrealSwarmOverview.
[55] Oskar Świerad, "Unreal's Rendering Passes," *Unreal Art Optimization*, last updated August 8, 2019, accessed July 31, 2021, https://unrealartoptimization.github.io/book/profiling/passes/.
[56] Oskar Świerad, "Unreal's Rendering Passes."

actor tool and had Level of Detail (LODs) created.

### 3.2.2 [Methodology] Real-time Particle Operators (POP) Dust and Debris

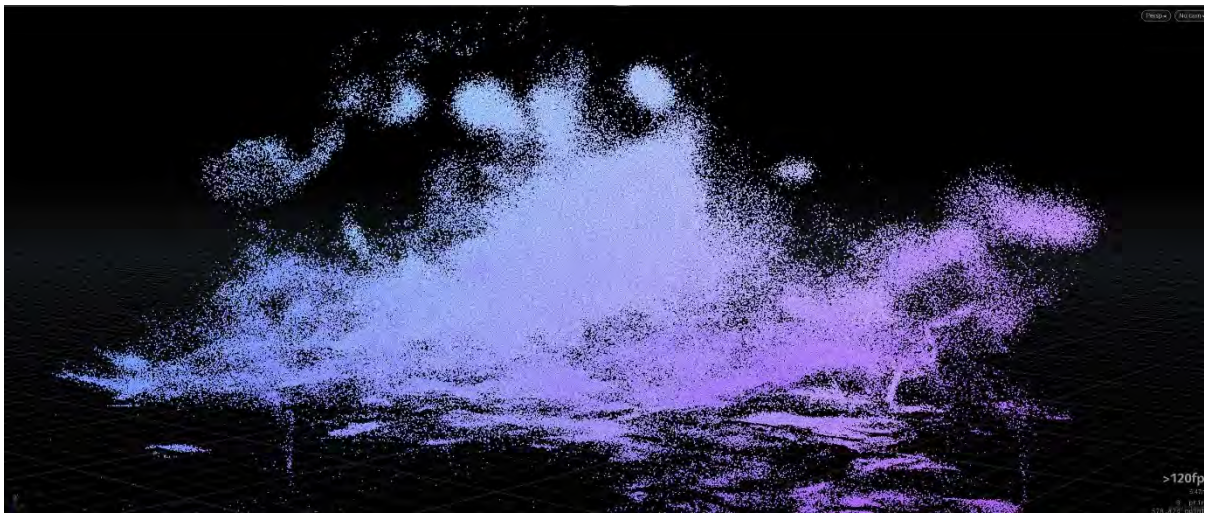


*Fig. 3.25 Houdini Particle (POP) Simulation*

The secondary FX element is created based on the results of the leading RBD simulation to add detail and visual interest to the entire scene. Houdini Particle Operators (POP) were used to generate the small fragment (rock) pieces and substrate kicked up by the impact. POP simulation type is relatively quick to complete due to only considering points in space and the forces that update their position, as shown in Fig. 3.25. However, it requires a great deal of attention in sourcing to create an exciting result.

When designing the particle system, the largest of all the challenges was configuring the output to work within Unreal's Niagara FX system. A Houdini POP simulation returns a massive point count to represent dust and small debris. On the other hand, Unreal has a

hardware limitation on the number of particles it can spawn based on draw calls[57] and Niagara cache files. Niagara cannot naturally read in a Houdini cache file, but SideFX has developed a Niagara plug-in for Unreal to alleviate the problem.[58] However, of critical importance is that the SideFX Houdini Niagara plug-in release number must match the version of Unreal. Plug-in version 4.26 will only work with Unreal 4.26 and not with 4.26.2/4.27. (Installing the plug-in requires following the steps outlined by SideFX in their instructional video.[59] SideFX's instructions are also available, written out in Appendix C.1 Installing Unreal Houdini Niagara Plug-In.)

Next, a particle simulation was designed that will look good and work with the plug-in. Designing for the plug-in requires a compromise to concede the point count of the simulation for the frame rate inside the engine. Compared to Houdini, which uses a .bgeo.sc file format the Niagara plug-in uses a .hbjson or .hjson file format.[60] A simulation source was created on the static model with the before mentioned limitations in mind. Please reference Appendix C.2 Define Particle Source Base, to read the steps for creating the particle source base; the result can be seen in Fig. 3.26.

---

[57] "Profiling Common Trouble Areas with Particle Systems," *Unreal Engine Docs*, ver. 4.26, last updated 2021, accessed July 31, 2021, https://docs.unrealengine.com/4.26/en-US/RenderingAndGraphics/ParticleSystems/Optimization/Results/.
[58] "Houdini Niagara Plugin – UE4.26," *Sideeffects Github*, ver. 4.26, last updated January 22, 2021, accessed April 20, 2021, https://github.com/sideeffects/HoudiniNiagara/releases.
[59] Mike Lyndon, "Houdini Niagara Installation," SideFX Houdini, April 29, 2020, video, 6:45, accessed April 20, 2021, https://www.sidefx.com/tutorials/houdini-to-ue4s-niagara/.
[60] Mike Lyndon, "Houdini to Niagara Workflow," SideFX Houdini, April 29, 2020, video, 8:47, accessed April 20, 2021, https://www.sidefx.com/tutorials/houdini-to-ue4s-niagara/.

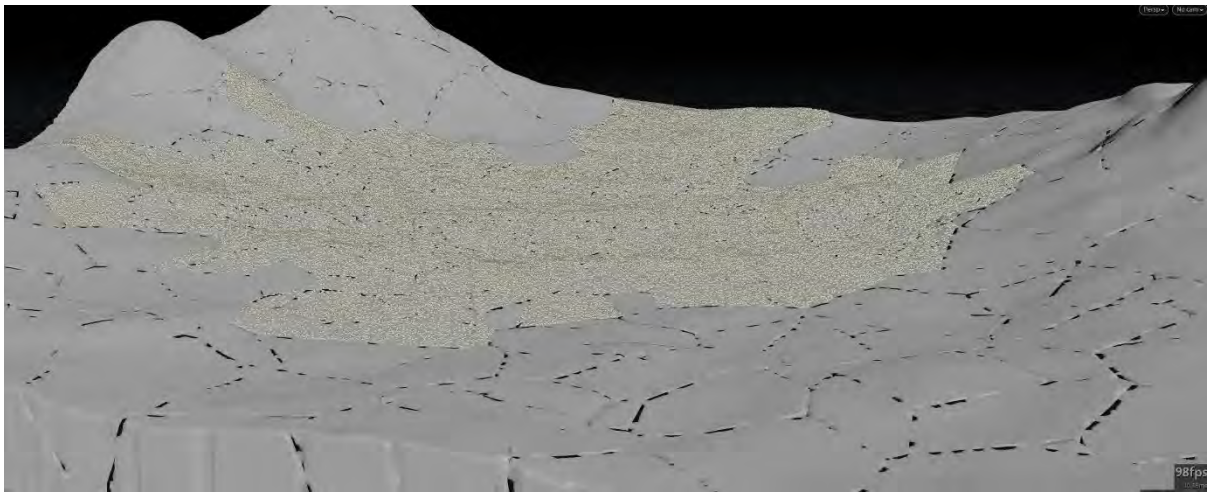



*Fig. 3.26 Particle Simulation Source Prep*

Once the static mesh and points are created, the next step is to add animation and begin source designing. Without the RBD simulations animation, the particle source would only be viable at the beginning of the sequence. The particles act as small particulates trailing from the larger RBD pieces by applying the animation to the particle source. (The breakdown of applying the RBD animation to the POP particle sources is included in Appendix C.3 Create Animated Particle Source.)

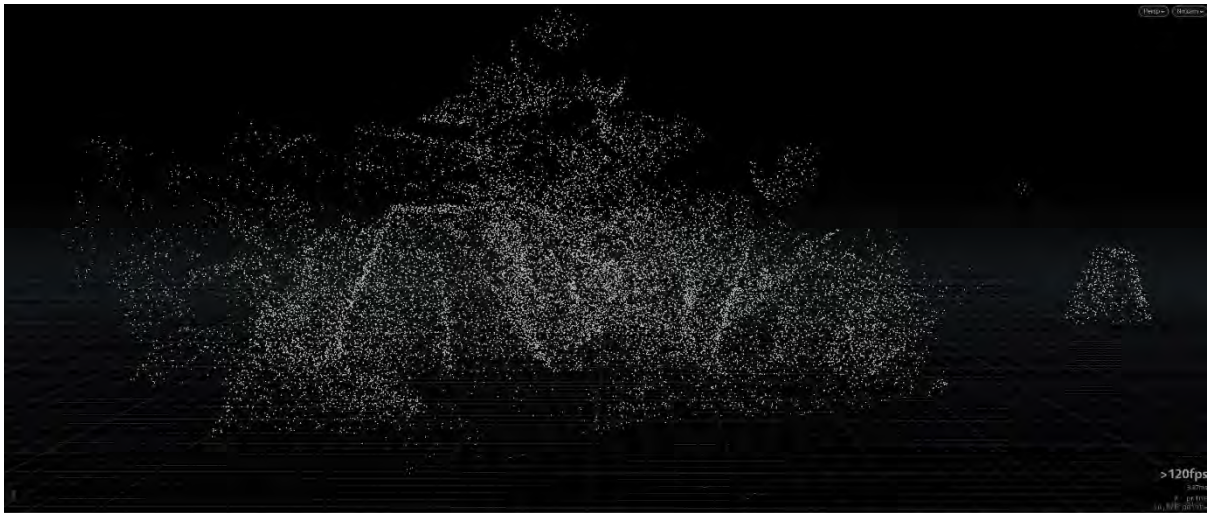



*Fig. 3.27 Particle Simulation Animated Source*

After generating the required point sources and applying the RBD simulation's animation

(Fig. 3.27) the source is ready to go through final prep before the POP simulation. The points are separated based on the spawning source to give render control after the simulation, and noise is added to their velocity for randomness. See Appendix C.4 for the written steps.

The final step of the particle sourcing is based on an Unreal problem that was presented when attempting to set up a Niagara system. When working with Houdini POPs, the simulation uses a life, age system to determine if or when a particle should die and be removed from the system. Unreal also has a life age system, but the two do not naturally recognize one another; therefore, Unreal will not kill a Houdini particle when its life attribute elapses. The Houdini Niagara plug-in works on the premise that each update (cycle or tick) of the Niagara system will read the Houdini point cache using a particles NID and update the necessary information, including position and velocity. Unfortunately, if a particle gets killed by Houdini prior to Unreal's removal, Niagara retains its last sampled velocity, causing it to drift off infinitely. When a particle fails to be removed in Niagara and drifts off aimlessly, new particles continue to be spawned by the point cache, causing a large influx of points. The considerable accumulation of points, near 900,000, inside Niagara can be seen in Fig. 3.28.

A multi-layer sprite method was discarded after attempting to render out the simulation within Houdini and use the renders as animated textures/atlases returned a flat image-like effect that did very little in the way of visual appeal. The chosen solution was to use the Niagara plug-in's capability of sampling Houdini point attributes inside Unreal and forcefully remove the points from Niagara. For the chosen method to be effective, the attribute used to kill the particles must be declared prior to the Houdini particle simulation.

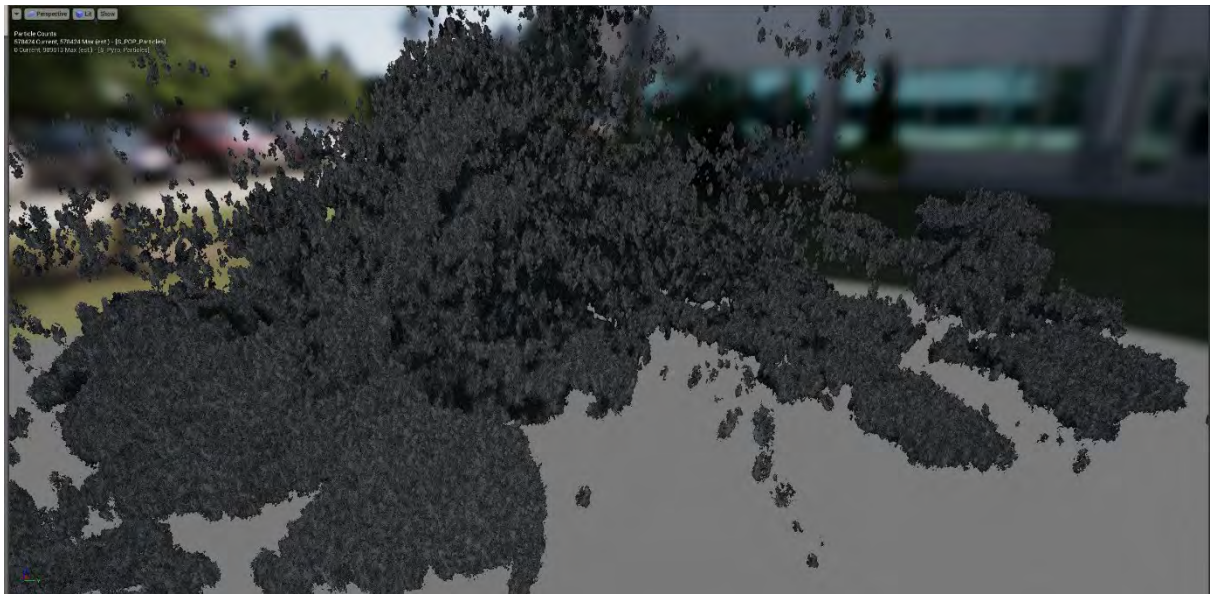
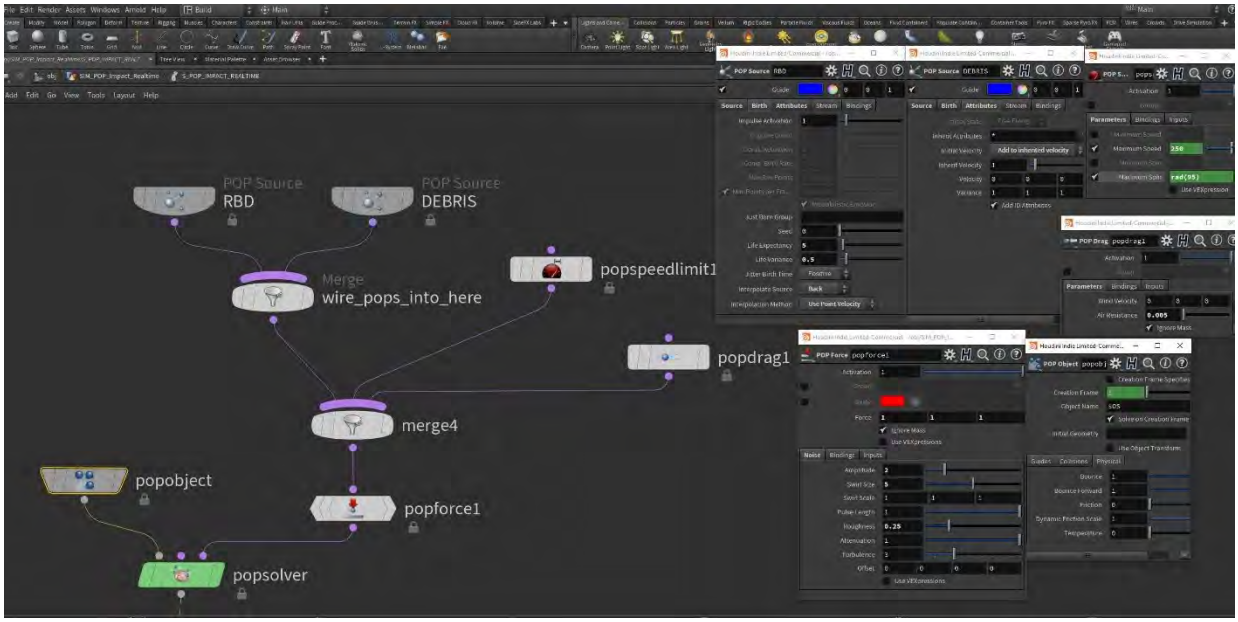


*Fig. 3.28 Niagara Particle Kill Error*

The POP network follows the same data flow structure outlined in *section 4.2, Rigid Body Dynamics*. The DOP network breakdown can be found in Appendix C.5 RBD Simulation DOP Network Point Sourcing and Solver and Fig. 3.29.




The POP simulation collisions include the original heightfield and the new RBD simulation

converted to a collision object. The DOP network method is referenced in Appendix C.6 POP Simulation DOP Network Collisions and Fig. 3.30.

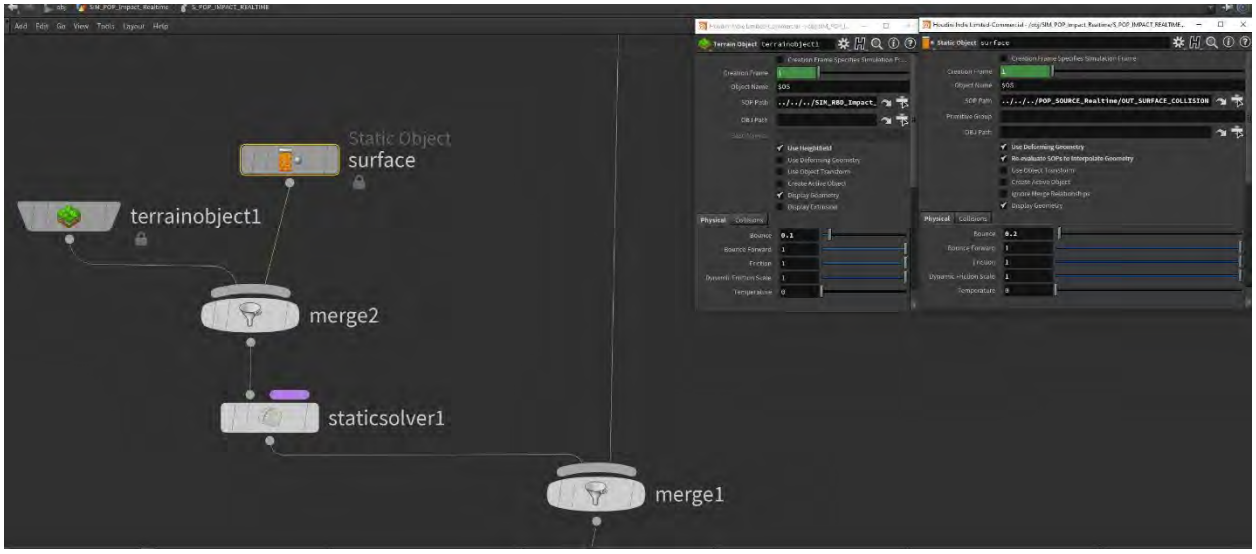



*Fig. 3.30 Houdini POP Simulation Collision Objects*

The setup of the POP simulation forces can be seen in Fig. 3.31 or read in Appendix C.7 POP Simulation DOP Network Forces.


*Fig. 3.31 Houdini POP Simulation Gravity and Output*

The output from the POP simulation is cached as a bgeo.sc file. All that remains in Houdini is the final cleanup and export of the point cache to Unreal.

The primary process for exporting the point cache into Unreal is updating the previously declared @dead attribute to reflect each point at the appropriate time accurately. A SOP Solver handled two design problems with the @dead attribute method. The challenge for the

created @dead attribute is the necessity to update over time and be attached to a specific point in a scene with a fluctuating point count. The SOP Solver node allows access for comparison between the geometry of the current and previous frame. The dead solver and particle culling are available in Appendix C.8 POP Caching and Dead Culling.

The result of the SOP Solver can be seen in Fig. 3.32. On the left, all particles, 578,424, that have a @dead attribute with a value of one, are colored blue. On the right, any particles added to the “remove” point group have been deleted, leaving 6,889.

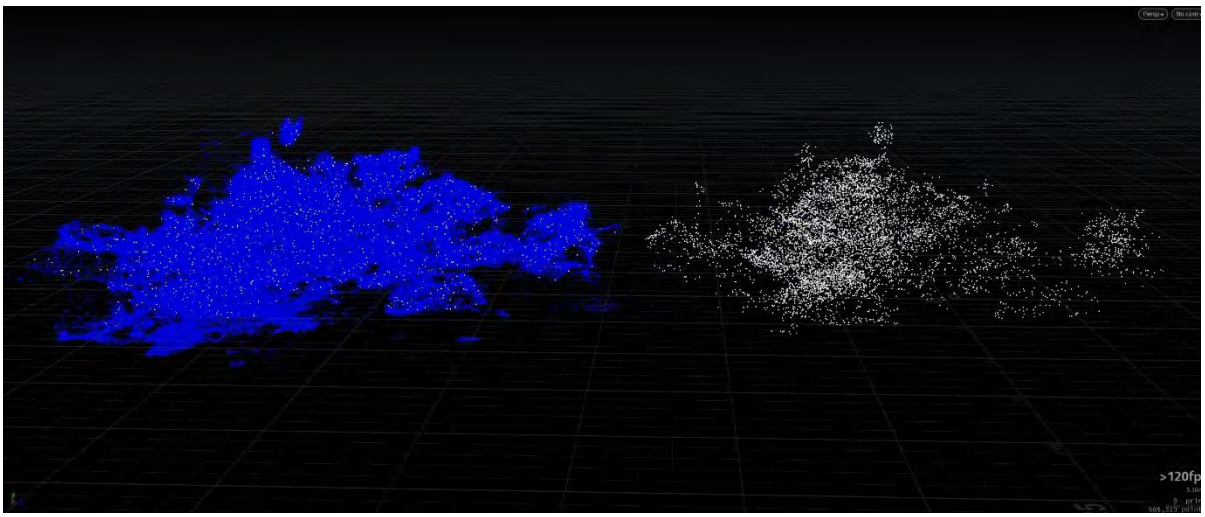


*Fig. 3.32 Houdini Particle Simulation Dead Solver Output*

Once the @dead attribute was updating correctly on the points, they were ready to be written out of Houdini. The writing process was handled by a Houdini Labs node called Labs Niagara ROP. However, there are two critical problems with the Niagara ROP node that must be addressed. First, there is a bug in the python three release of Houdini 18.5.462 and 18.5.495, causing the Labs Niagara ROP to fail to export the .hbjson and .hjson files. The error was resolved by downloading the python two release of Houdini. The second problem is that the Niagara ROP does not write out correctly unless Houdini is in Auto-Update mode, resulting in a 1kb file. The final POP export settings are present in Appendix C.9 POP Unreal

Export.

A detailed description of the Unreal particle import and system creation is presented in Appendix C.10 POP Unreal Import.

The entire Niagara POP system can be seen in Fig. 3.33, and the complete system settings can be found in Appendix C.11 Unreal Niagara System.

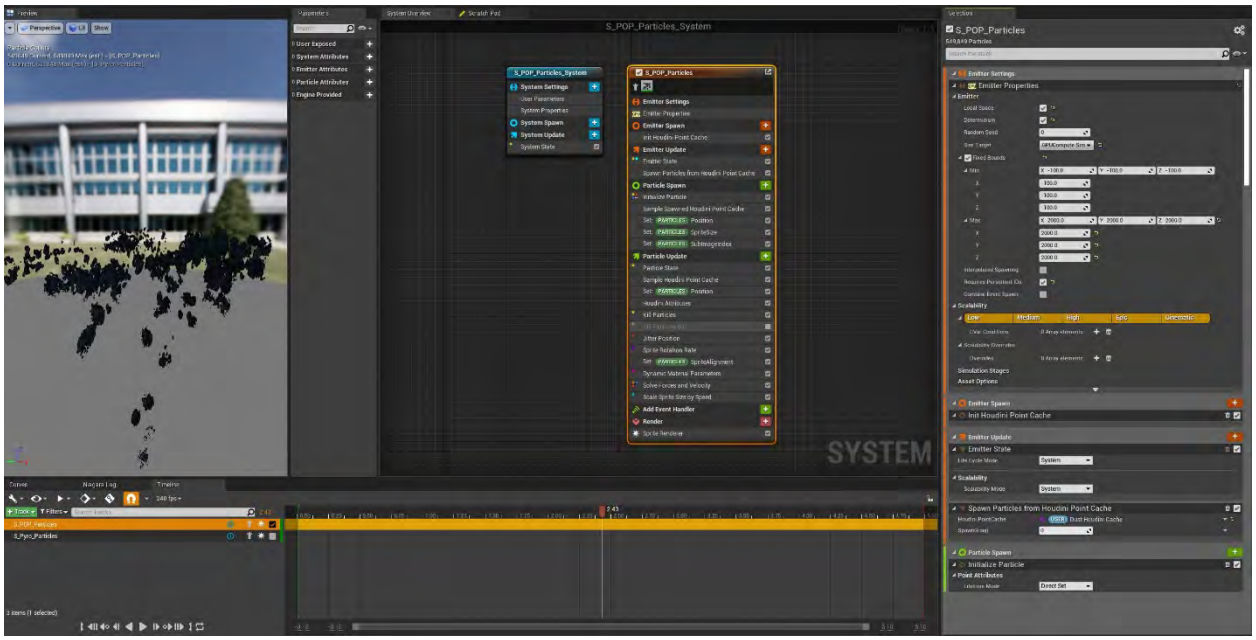



*Fig. 3.33 Unreal Niagara POP System Setup*

Gaining access to the Houdini attributes that the plug-in does not automatically provide presented a particular set of challenges. A Niagara Module Script must be created to reference the point cache and promote the attributes to Niagara. The Niagara Module Script can be used for other uses outside of reading Houdini attributes, but in this instance, runs on the use of the plug-in, creating a Niagara ID (NID) number used to sample the desired attribute. A walkthrough video from SideFX goes through the following steps for sampling

custom attributes into Niagara.[61] The module script setup used in the case study is explained in Appendix C.12 Houdini Attributes inside Unreal using a Niagara Module Script, and shown in Fig. 3.34.

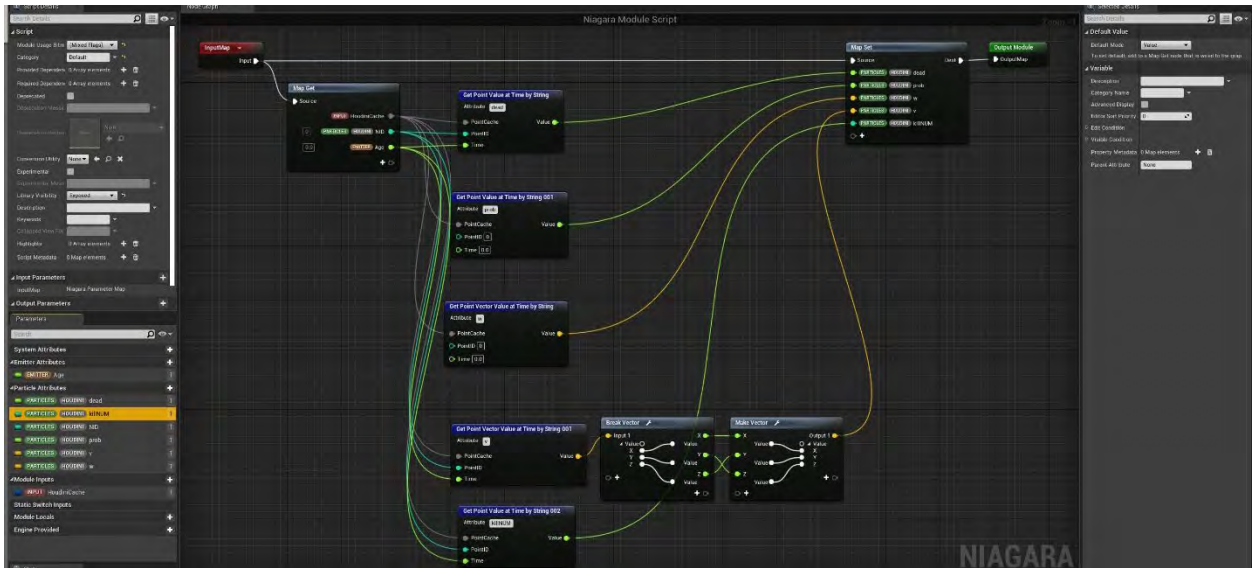



*Fig. 3.34 Unreal POP Simulation, Custom Attributes Niagara Module Script*

With access to the Houdini attributes within Niagara, the simulation can cull the particles using the @dead attribute. See Appendix C.13 for the written guide for using the @dead attribute to cull the particles.

A Jitter Position, Sprite Rotation Rate, and Dynamic Material Parameters are used to add variety to each particle and improve the simulation's visual look. Breakdown is available in Appendix C.14 Adding Particle Variety Inside Niagara.

The render method used was with sprites. A single particle inside Niagara could depict a large point count and include small rock fragments using sprites. However, a problem arises when applying the material to the Niagara particles. Since the sprite render system inside

[61] Mike Lyndon, "Sampling Custom Attributes," SideFX Houdini, April 29, 2020, video, 10:08, accessed April 20, 2021, https://www.sidefx.com/tutorials/houdini-to-ue4s-niagara/.

Unreal only accepts a single material, all particles are rendered with identical sprites, creating a repetitive result. An animation technique using SubUV's and a sprite atlas was levied to handle the repetitive problem. Each particle randomly selects and sets itself to a specific SubUV coordinate at spawn, which is then used inside the sprite atlas (Fig. 3.35) to choose a rock fragment arrangement.[62] Follow the guide in Appendix C.15 to set up the Niagara Sprite Render.

The atlas is made inside Houdini using similar RBD and POP sourcing techniques to break apart simple rock geometry and scatter points inside a volume. The scatter/fracture values are randomized to create different layouts in each frame. The rendered frames are combined together using the Houdini IMG context and are unpremultiplied by their alpha to prevent edge bleeding inside Unreal. Finally, the new image is written out as the needed atlases texture.

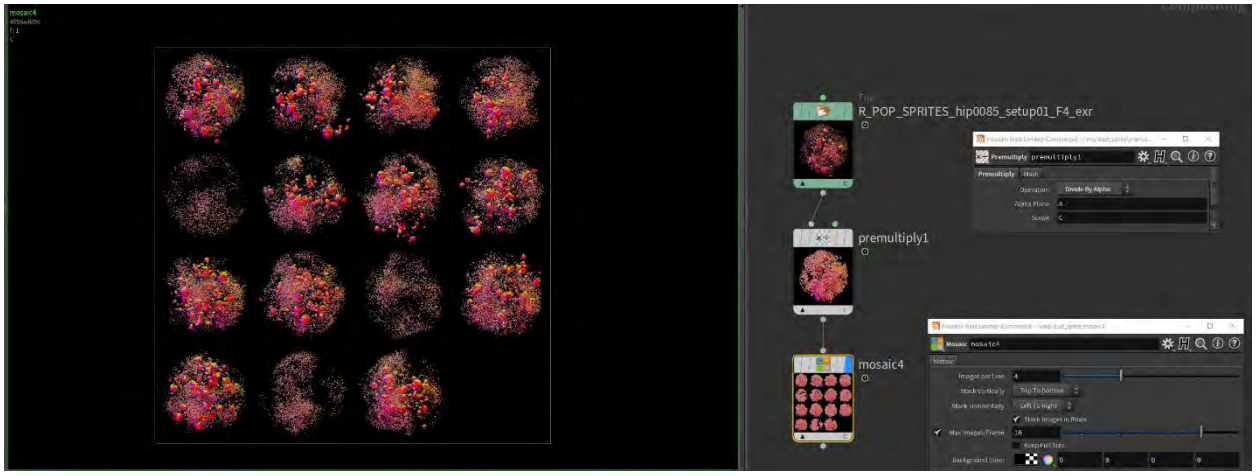



*Fig. 3.35 Houdini Unreal POP Rock Fragment Sprite Atlas*

After gathering all the necessary elements and setting up the Niagara system, the last

[62] "Unreal Niagara: Sub UV Index Particles," AndrewWeirArt, December 30, 2018, video, 13:20, accessed April 25, 2021, https://youtu.be/Hh4wxRocnQc.

component is the sprite material. The material ends up doing a lot of the detail work for the POP simulation besides the particle movement and any optimization controls. Because a sprite is a flat texture compared to actual geometry, the lighting by default is static and not able to move or react to lighting changes inside Unreal. A cheap solution is to render the sprites out of Houdini using a six-dimensional (6D) lighting setup, as used in Rebelway's "Realtime FX for Games and Cinematics" Course.[63] The sprite elements are lit from the top, right, and front (x,y,z). Each light receives a value of one in either the R, G, or B channel, and thus when brought into Unreal, the lighting can be split apart by color and live updated in the engine.

Once the lighting and color of the particles are complete, the particle opacity needs to be controlled and optimized. The opacity is primarily driven by the original particle atlas' alpha, which is then scaled by the particle's current distance from the camera. The final material network is available in Fig. 3.36, and the resulting particle simulation in Fig. 3.37.

[63] "Realtime FX for Games & Cinematics," *Rebelway*, accessed February 18, 2021, https://www.rebelway.net/realtime-fx-for-games-and-cinematics.

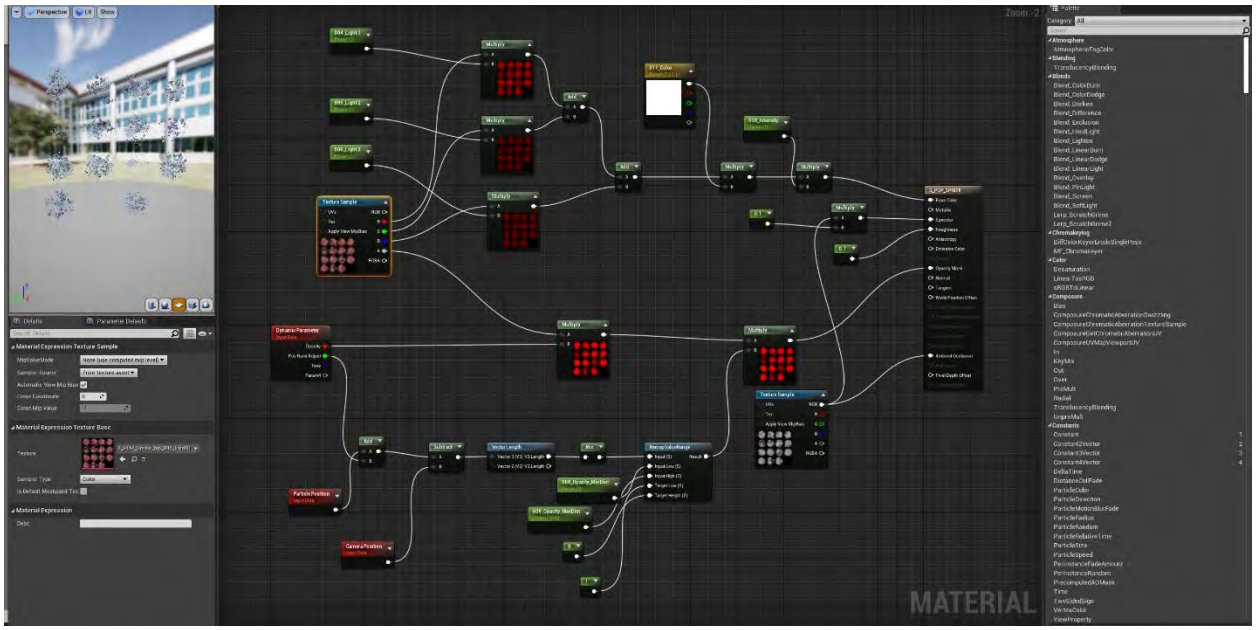



*Fig. 3.36 Unreal POP Sprite Material Network*

*Fig. 3.37 Unreal POP System*

### 3.2.3 [Methodology] Real-time Volumes (Pyro) Smoke and Dust

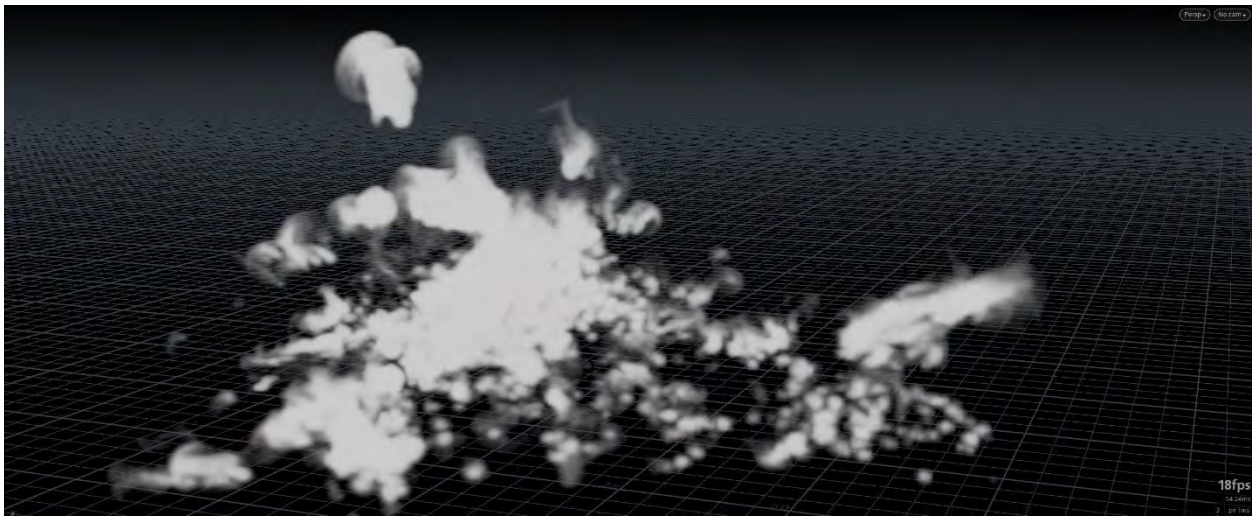



*Fig. 3.38 Houdini Particle Advect Volume (Pyro) Simulation*

The final simulation type utilized in the FX design is volumes (pyro) to generate many fine particles or dust clouds that the impact would create. The most considerable benefit of volumes is that they can represent an enormous amount of small particulates with greater ease and not require simulation of millions, if not billions, of points, thus reducing the strain upon the POP simulation. While volumes do come with a number of benefits, they also come at a cost. Currently, volumes are the second slowest to simulate inside Houdini, outmatched only by FLIP fluids, which utilizes volumes in its computation.

The difficulty working with volumes and Unreal was constructing a workflow that does not utilize voxels and instead could achieve a decent pseudo volume inside the real-time engine. The standard DCC method of representing volumes uses a volume-pixel (voxel) shown in the Pyro simulation of Fig. 3.38, a three-dimensional pixel, to solve depth, pressure, and transmittance. Unreal, as of 4.26.1, has no fully established system to read and display voxels, requiring the use of workaround methods.

There are two systems that could provide the desired results inside Unreal. A volumetric

ray marcher outlined by Ryan Brucks[64], or a particle-driven system. The ray marcher described by Ryan Brucks attempts to recreate a voxel system inside Unreal. The premise is that layering multiple parallel planes along the Z and X-axis[65] allows objects to interact with the volume by sampling, masking, shadowing, and blurring in a three-dimensional space. Thus, implementing a ray marcher insinuates that every engine tick has to recalculate very complex opaque materials while accounting for interacting objects, making it inappropriate for use in conjecture with other large-scale FX. The second is based upon work by Ian Harvey[66] to use Houdini particles driven by a volume simulation and then brought into Niagara and use a volume sprite texture. The first step is to create the volume source used in the volume simulation to drive the motion of the particles in Unreal; reference Appendix D.1 Creating Volumes (Pyro) Movement Simulation Source for complete details.

When converted to an SDF volume, the points lose all the assigned attributes. Without the particle simulation velocity, the volumes spawned at a given position began to disperse evenly with no regard to the original particle source direction. An attribute transfer returns the necessary attributes to the volume source. The steps to restore the particle attributes are detailed in Appendix D.2 Restoring Point Attributes.

---

[64] Ryan Brucks, "Creating a Volumetric Ray Marcher," *Shader Bits*, November 16, 2016, accessed June 12, 2021, https://shaderbits.com/blog/creating-volumetric-ray-marcher.
[65] Ryan Brucks, "Authoring Pseudo Volume Textures," *Shader Bits*, October 18, 2016, accessed June 12, 2021, https://shaderbits.com/blog/authoring-pseudo-volume-textures.
[66] Ian Harvey, "The Mill SDGM Real-Time FX Switching Gears to Niagara," *Ian Harvey*, updated March 9, 2020, accessed May 24, 2021, https://www.ianharveyvfx.com/post-pme0g/the-mill-sdgm-real-time-fx-switching-gears-to-niagara.

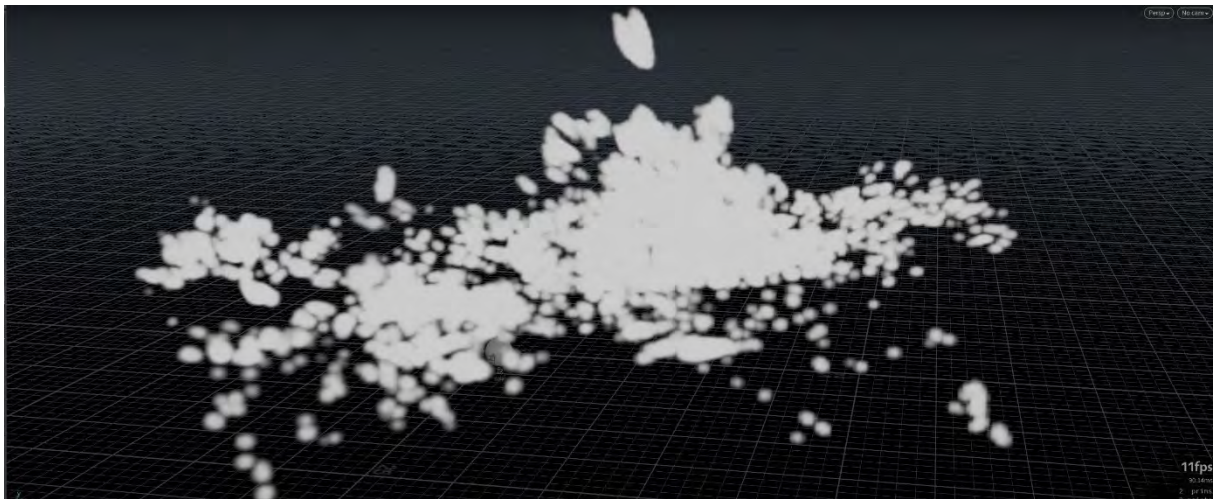


*Fig. 3.39 Houdini Volumes (Pyro) Simulation Rasterized Source*

The volume source seen in Fig. 3.39 is now ready to go through any final preparations for the pyro simulation. The addition of noise to the velocity, seen in Appendix D.3 Adding Velocity Noise, is used to prevent the volume from following the particle and RBD simulations movement.

Houdini's Sparse Smoke Solver was chosen due to its fast simulation time and ability to handle fast-moving sources. However, a question presented itself while designing the system. Standard Houdini volumes simulations require a fair amount of detail (number of voxels) to simulate an accurate and visually appealing result. By comparison, this simulation's only purpose would be to impart the motion of a volume onto the later POP simulation; therefore, there is no need for a high-resolution simulation since the only usable portion will be the motion. The Pyro network follows the same data flow structure outlined in *section 4.2, Rigid Body Dynamics*.

The volume DOP network is focused on importing and spawning the necessary voxel data for the sparse smoke solver. The network details are visible in Fig. 3.40 or outlined in Appendix D.4 Volume Simulation DOP Network Sourcing and Solver.

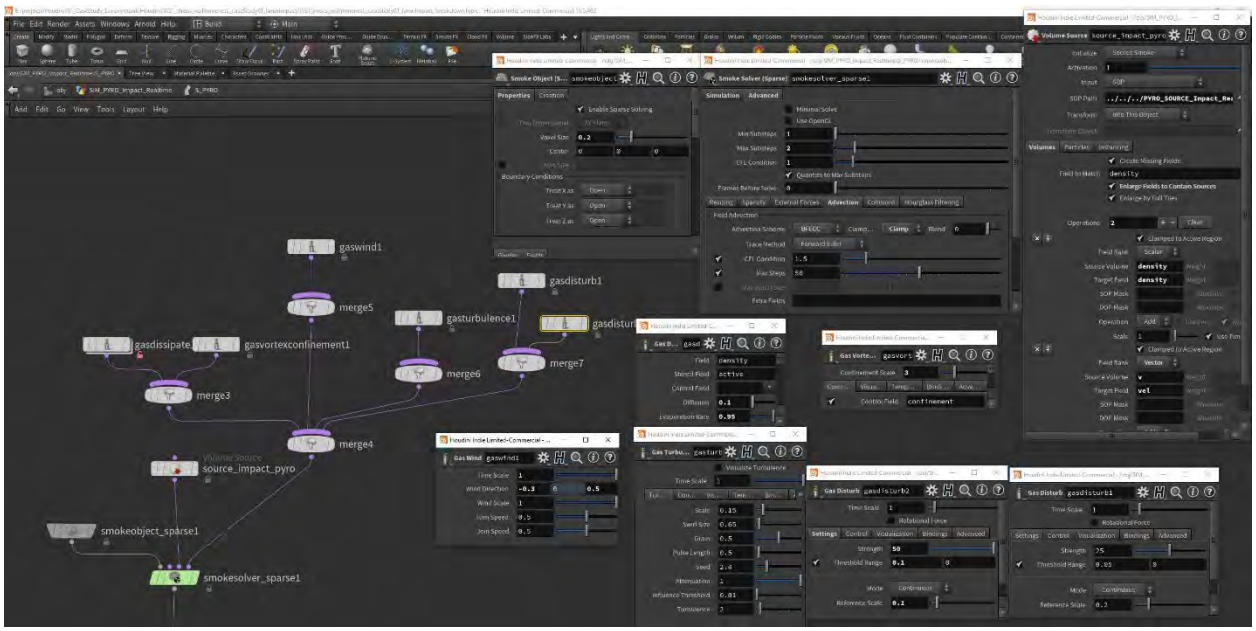



*Fig. 3.40 Houdini Volumes (Pyro) Simulation Import Source and Create Solver*

The DOP network must handle interaction with collision geometry. The collision objects include the original heightfield and the RBD simulation. The DOP network collision method is referenced in Appendix D.5 Volume Simulation DOP Network Collisions and Fig. 3.41.

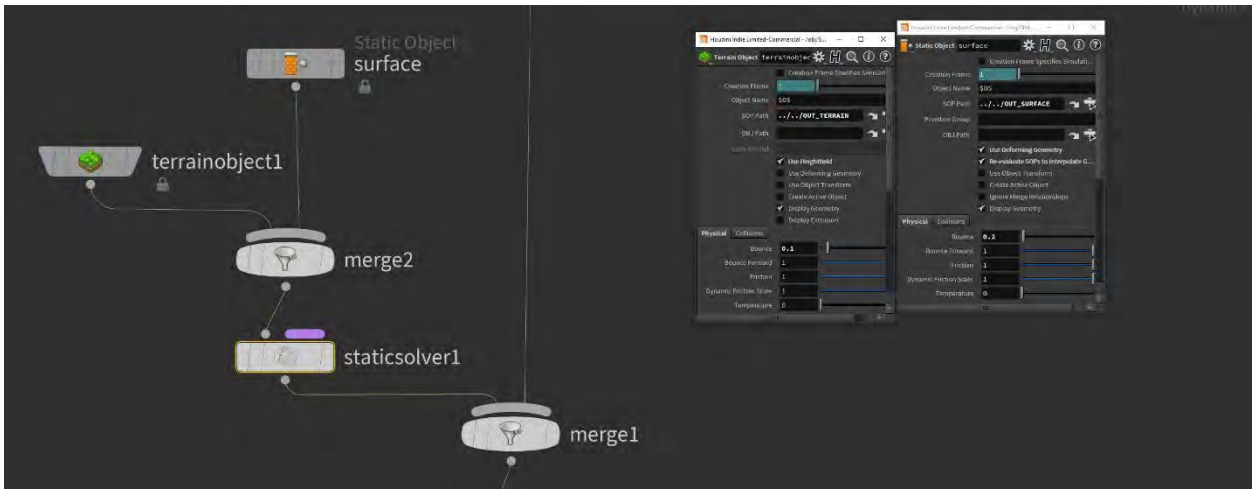



*Fig. 3.41 Houdini Volumes (Pyro) Simulation Second Level Collision Objects*

The DOP network can be seen in Fig. 3.42, or a written breakdown of the forces is described in Appendix D.6 Volume Simulation DOP Network Forces.


*Fig. 3.42 Houdini Volumes (Pyro) Simulation Gravity and Output*

While the simulation is being run, the result is removed from the DOP network, run through a Pyro Post Process SOP, and cached out a 16-bit bgeo.sc file. This output was used as an advection force inside a particle simulation and a final post-processing check for those particles. The particle simulation for the volume sprites is identical to that of the primary POP simulation done in *section 4.3, Particle Operators*, with the addition of having the particle's behavior driven by the previously created volume. A snapshot of the new particle network can be seen in Fig. 3.43, and a complete explanation in Appendix D.7 Volume POP Simulation Setup.

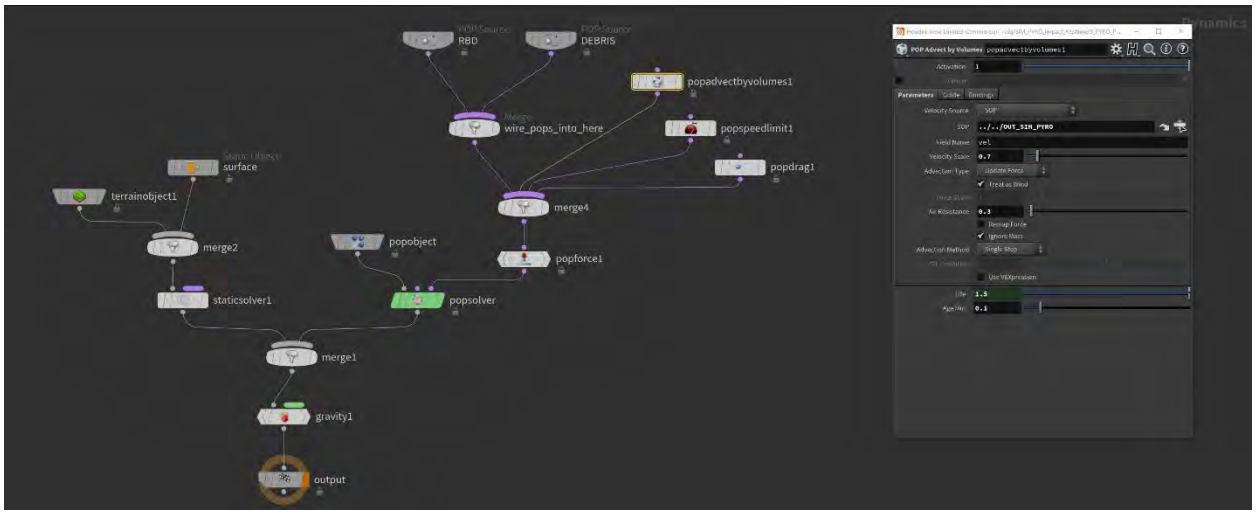



*Fig. 3.43 Houdini Volumes POP Simulation Volume Advection Method*

Once the new volume POP simulation is run, the points are cached out for final processing. The particles are passed through the same SOP Solver to create the @dead

attribute used in *section 4.3, Particle Operators*, and the points are culled. The culling of points is critical due to an issue inside Unreal caused by a large amount of semi-translucent materials overlapping. The remaining points are then checked against the original volume (pyro) simulations @density volume to ensure they exist within the simulation and given a color @Cd attribute to be used inside Unreal. The finalized points are then written out using the Labs Niagara ROP node as a .hbjson file, seen in Fig. 3.44. Reference Appendix D.8 for the written guide.

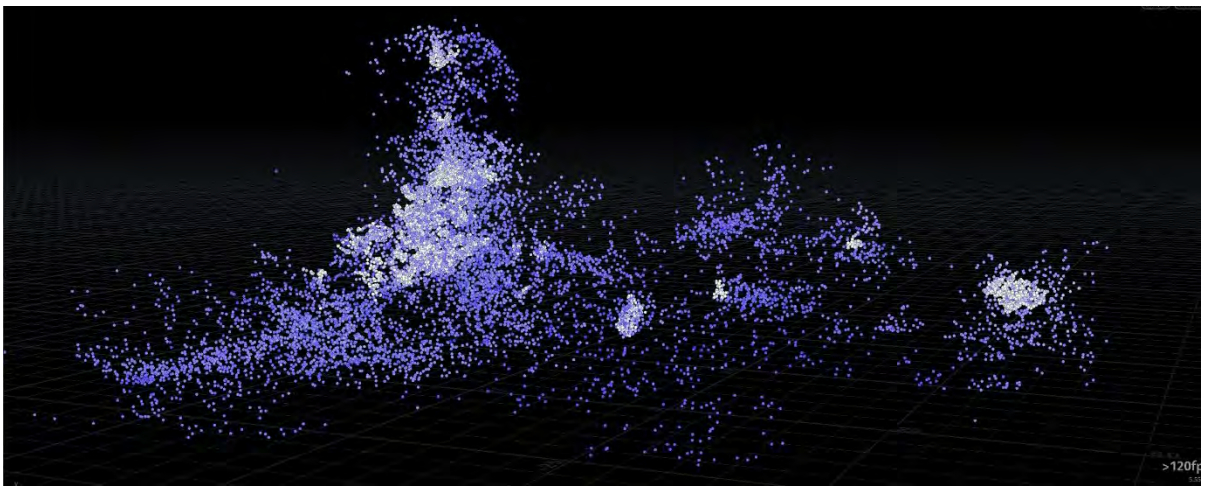


*Fig. 3.44 Houdini Volumes POP Simulation Output*

The simulation can be brought into Unreal with a usable point cache and placed inside a Niagara emitter. Unreal now has the movement of the volume's simulation. In gaining the volume behavior on particles, the system has now lost the visual element of a wispy volume. Similar to other elements in Unreal, the detail of the volume will be regained through a material. The Niagara particle system settings are detailed in Appendix D.9 Volume POP Unreal Niagara Setup.

Compared to the previous particle system, the particles in the volume simulation represent a section of a larger whole simulation instead of individual rock and dust clusters. The difference in elements requires that the sprites be more prominent in size to help

recreate the large volume. The resizing of the volume sprites is presented in Appendix D.10 Volume POP Unreal Niagara Setup.

Once particles have the appropriate position and scale, the final step is to set up alignment. Particle alignment is described in Appendix D.11 Volume POP Unreal Sprite Alignment. The dynamic parameters are detailed in Appendix D.12 Volume POP Unreal Dynamic Material Parameters.

The correct way to create a float type dynamic input script is to click the drop-down arrow of the parameter and select “New Scratch Dynamic Input.” The dynamic input’s function can be created/edited to give the desired options inside the Niagara Scratch Pad. See Fig. 3.45 for the float remap values script. When working with the dynamic material inputs, specifically opacity, a problem became apparent. When wanting to remap the output of a curve from zero to one to a new range, there is no remap dynamic script built into Niagara.

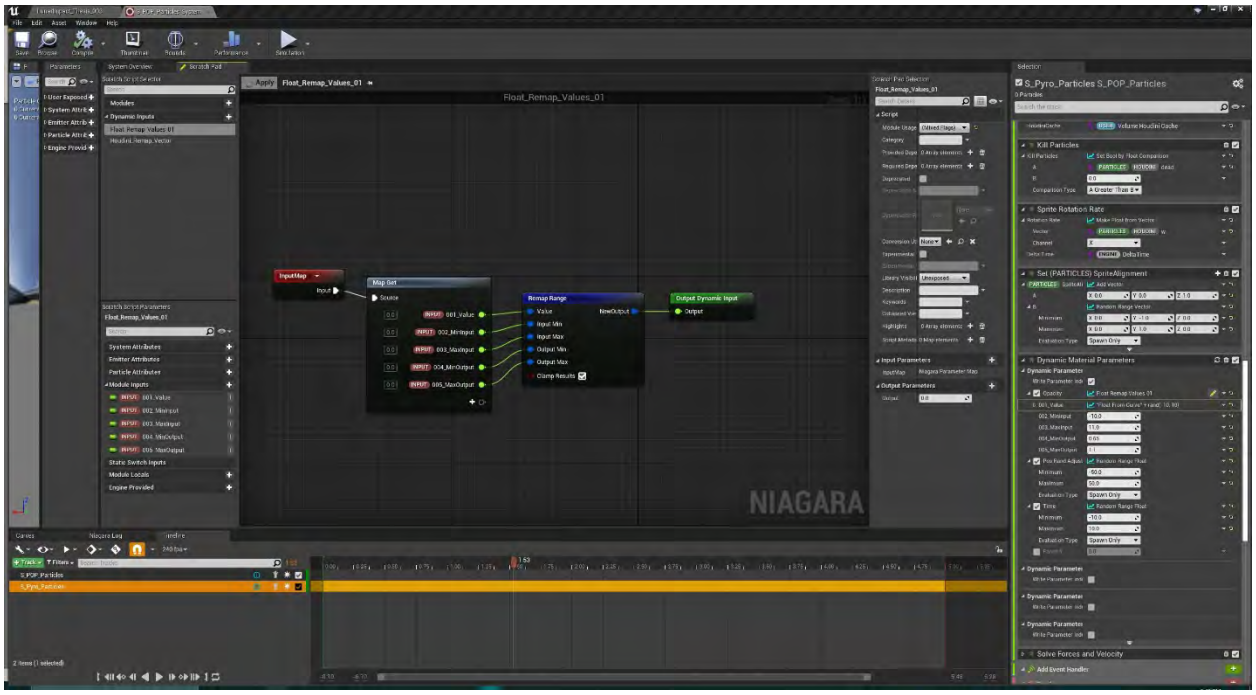



*Fig. 3.45 Niagara Scratch Pad Dynamic Input, Float Remap Values*

Similar to the POP simulation, a sprite render method for the volume simulation was used. However, a problem dealing with the material arose. The material looked uncanny when the volume sprites moved with the motion of a volume, but the sprites themselves did not change and stayed a static texture. A secondary volumes (pyro) simulation created an animated texture sheet for the sprites to handle the problem best. The volume sprite simulation setup can be found in Appendix D.13 Volume Sprite Simulation.

When the simulation was completed, the result was rendered out as a sprite texture using a 6D lighting setup for better control inside Unreal, as mentioned in *section 4.3, Particle Operators*. Unfortunately, when testing the animated texture, an issue arose within Unreal. Inside Niagara, some particles were dying quicker or slower than the volume sprite could loop through the animation. The solution to the volume sprite was to create a loopable volume texture outlined by Urban Bradesko.[67] The atlas created for the animated volume textures in Unreal can be seen in Fig. 3.46.

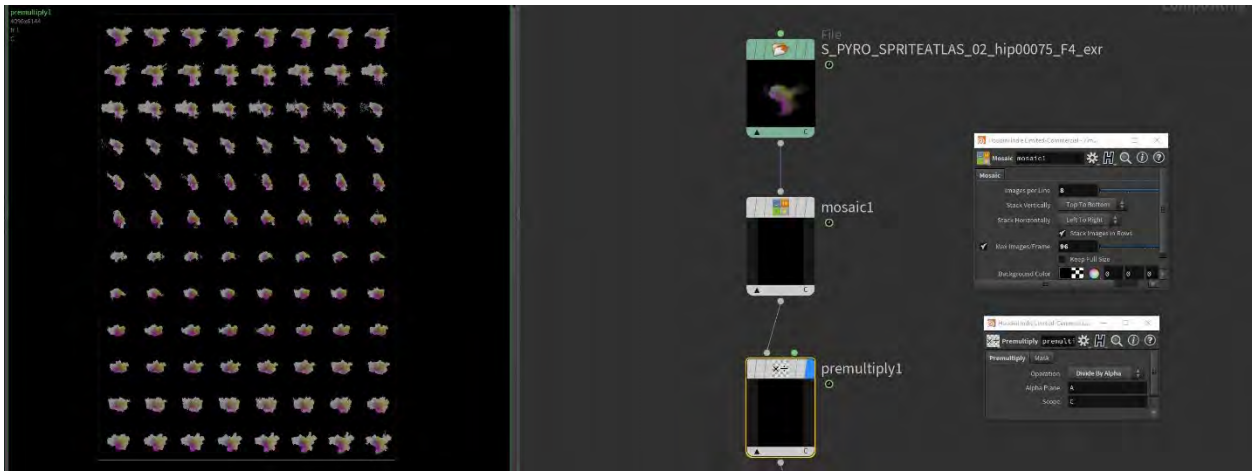



*Fig. 3.46 Houdini Volume Sprite Atlas*

A loopable simulation texture runs off the principle that the beginning and end frames of

[67] "Creating Looping Realtime FX Using Houdini | Free Webinar Replay," Rebelway, January 28, 2021, video, 1:17:31, accessed June 18, 2021, https://youtu.be/0NgNQlpmPlQ.

the sequence are connected or the same. The way to achieve this is by time shifting the volume simulation's midpoint to the first frame, allowing the first and last frame to be hidden in the middle of the animation. The full Houdini setup can be seen in Fig. 3.47 or Appendix D.14 Loopable Simulation Texture.

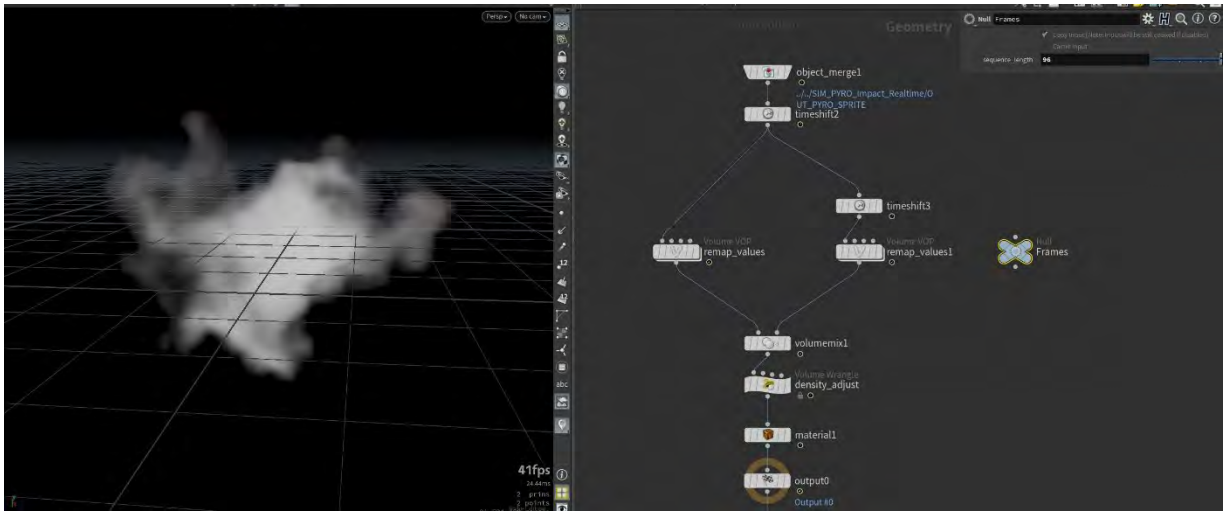



*Fig. 3.47 Houdini Volumes Loopable Texture Setup*

Once the loopable volume is ready, the result is rendered using the 6D lighting setup. Initially, the sprite was rendered at 2048x2048 since it would be viewed on a large scale inside Unreal. Unfortunately, the render scale caused issues inside Unreal. Unreal has a max texture import size of 16,384 x 16,384, which a ninety-six frame texture atlas rendered at 2048 x 2048 easily exceeds. The import issue was resolved by reducing the render size of the sprite, thus shrinking the atlas size respectively. However, with the atlas successfully brought into Unreal, the volume sprites caused a hit to performance. It was decided to render each frame at 512 x 512, which would provide enough detail when viewed but reduce strain on Unreal.

The most critical component of the volume workflow is the Unreal material. The material will have control over multiple aspects of the volume in an attempt to regain detail. Since

the volume sprite had to be written out as a static image atlas, the sprites would display the entire atlas by default. A Flipbook animation node was used to play the image sequence one subuv at a time. After the texture is animated correctly, the next step is to utilize the 6D lighting and create a smarter material. Instead of having an artist manually update the lighting position of the 6D lighting setup, Unreal instead will compute where the lighting would exist in relation to the sprite and choose the correct light. Once the lighting is set up, the particle's alpha is used in another smart material element to calculate the opacity of the particles. At the same time, an artist still has control over the final look of the particles. The complete material network is available in Fig. 3.48, with each subpart displayed in Fig. 3.49-52.

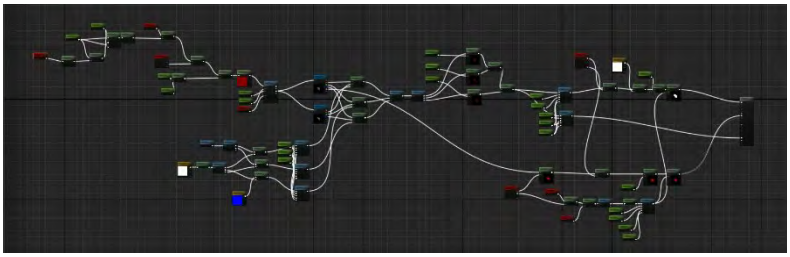


*Fig. 3.48 Unreal Volume Material*

The first portion of the Unreal volume material is the re-animation of the volume sprite and the time controls, shown in Fig. 3.49. The time control is also outlined in Appendix D.15 Unreal Material Time Component. The time control section of the material was inspired by the tutorial by Ben Cloward, "Flipbook Animation – UE4 Materials 101 – Episode 5."[68]

The random value added to the user inputted timescale adds randomness to each particle's time to ensure they do not run at equal speeds. With the new timescale prepared, the next step is to add error protection. If the artist sets the timescale to a zero value, the

[68] Ben Cloward, "Flipbook Animation – UE4 Material 101 – Episode 5," December 19, 2019, video, 16:48, https://youtu.be/ZWAF_f2aP9s.

random added value would still generate numbers above zero, thus causing some particles to run the animated texture. An “If” comparison function fixes the problem by checking to see if the inputted timescale is greater than zero and, if not, set the value to zero.

With the TimeScale now ready, the next step is to create the correct animated UVs for the texture atlas. The node used to generate the UVs is called Flipbook and requires the number of rows and columns used in the texture atlas. The Flipbook settings can be found in Appendix D.16 Unreal Flipbook Function.

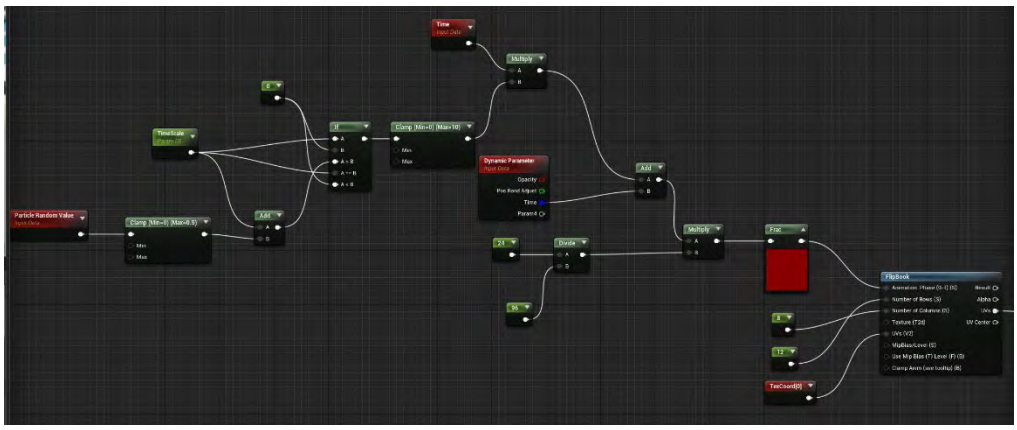



*Fig. 3.49 Unreal Volume Material, Animate Texture*

The animated UVs are then connected into the two TextureSample nodes used for the 6D lighting setup. The two TextureSamples now give the material the ability to approximate the lighting of any situation roughly. Typically, each RGB channel would be separated from the texture atlas, and a Linear-Interpolation (Lerp) would adjust between the different lighting inputs using a zero to one value. However, instead of manually updating the material each time a change has to be made to lighting, Unreal should be able to adjust automatically. The question becomes what returns a zero to one value that can be computed within an Unreal material. Unreal has a Camera Direction Vector built into the material workflow, while a light position can easily be generated from a floating-point vector

input. A dot product function can calculate the difference between the direction of two vectors and returns a value between minus one and one based on how close they are to facing the same direction. Simply remapping the output of a dot product function between zero and one can return the necessary value to lerp the 6D lighting setup. With the new setup, the user would only have to update the position of the light on the material instance, and the entire system would update correctly. The final smart lighting setup can be seen in Fig. 3.50, or reference Appendix D.17 Unreal Smart Lighting Setup.

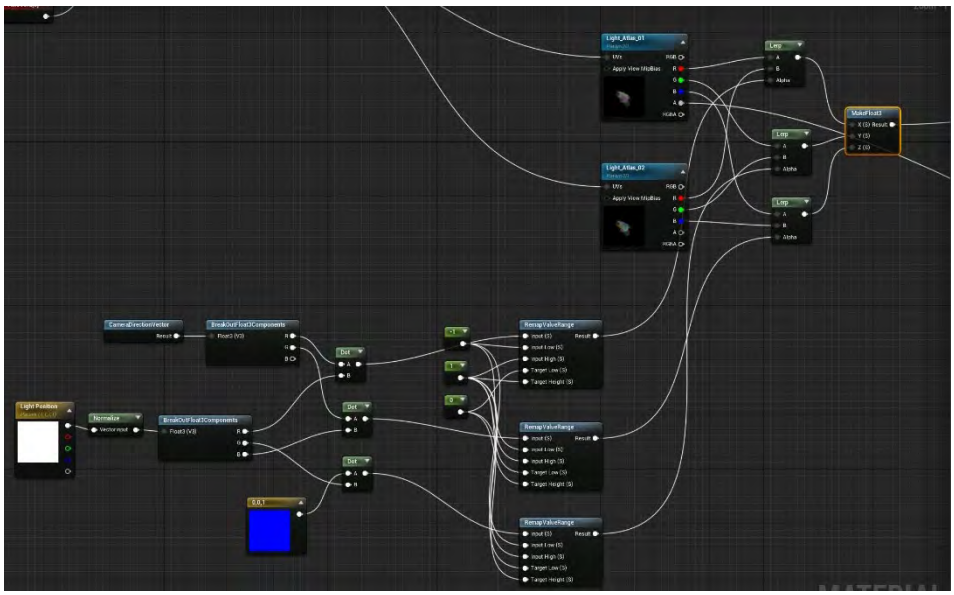


*Fig. 3.50 Unreal Volume Material, Smart Lighting*

Unreal is now capable of creating a more believable lighting setup for the volume sprites. Overall, the artist controls the 6D lighting intensity returned by Unreal, the sprite's color, and the grayscale value. The artist control setup is viewable in Fig. 3.51, with a written breakdown in Appendix D.18 Unreal Material Artist Controls.

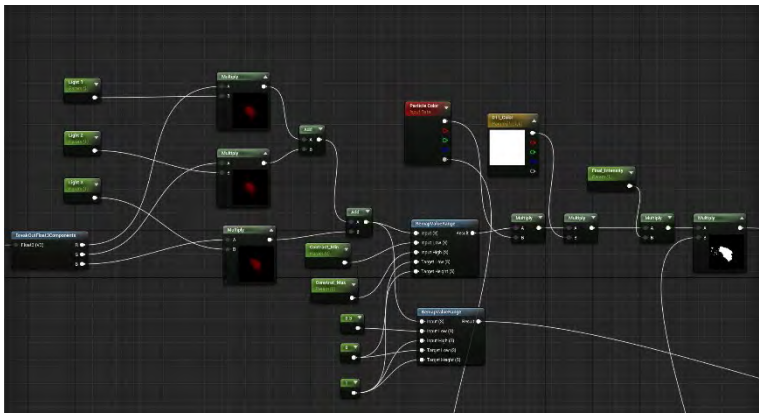


*Fig. 3.51 Unreal Volume Material, Arist Controls*

All that is left for the volume material is the creation of the opacity of the sprites. In general, the alpha generated from the Houdini sprite render would be passed to the Unreal material as the opacity mask. Unreal defaults to cull particles, specifically the Niagara Particle emitter, based on visibility and its actor bounds.[69] The culling process causes particles to exist entirely within the scene regardless of whether the camera sees them or not, until the camera culls the Niagara emitter, causing a popping effect. To best answer this problem, an auto distance opacity control is added to optimize the Unreal scene by turning the particles entirely transparent when out of range and reduce the necessity of user input. The smart opacity setup is viewable in Fig. 3.52, and the final result inside of Unreal in Fig. 3.53. View Appendix D.19 Unreal Material Smart Opacity Setup for the Unreal Opacity.

[69] "Visibility and Occlusion Culling," *Unreal Engine Docs*, ver. 4.26, last updated 2021, accessed May 9, 2021, https://docs.unrealengine.com/4.26/en-US/RenderingAndGraphics/VisibilityCulling/.

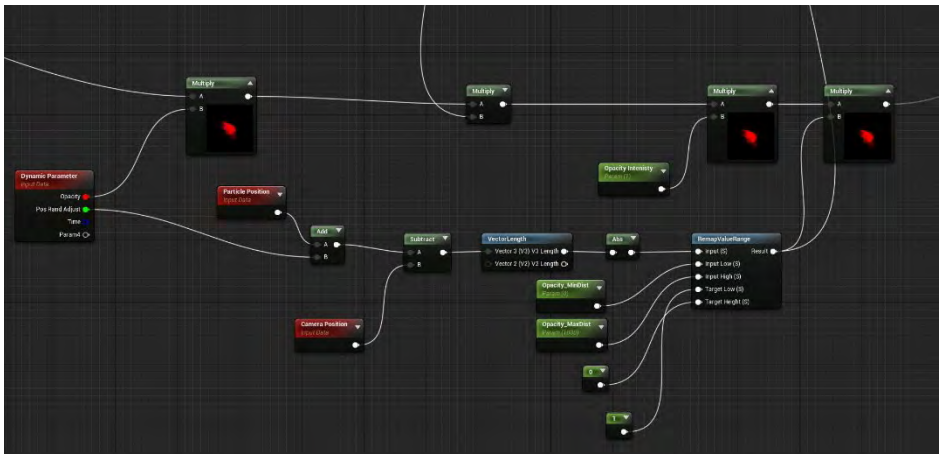



*Fig. 3.52 Unreal Volume Material, Opacity by Distance*

*Fig. 3.53 Unreal Volume Simulation with Lighting Controls*

### 3.2.4 [Methodology] Real-time Scene Setup

*Fig. 3.54 Unreal Post Processing*

After the critical FX elements and systems are created within Unreal, look development and optimization was completed. The aim is to have a scene inside Unreal with as much visual detail as possible without taxing the FPS any further than necessary. The results can be seen in Fig. 3.54.

The first way to improve the visual look of the scene past the FX themselves is to include the background base elements created earlier. The best choice for the static geometry was to export as an FBX with the @name attribute to define the hierarchy. The initial hierarchy was designed around the @name attribute being driven by the point number that spawned the building. Unfortunately, having the hierarchy based on the spawn point alone caused conflict within Unreal when attempting to apply materials to the different parts of the station since the individual material pieces were fused together under one @name. The @name attribute was generated using two different components compared to the previous one to solve the namespace and material issue. The two used components were the applied material and the building name. The naming process used from Houdini to

Unreal can be found in Appendix E.1 Houdini Base FBX Export along with Fig. 3.64.

Importing the static meshes and applying the various materials caused a new set of problems. Compared to the previous naming method, which resulted in a total of seven objects, the new @name returns five-hundred fifty-six individual models requiring twenty-three different materials. The new static mesh/material count put a significant strain upon Unreal. The easiest way to address this newfound problem is to use the Merge Actor Tool inside Unreal, outlined by Sam Deiter in the *Building Better Pipelines, LODS & Merging Static Meshes* course on Unreal's online learning portal.[70] The resulting static mesh count was reduced from the original five-hundred fifty-six using twenty-three materials to nineteen meshes, each using one material. The final issue to address was the fact that the model did not contain any Level of Detail (LODs) built-in and the lightmap resolution scale. Each of the nineteen static meshes uses Unreal's built-in LOD generator to create four LODs that are manually adjusted for screen size. The lightmap resolution is updated to reflect better the necessity of the static mesh within the scene, between sixteen and one-hundred twenty-eight. One example of the merged actor can be seen in Fig. 3.55.

[70] "Unreal Online Learning Courses," *Epic Games Online Learning*, updated 2021, accessed February 15, 2021, https://www.unrealengine.com/en-US/onlinelearning-courses.

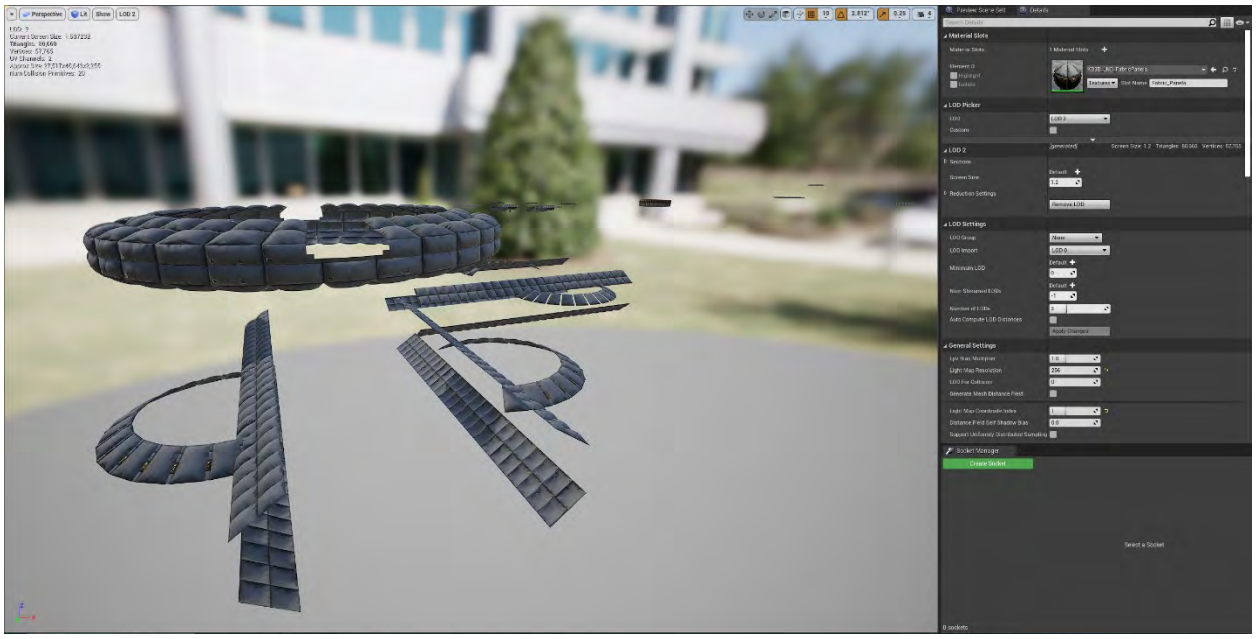


*Fig. 3.55 Unreal Merged Actor LOD and Lightmap*

Increasing the visual fidelity to include reflections was achieved in the following manner. The material within Unreal does not create/capture the reflections of lighting and the world around it by default. Reflection capture spheres must be placed, and the reflections baked into the objects to have reflective objects. However, baking reflections come at a cost, and the more reflective materials baked increased resource draw inside Unreal. After discussing the case study's requirements with SCAD ITGM Professor Charles Shami, it was determined that the best solution to the problem was to compromise having reflection only baked near the lunar impact site and not on sites/buildings far in the background. The reflection capture spheres can be seen in Fig. 3.56.

*Fig. 3.56 Unreal Reflection Capture Spheres*

After the reflection capture is placed and built into the lighting, a star map representing outer space was created. The method chosen was to create an HDRI Backdrop upon Professor Shami's suggestions after attempting to use the BP_SkySphere, which ended in a repetitive and undetailed result. The HDRI Backdrop plug-in must be enabled, which is not loaded by default, to use an HDRI within Unreal. Epic Games has outlined the steps to using the plug-in in Unreal's documentation,[71] or reference Appendix E.2 Using Unreal HDRI Backdrop.

The image used for the HDRI was blurry and lacked definition despite being an 8k image on the first import. Upon further investigation with Professor Shami, it was determined that since the HDRI Backdrop was a significant distance from the camera, the texture was displayed at its lowest MIP level. The MIP gen settings were turned to No MIP Maps inside the texture settings, and the issue was resolved. The HDRI Backdrop is available in Fig. 3.57.

---

[71] "HDRI Backdrop," *Unreal Engine Docs*, ver. 4.26, last updated 2021, accessed August 1, 2021, https://docs.unrealengine.com/4.26/en-US/BuildingWorlds/LightingAndShadows/HDRIBackdrop/.

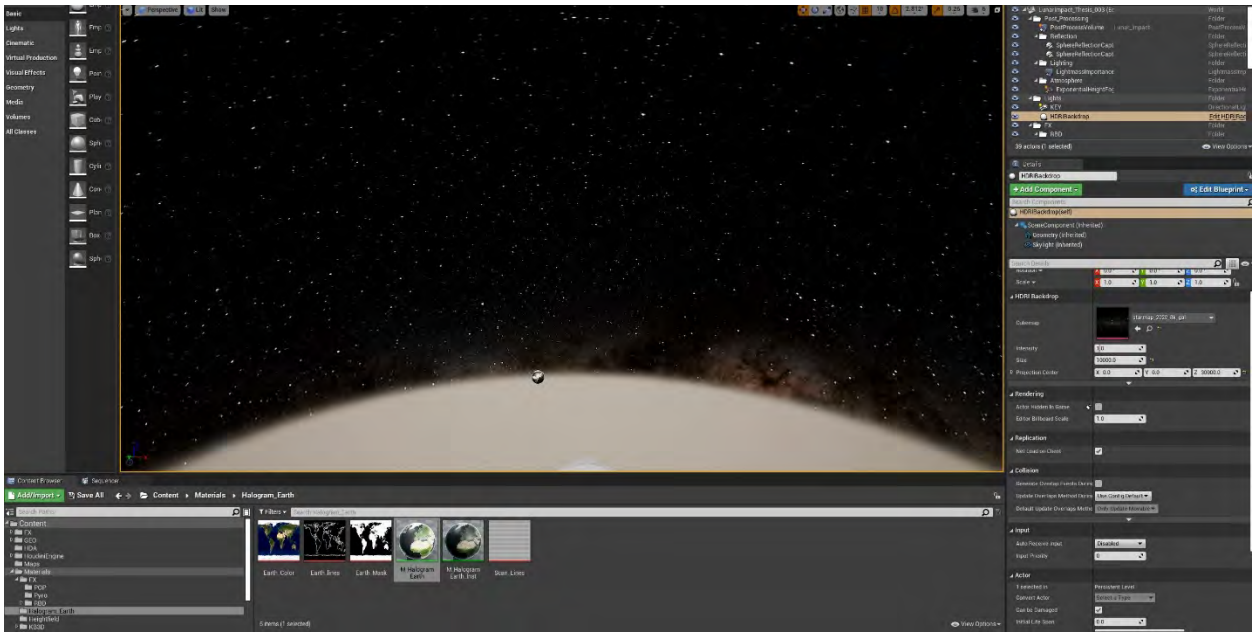


*Fig. 3.57 Unreal HDRI Backdrop Star Map*

Instead of painting an Earth texture onto the HDRI and used within the backdrop, it was decided to use an actual sphere model with an Earth material applied, provided by Professor Shami. By having a physical piece of geometry in the scene, there is greater control over the look/feel of the object and respect parallax all the better within the camera. (The Unreal Earth model can be seen in Fig. 3.58, and Appendix E.3 Create Unreal Earth Model.)

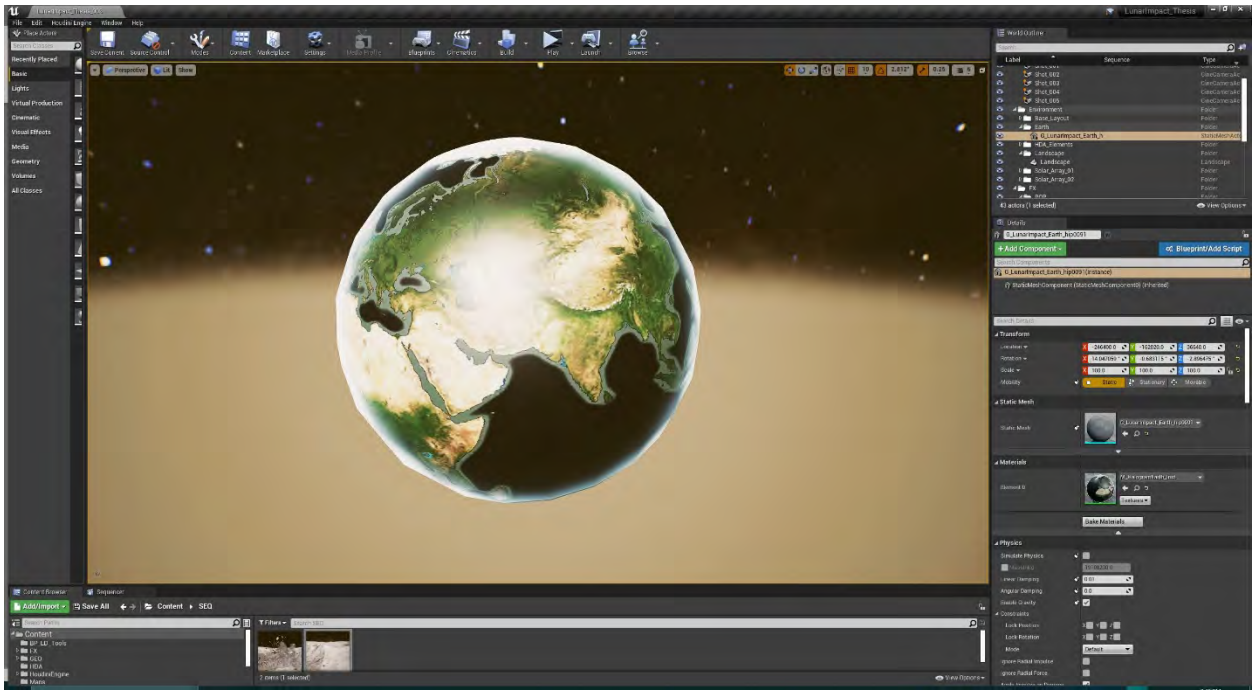


*Fig. 3.58 Unreal Earth Model*

Six different lunar surface rock models were added to address the remaining flat appearance of the surface. The rocks were created within Houdini using a similar technique used previously in *section 4.2 Rigid Body Dynamics* for the impacting rocks and surface mesh. However, this led to a problem within Unreal. Either the rocks had to be placed by hand or a Houdini scatter HDA, which requires a recook of the Engine with each change. At Professor Shami's suggestion, Unreal's foliage tool was used instead. The largest advantage the foliage toolset offers is the ability to live place and update the actors within the scene. Compared to the Houdini Engine, where a recook occurs each time a change is made and makes a new static mesh, the foliage tools allow the randomization of orientation and scale live.[72] The foliate step description is available in Appendix E.4 Lunar Rocks using Unreal Foliage.

---

[72] "Foliage Tool," *Unreal Engine Docs*, ver. 4.26, last updated 2021, accessed August 3, 2021, https://docs.unrealengine.com/4.27/en-US/BuildingWorlds/Foliage/.

The rocks are then painted around the scene paying attention to placement in relation to the main FX. The results can be seen in Fig. 3.59.

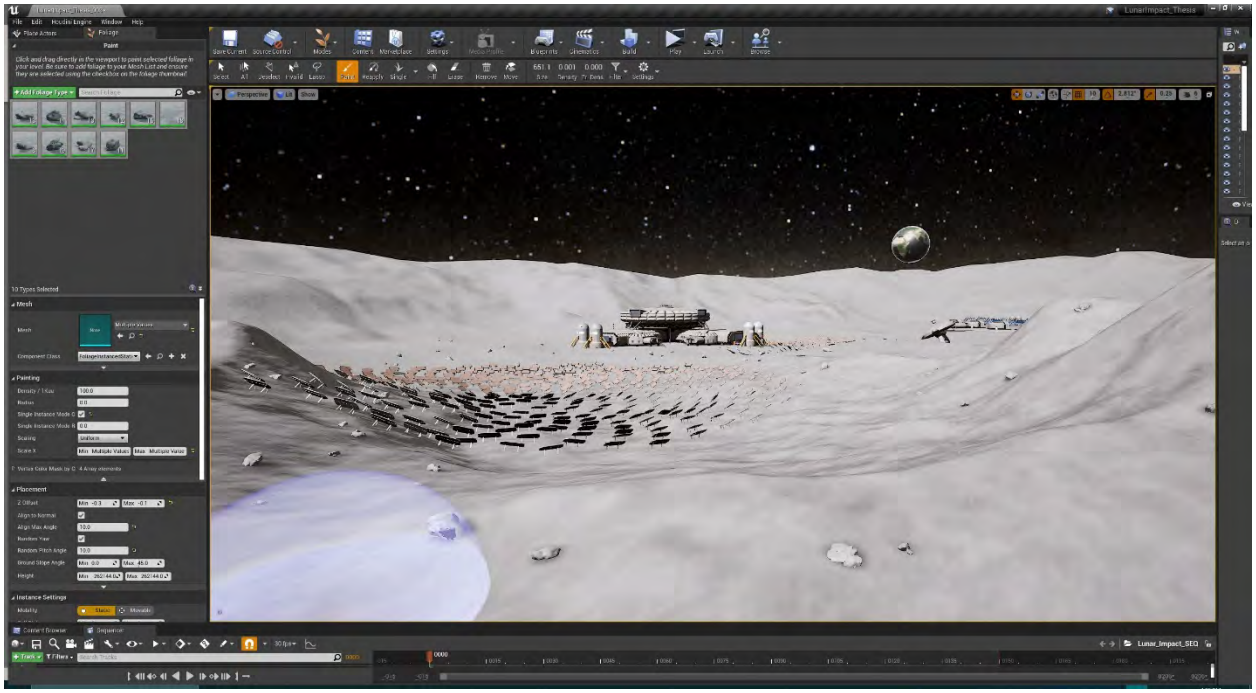


*Fig. 3.59 Unreal Foliage Tool Rock Placement*

Once the final rock is placed, the Unreal scene is complete in terms of needed geometry.

### 3.2.5 [Methodology] Solutions to Visual Fidelity and Optimization in Unreal

The following steps deal with visual fidelity inside Unreal and any needed optimization, including lightmass importance and post-processing.

The lunar scene is populated in its current state, but the edge between the landscape and the HDRI Backdrop feels too clean. The most straightforward remedy to the border was to use an Exponential Height Fog to create a soft gradient color transition along the HDRI. It should be noted that the Exponential Height Fog is used for visual appeal, in contrast to scientific accuracy. The use of the Exponential Height Fog can be seen in Fig. 3.60, with the

written component found in Appendix E.5 Unreal Exponential Height Fog.

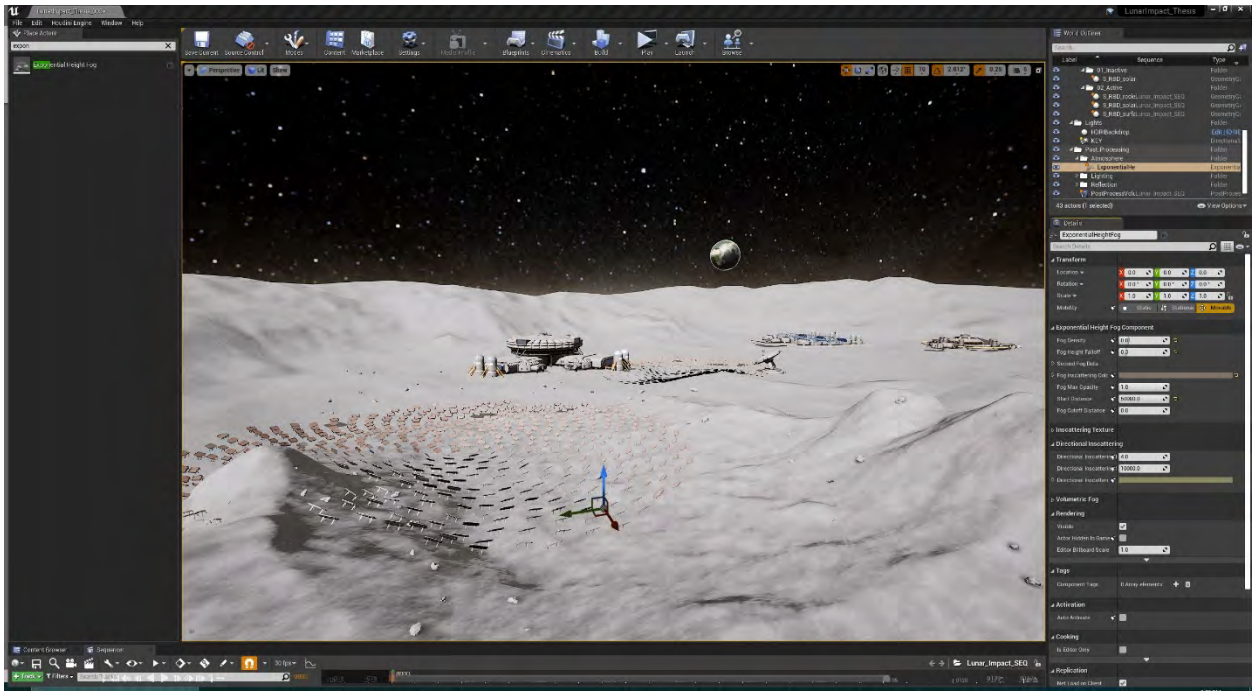


*Fig. 3.60 Unreal Exponential Height Fog*

Lightmass deals with the Unreal scene lighting computation dealing with the complex interaction of elements, such as area shadowing, diffuse reflection, and the precomputation of stationary and static mobility lights.[73] However, the entire scene will be analyzed at the highest quality without telling Unreal where to focus the lighting calculations, causing longer bake times and the unnecessary use of texture detail. The easy solution is to force Unreal to focus its calculations within a specific region using a Lightmass Importance Volume. The Lightmass Importance Volume forces Unreal to emit photon samples at the pre-set lighting quality within the bounds while anything outside the bounds is reduced to one photon sample.[74] Lightmass Importance settings can be found in Appendix E.6 Unreal Lightmass

[73] "Lightmass Basics," *Unreal Engine Docs*, ver. 4.26, last updated 2021, accessed April 15, 2021, https://docs.unrealengine.com/4.26/en-US/RenderingAndGraphics/Lightmass/Basics/.
[74] "Lightmass Basics."

Importance Volume.

Post-processing is perhaps one of the most critical elements within the Unreal scene for creating a higher visual fidelity besides the actual FX. Any visual elements that would typically be created after principle photography in the off-line workflow, such as a vignette or the application of a Look-Up-Table (LUT), have to be applied in real-time. Post-processing can be applied in one of two ways depending on the desired result. The first method is to adjust the attribute on the object/actor that generates the effect. For example, to change the temperature of the Unreal scene, the user could go to the lighting setup and adjust the attribute, temperature. However, adding a vignette onto the final render would require the user to change the settings within the renderable camera. Instead, the second option is to create a Post-Processing Volume[75] and use it to override the settings of all elements within its bounding region, including scene temperature, camera chromatic aberration, vignette, and the application of a LUT. The initial post-process volume setup can be found in Appendix E.7 Unreal Post Process Volume.

A problem arises within Unreal when attempting to work with LUTs and the post-processing volume. In many VFX software, including Photoshop and Nuke, a LUT is read in through a .cube file format, allowing the program to read the file and apply the color changes. A .cube file is a text file that defines RGB values used in a look-up table to adjust the values of an image. The .cube file is human-readable, written in decimal format, and capable of containing three one-dimensional tables or a single three-dimensional table.[76]

---

[75] "Post Process Effects," *Unreal Engine Docs*, ver. 4.26, last updated 2021, accessed July 28, 2021, https://docs.unrealengine.com/4.26/en-US/RenderingAndGraphics/PostProcessEffects/.
[76] *Cube LUT Specifications Version 1.0*, (Adobe, 2013) accessed August 1, 2021, https://wwwimages2.adobe.com/content/dam/acom/en/products/speedgrade/cc/pdfs/cube-lut-specification-1.0.pdf.

Unreal does not accept a .cube file, instead opting for a single three-dimensional LUT image file in comparison to the .cube's text-based one/three-dimensional tables. The Unreal image file must be a 16x16x16 LUT unwrapped to a 256x16 texture.[77] Epic Games provide a neutral LUT file template (Fig. 3.61) to apply the desired changes prior to being used within the post-processing volume, but requires the end user to transfer any existing .cube LUT files to an image format (Fig. 3.62).

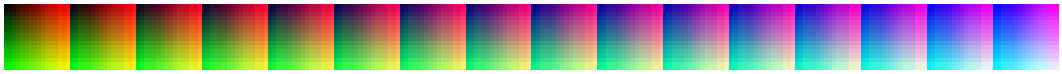


*Fig. 3.61 Unreal Neutral Look-Up Table (LUT)*

*Fig. 3.62 Unreal Edited Look-Up Table (LUT)*

There are two primary ways to create the LUT image files used within Unreal. The first is to take the template LUT .png provided and manually create adjustment layers within Photoshop to achieve the desired look outlined in Unreal's LUT documentation.[78] The second option is to use Photoshop or Nuke to apply a preexisting LUT to the template .png. It was decided to use Nuke and a preexisting LUT to create the required image file (Fig. 3.63). Please reference Appendix E.8 Creating Unreal LUTs Inside Nuke for the Nuke guide to creating an Unreal LUT.

[77] "Using Lookup Tables (LUTs) for Color Grading," *Unreal Engine Docs*, ver. 4.26, last updated 2021, accessed August 2, 2021, https://docs.unrealengine.com/4.26/en-US/RenderingAndGraphics/PostProcessEffects/UsingLUTs/.
[78] "Using Lookup Tables (LUTs) for Color Grading."

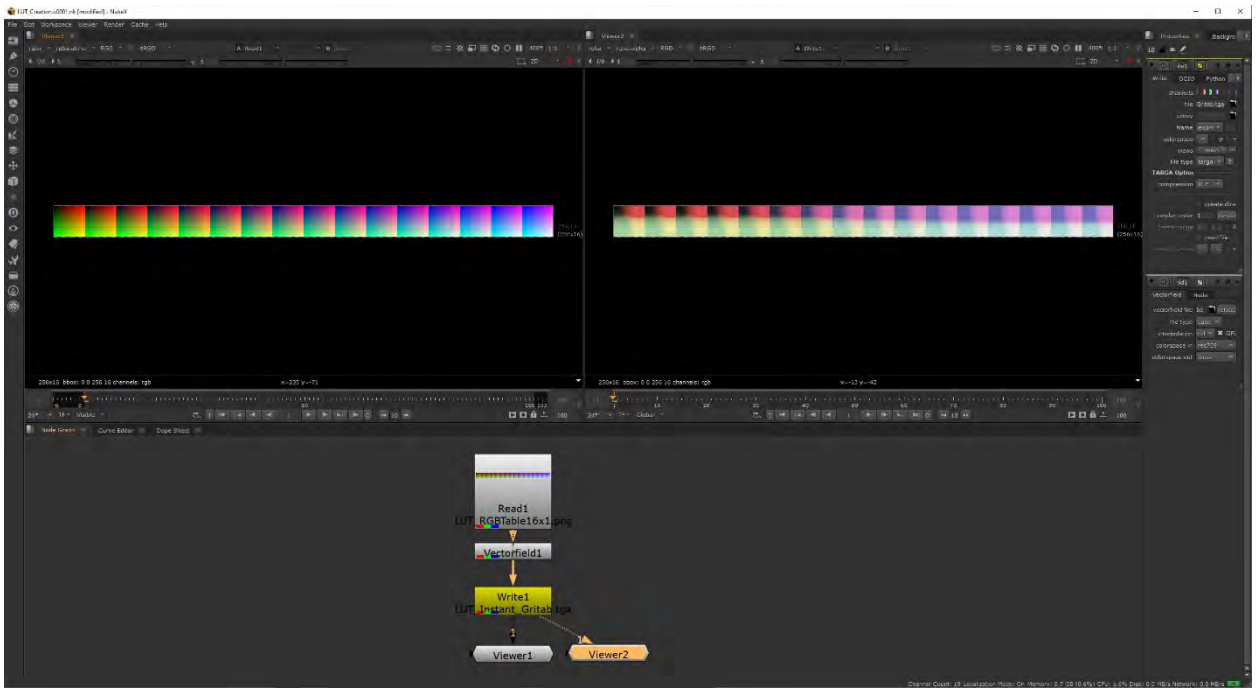



*Fig. 3.63 Nuke Look-Up Table (LUT) Creation*

After the LUT is created, the final step is to apply it within the post-process volume. See Appendix E.9 Unreal Post Processing Utilizing a LUT.

Final look development and optimization inside Unreal has several differences and similarities to designing for an off-line methodology. While similarities do exist, the differences cause more significant challenges that must be considered before transferring data from the choice DCC.

## 4. FX Timeline

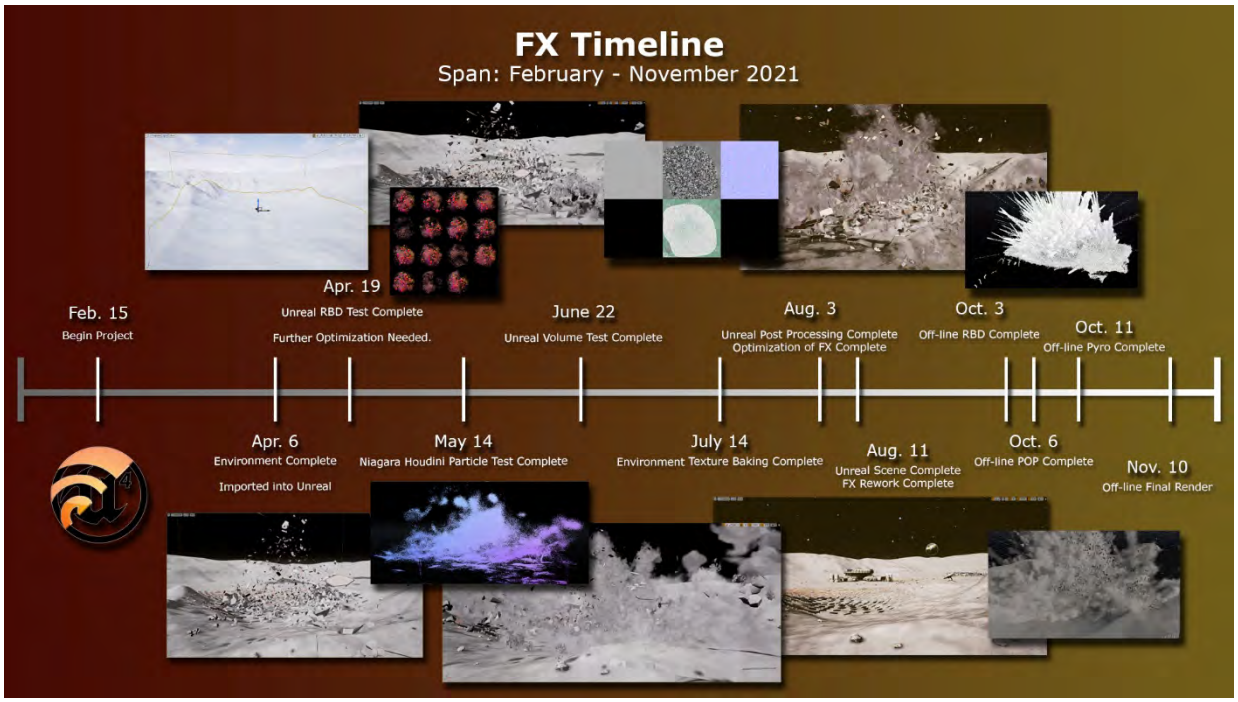



*Fig. 4.1 FX Timeline*

Notes were taken during the process to best document the challenges faced when creating the real-time FX. The FX timeline and notes archived all the steps taken and the conversations with the committee members. The timeline will begin from principle layout and pipeline construction to final capture settings. Some areas overlap or reverse as ideas were tested and proved effective or ineffective. The final timeline taken from these notes and the project as a whole can be seen in Fig. 4.1.

## 5 Environment Summation

Thanks to Unreal's landscape toolset, there were little to no differences between the steps taken for the off-line and real-time setups when constructing the base environment elements. The challenges and differences between the two workflows presented when

attempting to migrate the data from Houdini into Unreal.

Houdini HDA's and the Houdini Engine fixed the issues of different heightfield formatting and world-scale but presented a new set of challenges that must be considered when implementing into Unreal. The challenges were HDA actor spawn location and project memory clogging.

Best practices are to always zero out to world origin any instance of an HDA actor. This avoids a miscalculation of the returned actors' location. Where the heightfield and geometry meet, a slight seam appears in high slope regions. Thus the environment elements must be designed to keep these seams in lower sloped or inconspicuous locations.

Another issue when testing HDA's within Unreal is that all created assets are recreated in the Houdini Engine Temp folder of the project with each recook of the node. Multiple node cooks can very quickly cause memory clogging within Unreal. Regular cleanups of the old geometry version within the temp folder will prevent memory overload and system drain.

### 5.1 RBD Summation

In contrast to the environment construction, the Rigid-Body Dynamics had a noticeable difference between the necessary steps for off-line and real-time. The challenges and differences held such influence as to change the initial simulation geometry construction.

There are several vital limitations to keep in mind when designing RBD-based FX for real-time. When fracturing the initial FX geometry for Unreal, it is critical to remember that polycount restricts real-time, so the detail must exist more within the shader compared to geometry. Instead of fracturing the geometry into a large number of small pieces and using constraint attributes to glue the pieces together, it is advised to fracture based on distance

from impact and reduce interior face detail.

The Houdini Engine's recomputation of vertex normals caused a mismatch between elements. The incorrect geometry was removed from Unreal and made a part of the imported alembic cache. It is recommended to use the Houdini Engine for any non FX related pieces, like environments that are not fractured.

Unreal requires that all UVs be cleanly laid out without any overlap or nested islands to such a degree that the UV Layout scale inside Houdini (512x512, 1024x1024, 2048x2048) actively affects the lightmap scale and quality inside Unreal. It is highly recommended to pack the UVs at one resolution and create a second UV set at another as lightmaps and unchecking "Generate Lightmap UVs" inside Unreal.[79] Houdini's UV Flatten and UV Layout using the "iscale" variable, explained in the *Preparing RBD FX Mesh* section as "Island Scale Attribute," are incredibly helpful in creating the needed lightmaps and setting their resolution.

Another significant issue is with the alembic file format itself. The file format is experimental inside Unreal and has proven prone to difficulties. Alembic caches are smaller and naturally have the animation attached to the geometry mesh, unlike FBX; however, once imported into Unreal, the user loses all access to the static mesh controls, including the creation of Level of Detail (LODs) and lightmap resolution settings. Upon further investigation, it was discovered that unless the "Running" attribute is keyframed for correct activation time, the RBD elements will loop throughout a sequence longer than the cached simulation. Perhaps in the future, when the import is more stable, alembic can be used

---

[79] "Generating Lightmap UVs," *Unreal Engine Docs*, ver. 4.26, last updated 2021, accessed July 28, 2021, https://docs.unrealengine.com/4.26/en-US/WorkingWithContent/Types/StaticMeshes/AutoGeneratedLightmaps/.

reliably, but as of 4.26.1, this user does not recommend the alembic file type.

One solution is the Vertex-Animation-Texture (VAT) tools by SideFX Houdini, newly updated in September of 2021.[80] The tool allows an RBD simulation to be written out as a readable texture applied to a static mesh within Unreal using special VAT material functions. Another solution, if possible, is an update by Epic to allow the alembic file format to have control over the static mesh component of the animation similar to the FBX file but maintain the animation controls.

### 5.2 POP Summation

The Particle Operator (POP) simulation had some essential differences in the off-line and real-time methodologies. The challenges had an impact on the exporting of the final simulations and how detail creation is handled.

Prior to working with POP simulation FX for real-time, many less noticeable problems must be considered to succeed in the end result. To be able to use Houdini point caches in Unreal, the Houdini Niagara plug-in must be installed and match exactly the version of Unreal.

It is highly recommended to base current pipelines around the download and use of a Houdini python two release until the transition to python three is confirmed stable.

It is important to ensure the Houdini Labs toolset is updated using a minimum version of 440. As of 18.5.462/495, the Houdini Niagara ROP only writes out correctly when Houdini is set to run in Auto-Update mode.

---

[80] "Vertex Animation Textures in Unreal," *SideFX Houdini*, updated September 7, 2021, accessed September 17, 2021, https://www.sidefx.com/tutorials/vertex-animation-textures-for-unreal/.

When reading in the point cache, Niagara fails to either spawn or update the particles. Special point cache reference functions must be placed within the emitter state, particle spawn, and particle update to correctly use the data inside the Niagara emitter system.

The Niagara system emitter must be set to run on the GPU; otherwise, the CPU will lag before reaching even twenty-thousand particles, compared to Houdini's typical particle simulation size of one million or more.

Niagara has a problem with growing point count numbers due to the Houdini plug-in point cache. Niagara also has the problem of not recognizing when particles have died. An attribute must be created to declare the particle dead before being reaped inside Houdini, allowing Unreal to ensure the particle is culled.

A prevalent practice in Unreal is to cull objects based on the camera view and distance to reduce resource drain. This culling caused the particle system to flicker when placed within the world. Thanks to a discussion with SCAD ITGM Professor Ari Cookson, it was found that the particle flickering was due to the bounding box size. The particles were traveling outside the defined visible bounds of the simulation, causing the particles to be constantly culled and immediately respawned infinitely. The solution was to increase the max bounds inside the Niagara system and increase the emitter element's bounding box inside the scene.

At the moment, many of these challenges and difficulties arise due to Niagara's interface and its overall design. This user found the system in its current state to be difficult to use in designing FX, especially its requirement of sampling a single point cache in a minimum of three places. In future iterations, designing the interface using the blueprinting system as inspiration would be more intuitive, creating a precise flow of information through the system similar to Houdini and other node-based DCCs. Another benefit of a blueprint-

inspired system would be the ability to chain systems/emitters together, allowing ease of event handling and linking of parameters, which is challenging to keep clearly defined in version 4.26.1.

### 5.3 Pyro Summation

The Volume (Pyro) simulation had numerous pivotal challenges in designing the real-time FX compared to the off-line methodology. These key challenges required the use of the combination of multiple simulation types, including POP from the previous section.

The use of volumes can be an incredible boon when designing FX for real-time, especially in cinematics and world-building. Nevertheless, special care must be paid in planning and execution to avoid fatal errors and problems that arise from a real-time engine. The most significant design problem/flaw is that Unreal has no proper volume (voxel) support compared to DCC programs.

There are ways to fake a voxel-based system, like a ray marching system outlined by Ryan Brucks, but these solutions quickly can become expensive and inappropriate to apply with other large-scale FX. The question then returns to the best approach to creating pseudo volumes and the problems that arise.

The first challenge in creating the pseudo volume is how to recreate the motion of the volume within Unreal. One simulation type that has proven successful when recreated for Unreal is particle simulations. The solution was to advect a particle simulation's motion using a volume simulation. The resolution of the driving volume was reduced to improve Houdini optimization since hi-resolution is not needed to drive a particle's movement.

The total number of points had to be carefully regulated to prevent GPU overdraw issues

within Unreal and the translucent sprite material overlapping. The point cache is brought into Niagara once the particle simulation is complete using the same outlined method in *section 4.3, Particle Operators*.

The problem with volumes is the changing/undulating mass, unlike the initial particle system, which was based on static meshes moving through space. The answer was to create a material flipbook that could be animated inside the material and placed on the moving particles.

When a volume material was applied to the particles, the sprites had a cutout figure appearance. The edges had a large amount of anti-aliasing, and the sprite had no transparency. The best way to fix the sprite material was to set the shader output method to translucent instead of masked, but this removed the ability to use normal maps to help remove the flat appearance.

A problem within Niagara is that it does not have a built-in script to remap float values in the interface. A dynamic input script was created inside the Nagara scratchpad to give the Niagara system the ability to remap the opacity from one value to another.

At this point, there are problems and limitations on the total visual fidelity provided by Unreal volumes for hero assets. The possible improvements could be but are not limited to total voxel importation support, an optimized ray marcher system, or volume-specific material/sprite options. Until there is more support for volume creation inside Unreal, it is recommended to limit volume FX to non-hero assets, such as background plumes or other supporting elements.

### 5.4 Real-time Summation

Unreal and game engines naturally have techniques built-in that are not common in Houdini and other DCCs. The largest challenges to tackle were material draw calls, level of detail (LODS), and lightmaps. It is possible to import a static mesh into Unreal as an FBX with multiple material assignments, but the more assigned material, the larger the draw call. Instead, the meshes are broken up into various pieces inside Houdini based on the assigned material. Then the actors are merged within Unreal to create a single static mesh with one material assignment.

Since polycount is incredibly important for Unreal, the high detail static meshes are draining resources since Houdini's LOD system was not set up. Each static mesh uses Unreal's LOD generator to create four LOD levels and then are manually set to specific screen sizes.

Adding lunar rocks to the surface of the moon causes issue with efficiency within the engine. Unreal has a detailed foliage tool that works innately within the program and allows the painting of objects within a scene without the Houdini Engine's detriment of recooking each time a change is made.

Lightmaps are a concept unique to game engines since off-line DCCs do not need to pre-compute lighting. It is recommended to adjust the lightmap resolution scale based on two factors: the overall space the object will occupy within the scene and the importance of the object to the camera and action currently happening within the shot.

It is recommended to focus the lightmass build using a Lightmass Importance Volume only in critical areas, such as the main shot.

Since Unreal is a real-time engine, all the post-processing that would typically happen inside a separate DCC occur live in the engine, leading to possible issues. Instead of

adjusting each effect in the defining element, it is advised to use a Post Process Volume as a master control for all post-processing.

The Post Processing Volume does have a problem with Look-Up Tables (LUT). Instead of using the common .cube, text-based file format, Unreal uses a three-dimensional one-table image file. [81] The difference in LUT file format causes issues by requiring the end-user to create or convert existing .cube files into an Unreal friendly image. Unreal's post-processing and environment tools offer a large amount of variety and options to influence the final look and performance of the scene.

[81] "Using Lookup Tables (LUTs) for Color Grading."

## 6 Conclusion for Designing FX for Real-Time and Off-Line


*Fig. 6.1 Off-line, Real-time Case Study Side by Side*

An impact on the lunar (moon) surface was chosen as a case study to examine the advantages and disadvantages of designing effects for virtual production and real-time environments. The problems and solutions found in the case study when designing the effects for real-time demonstrate the dependency upon off-line techniques that still exist within the real-time pipeline.

The examination clarified the necessity when designing effects for real-time to identify critical components that could be designed modularly. When the effects were designed with crucial elements/steps designed in a modular fashion independent of other FX elements,

changes could be made mid-project seamlessly with minimal effort. Specific pieces could be modified and reimported into Unreal with ease.

The most critical point between the systems is what needs to be achieved, compared to the time available to achieve the shot. The cost of time accrued in simulation reworks and loss of visual fidelity offset any gains offered by real-time in this test case. When designed on the scale of an industry project for an off-line hero effect ported into real-time, the simulation had to have a fraction of the off-line elements due to current engine limitations.

The best approach to designing FX for a real-time environment at this time is viewing Unreal as a staging ground with the ability to render modular effect assets within the completed scene.

Unreal is currently not a feasible substitute for creating or displaying photorealistic hero FX assets. For Unreal to be a reasonable substitute for displaying hero FX assets from an off-line workflow, specific tools and updates must be made to the RBD import, Niagara controls, and volume simulations.

RBD simulations must have a unified file format and a more extensive engine poly-count support to handle the import from other DCCs, compared to the current FBX or experimental alembic. The Chaos system was not tested due to being reported unstable and available only in a special beta build of Unreal during the examination timeframe.

The Niagara particle system needs a number of updates in order for it to be a viable off-line hero FX alternative. The interface could benefit from a rework. An alternative to sprites would also be helpful for FX creation; the system needs to handle a larger particle count and better controls for defining attributes of particle behavior.

Volumetric (Pyro) voxel-based simulations are not currently supported within Unreal. The most impactful updates would be the support of voxels within Unreal or creating an

optimized system to replicate volumes, a more efficient ray-marcher.

The project was not successful in creating photorealistic elements to be used in real-time. The effects are limited in control and break when pushed beyond specific predefined settings. The most significant limitation for this project was primarily due to hardware restrictions. The project was designed and built using a GTX 1080 TI graphics card, which would return a semi-consistent 43-48 FPS once the scene was optimized. When the same optimized project was run on another machine using an RTX 3070, the returned frame rate was 62-71 FPS. Hardware limitations impacted the fundamental design of the real-time effect.

The introduction of virtual production and real-time engines within the film and television world has started a movement akin to a new renaissance of creation within the industry and will continue to influence technology. While FX in a real-time environment is not fully realized, it is in demand and will continue to be in the future.

# Appendices

## Appendix A.1 Creating Houdini Heightfields.

- Read in *Terrain Party* scan data using heightfield_file, Surface Operator (SOP). The two locations used were the Atacama Desert, Chile, and the Meteor Crater, AZ.
- Use heightfield_layer SOP set to "Add" and combine the scan data.
- Remap the @height attribute using heightfield_remap so that the value of the minimum bound is at zero on the y-axis.
- Create a heightfield_maskbyfeature to generate a mask based on the slope of the volume.
    - A heightfield_maskblur then blurs the mask to create smooth edges.
    - The mask and heightfield are fed into a heightfield_noise set to "Add," which applies variety to the masked areas.
    - Finally, the mask is cleared using a heightfield_maskclear.
- The feature masking and addition of the noise process are repeated using varying noise parameters to add irregularity to the volume.
- The heightfield is remapped back above zero on the y-axis and resized using heightfield_xform to the correct world scale.

## Appendix A.2 Base Layout using Variant Attribute.

When using a randomly generated @variant attribute in a CopyToPoint SOP, a disproportionate number of large-scale buildings were created. Instead, the models were grouped by relative size. The @variant attribute was converted from an integer to a string, allowing the concatenation of the @variant attribute, thus giving control over the model size chosen based on group number. For example, a small model in group one numbered seven

would be given the concated @variant attribute “1_7”, compared to a larger model in group four, numbered three being “4_3”.

- The area for buildings to be placed is painted on the heightfield using a heightfield_paint SOP. The painted mask is combined with a heightfield_maskbyfeature SOP using a heightfield_wrangle to avoid any steep angles.

```
f@mask *= volumesample(1,"mask",@P);
```

- The heightfield is fed into the first and second input of a heightfield_scatter SOP.
  - The heightfield_scatter is set to “Total Points using Mask.”
  - The desired number of buildings (20) is input as the total point count.
- AttribDelete SOP removes all attributes except N, P, orient, and pscale.
- Blast SOP, set to @name=height removes heightfield but leaves scattered points.
- AttribRandomize generates a random 4D orient attribute to have uneven directions of the buildings.
- Each asset is packed and given a string @variant attribute. The models are then merged and given a string group. When concatenated together, each model has a unique identifier. Example: 1_3 = Group 1, Asset 3.
- New Variant Attribute
  - Before ForEach SOP
    - A random group index is generated for each point and assigned.
    - The point goes to its assigned group index and retrieves the total number of variants in the group.
    - An asset index is generated and concated with the group index to create the final @variant attribute.

- Inside ForEach SOP, set method to “Fetch Feedback” and “By Count” on the ForEach_end
    - The points are rayed to the surface of the heightfield once more using a ray SOP set to “minimum distance.”
    - The variant attribute grabs the bounding information of the correlating model.
    - The bound information opens a point cloud and searches for other scattered points within that radius.
        - If another point is found, a movement vector is created, and the points are moved apart.
    - The pcopen vex function does not reinitialize itself when running from within a for loop inside a wrangle node, so a foreach block was created to run a given number of iterations.
- The packed model is then copied to the points using CopyToPoint SOP and the @variant attribute as the identifier.
- Any unneeded attributes are removed using AttribDelete SOP.
- Finally, manual transforms are applied to specific buildings as desired.

**Appendix A.3 Updating Heightfield for Base Location.**

- ForEach SOP set the method to “Fetch Piece or Point” and the iteration method to “By Pieces or Point” on ForEach_End.
- On the ForEach SOP, click “Create Meta Import Node.”
    - On the ForEach_begin_metadata SOP, set the method to “Fetch Feedback.
- Before ForEach SOP:

- Connect the packed base layout into the ForEach_begin.
- Connect the heightfield into the ForEach_begin_metadata.

- Within the ForEach SOP:
  - Wire model feeds into attribute wrangle and, using VEX, creates a position attribute called “packedPos.”
  - Connect heightfield metadata import into the first input of heightfield_maskbyobject. While connecting model feeds into the second input.
  - A heightfield_maskexpand SOP expands the created mask by a radius of 10.
  - Heightfield_maskblur SOP, then blurs the mask by the channel referenced value ch("../heightfield_maskexpand3/radius"), equal to 10.
  - Using a volume_wrangle SOP the heightfields @height attribute is linear-interpolated (lerp) between its current height and the y position from the created @packedPos attribute.

```
vector pos = point(1,"packedPos",0);
vector bound = point(1,"bound",0);
float offset = chf("Height_Offset");
f@height = lerp(f@height,(pos.y - (bound.y/2) + offset),f@mask);
```

  - A heightfield_maskclear SOP clears the current mask on the heightfield.
  - The masking and height update process is repeated aimed toward a specific model to ensure the heightfield is updated correctly.

**Appendix A.4 Solar Panel Phyllotactics Based on Lunar Surface.**

- Volume_wrangle removes any overlapping mask areas in comparison to the building

layout.

```
f@mask *= 1-volumesample(1,"mask",@P);
```

- Attribute_wrangle and vex set to run over “Detail (only once)” creates points in a phyllotactic pattern[82].
- An orientation normal is calculated based on the points position from the world origin.

```
v@N = -normalize(v@P - {0,0,0});
v@N += chv("Normal_Adjust");
```

- CopyToPoint SOP instances the phyllotactic to the previously scattered points.
- A ray SOP set to “Minimum Distance” moves the phyllotactic points to the surface of the heightfield.
  - The points are connected in the first input with the updated heightfield in the second.
- The orient normal is adjusted to account for the angle of the surface sitting upon using code within the HF scatter node to create new orientation normals along the surface. These normals are then rotated using the original inward-facing normals, and finally, the HF normals are set as the UP vector.
- The new normals are compared using a dot product, and an inputted up vector {0,1,0} and then removed if beyond a specific threshold. Points also self-check to see if they are within the bounding area of an individual solar panel and removed to prevent a collision.

---

[82] Deborah R. Fowler, “Phyllotactic Pattern,” *Deborah Fowler Math for VSFX*, last updated February 13, 2021, accessed March 8, 2021, http://deborahrfowler.com/MathForVSFX/Phyllotaxis.html.

- Each building creates a bounding box used to group any solar panels that got placed inside and deleted.
- The solar panels are copied to the new points and are grouped based on the original point that spawned its phyllotactic.

**Appendix A.5 Exporting Heightfield Data from Houdini to Unreal.**

- Connect completed heightfield into a heightfield_resample SOP.
    - Set sampling to match desired output size in pixels (4096 X 4096).
- Plug resample into a heightfield_output SOP.
    - Set "Check Format" to "Single Channel."
    - Set "Type" to "16b Fixed."
    - Set "Image Channel" to "Height"
    - Turn on auto remap and set to zero and one.

**Appendix A.6 Houdini Engine HDA Setup.**

- The HDA created named "realtime_env" allows users in Unreal to input Houdini read parameters for the desired HF, Mask Object, and Houdini Geo to be used. Along with object scale and adjustments to the heightfield visibility radius.
- Within the HDA Inactive Geometry:
    - The user inputted inactive geometry is feed into an attribute_wrangle SOP to calculate the centroid and move the geometry to the world origin.
    - The normal attribute N is promoted from vertex to point type using the attribpromote SOP.
    - The points then grab their normal and determine if they are the outermost

points and set themselves in the “ray” point group based on the dot product.

- The ray point group is expanded to give a more extensive data set. Any points that exist at the base of the model are ignored.
- Each point within the expanded ray group calculates a rayIncident vector that respects the surface of the original heightfield. Then the original ray point group grabs this incident vector.
- CONTINUED IN “HDA Inactive Geometry Continued” section.

- Within HDA Heightfield:
  - The user inputted heightfield is masked by the user inputted mask object using a heightfield_maskbyobject SOP.
  - A heightfield_xform SOP references the centroid transformations from the user input inactive geo. It then transforms the heightfield so that the center of the masked region is not resting on the world origin. Used reference:

    Translate X = -point("../OUT_CENTER/",0,"center",0)

    Translate Z = -point("../OUT_CENTER/",0,"center",2)
  - The mask volume is expanded or shrunk depending on the user inputted value using heightfield_maskblur SOPs and a switch node.
    - One maskblur is set to “Shrink” while the other is “Expand.”
    - The switch node decides which of the two functions to use depending on using a channel reference.

      if(ch("../HF_Visibility_Radius")<0,0,1)
  - A heightfield resample SOP uses a user inputted value to up-rez or down-rez the voxel scale.
  - A name SOP renames the mask volume to “Visibility” to be used inside an

Unreal material as the opacity mask using the "Landscape VisibilityMask. [83]"

- The finalized heightfield is merged with the completed inactive geometry and outputted to Unreal for conversion to a landscape actor.

- Within HDA Inactive Geometry Continued:
  - The masked heightfield is converted to act as a collider for the edge points on the geometry.
    - Inside a volume_wrangle SOP:

      f@height *= f@mask;

      f@height += 1;
  - The "ray" point group on the inactive geo is passed through a ray node using the rayIncident vector as the direction and colliding with the heightfield mask region.
    - "Entity" is set to "Points."
    - "Method" is set to "Project Rays."
    - "Direction" from equals "Attribute."
      - "Attribute" is rayIncident.
    - Any unneeded attributes and groups are deleted.
    - An unreal_material SOP is created to assign materials for the inside and outside faces.
    - The heightfield and completed geometry are merged, and the display flag is set on a Null output node.

---

[83] "Landscapes," *Houdini Engine for Unreal*, ver. 18.5, last updated 2021, accessed April 2, 2021, https://www.sidefx.com/docs/unreal/_landscapes.html.

- Import HDA into Unreal.
    - Drag and drop HDA into the Unreal scene.
    - Input desired heightfield, mask object, and inactive geometry.
- Create materials to the lunar surface.
    - Right-click and select “Copy Reference” on the heightfield material. Paste reference into the material path of the HDA[84].
- Ensure that the HDA’s translations are zeroed out and click rebuild.

Once rebuilt, bake out the geometry to Unreal and clean Houdini Engine temp files.

**Appendix B.1 Preparing RBD FX Mesh.**

- Read in the previously created heightfield using a File or FileCache SOP.
- Using heightfield_paint SOP set to “Replace,” paint a mask in the desired impact region.
- Convert the heightfield to polygons using the convertheightfield SOP.
- Any geometry that is not masked is removed using an attribute_wrangle set to run over “Points.”

```
if(f@mask == 0)
{
    removepoint(0,@ptnum);
}
```

- The newly created surface is then moved back to the surface of the heightfield using

[84] “Materials,” *Houdini Engine for Unreal*, ver. 18.5, last updated 2021, accessed April 2, 2021, https://www.sidefx.com/docs/unreal/_materials.html.

a ray SOP with the method set to “Minimum Distance” to ensure a match.

- An extrudevolume SOP then extrudes the surface by a depth of -5 to create thickness.
    - The node is also set to export the “Top,” “Base,” and “Side” groups.
- Any unnecessary attributes are removed using the attributedelete SOP.
- A ray group is created on the outer edge points using the same method in *section 4.1 FX Environment Creation* for the lunar surface HDA.
- Multiple group nodes selected the outer edges on the model to create a UV seams group called “uvSplit.”
- Attribute wrangles running over “Primitives” then set the UV scale for each part of the model. The top surface of the model is thus prioritized for space and packing compared to the sides and bottom.
    - extrudeTop

        f@uvScale = 2.0;
    - extrudeSide and extrudeBase

        f@uvScale = 0.2;
- The uvflatten sop separates the UV islands based on the previously defined seams.
    - Flattening Method = Angle-Based (ABF).
    - Preserve Seams = True.
    - Flattening Constraint Sems = "uvSplit" group.
- uvlayout SOP then arranges the UV islands.
    - Variable Scaling.
        - Island Scale Attribute = "uvScale."
    - Packing.

- Island Rotation Step = 22.5.
- Avoid Packing Over Non-Group Islands = True.

**Appendix B.2 Preparing Fracture RBD FX Mesh.**

- Take the FX mesh and, using an attribute wrangle set to run over "Points," move the geometry to the centroid.

```
vector center = getbbox_center(0);
vector min = getbbox_min(0);
v@min = min;
v@center = center;
```

- Prior to Fracture, Geo Prep.
  - Promote vertex normal to point normal using attribpromote SOP.
  - Feed the original FX mesh into input zero and the attribpromote into input one of an attribute wrangle and create a new ray group using the same method outlined in *section 4.1 FX Environment Creation*.
  - Use a blast SOP to remove all but the extrudeSide group of the extrudeVolume SOP.
  - Create a noReduce point group by connecting blast SOP into input one of an attribute wrangle, and the FX mesh into input zero.

```
int pt = findattribval(1,"point","num",@ptnum);
if(pt != -1)
{
        setpointgroup(0,"noReduce",@ptnum,1,"set");
}
```

- PolyReduce the FX mesh.
    - Group = extrudeTop.
    - Percentage to Keep = 6.
    - Use Only Original Point Position = True.
    - Hard Points = Ray and noReduce.
- groupcombine SOP to create noise group.
    - Group Name = Noise.
    - Method = Equals All But.
    - Group = Ray.
- Attribute wrangle to create “rest” attribute.

    v@rest = v@P;

**Appendix B.3 Fracture Point Creation.**

- Prior to Fracture, Point Creation.
    - Read in the packed impact rocks.
    - Use Add SOP to replace the geometry with points.
    - Ray SOP points to the surface of the heightfield setting the “Direction from” to “attribute” and using the velocity “v.”
    - Set @pscale attribute using an attribute wrangle.

        f@pscale = chf(“Pscale”);
    - Copy sphere to the points using copytopoint SOP.
    - Use VDB from particles SOP set to Distance VDB to create a volume from the scattered spheres.
    - Create a volume from the FX mesh using a VDB from polygon SOP.

- Merge the two distance volumes using vdbcombine set to “SDF Intersection.”
- Convert the Sine Distance Field (SDF) to a Fog volume using convertvdb SOP.
- Rename volume from surface to density with a name SOP.
- A volume vop SOP then adds, fits, and multiples noise into the density.
- Points are then scattered inside the density volume using a scatter SOP.
- An ID is then created on each of the points using an attribute wrangle.

```
i@id = i@ptnum;

i@totalPT = i@numpt;
```

- This process is then repeated multiple times scatter points in desired regions of the mesh. The only difference is in the attribute wrangle that creates the point ID. Each iteration takes into account the previous number of points and changes the ID accordingly.

```
int totalPt = point(1,"totalPT",0);

i@totalPT = i@numpt + totalPt;

i@id = i@ptnum + totalPt;
```

**Appendix B.4 Fracture RBD FX Mesh.**

- Feed the fracture mesh and fracture points into a voronoifracture SOP to create the pieces.
- Use a UV Unwrap and UVLayout on the newly created inside group.
- A clean SOP set removes any degenerated primitives, consolidates points, and removes unused points.
- An assemble SOP set to “Create Packed Geometry” with “Create Name” turned off creates the packed geometry needed.

- A group SOP selects the packed-geo based on the impact points from earlier as the bounding object.
- The geometry is run through a split SOP using the selected impact pieces.
- The impact pieces are given a pack number and pack position.

```
i@packNum = i@ptnum;
v@packPos = v@P;
```

- The geometry is unpacked, and each piece is given a random number of fracture points between twenty and forty.
- The same process as before is repeated within a ForEach SOP with the exception that the scatter SOP references the random number of fracture points each piece has using the following channel reference.

```
point("../get_fracture_points_total/",0,"fracPTS",0)
```

- The newly fractured geometry is recombined with the larger fracture pieces and unpacked for final processing before simulation.

**Appendix B.5 RBD Mesh Cleanup and UV Layout.**

- An attribute wrangle gives the new inside primitives a f@uvscale value of 0.2.
- The inside faces are then UV unwrapped and arranged.
- Place a divide SOP and set “Maximum Edges” to three.
- A clean SOP set to “Remove Degenerate Primitives” and “Remove Unused Points” prevents Unreal from crashing on import.
- The unreal_material SOP assigns material paths to the inside and outside face groups of the mesh.
- The geometry is repacked using an assemble node with “Create Name Attribute”

turned off and transferring all attributes.

**Appendix B.6 Set Active and Inactive Simulation Geometry.**

- Use an attribute wrangle to create an identifier attribute for later reference.

  i@packedPT = i@ptnum;

- Set all points to be inactive using an attribute wrangle.

  i@active = 0;

- Group pieces to reactivate using a group node with the same system as earlier when creating fractures.
- In a new branch unpack the geometry and transfer the packedPT attribute.
- Using a blast SOP remove all geometry except the extrudeSide group.
- Feed the original packed geometry into the zero input of an attribute wrangle, and the blasted sides into the first input. Activate all points that are selected and do not exist along the outer edge of the mesh.

```
int pt = findattribval(1,"point","packedPT",i@packedPT);
if(pt == -1)
{
        i@active = 1;
}
else
{
        i@active = 0;
}
```

**Appendix B.7 Create Constraint Geometry.**

- Glue Fixed Constraint
    - Unpack the geometry.
    - Use a Connect Adjacent Pieces SOP to create the connecting primitives.
- Glue Cluster Constraint
    - Delete the nonactive geometry using a blast SOP.
    - Unpack the pieces.
    - Use a Connect Adjacent Pieces SOP to create the connecting primitives.
    - A glue cluster SOP then generates different primitive clusters on the primitives.
    - A split SOP set to "@strength=-1" then breaks the clusters into different pieces.
- The constraints are then merged and passed to the DOP network.

**Appendix B.8 RBD Simulation DOP Network Import.**

(The following list will be transcribed flowing from left to right.)

- DOP Fetch Geometry and Create Solvers
    - Import simulation pieces using the RBD Packed Object node.
        - Connect multiple RBD Packed Object nodes using a merge node.
    - A POP Speed Limit is wired into the third input of a Rigid Body Solver to control the overall simulation speed and rotation.
    - A SOP solver is used to reference the velocity on the impact rocks for activation.
        - An attribute wrangle inside the SOP Solver uses the DOP Geometry as

the first input and an object merge to the input geo as the second input.

```
if(@Frame == i@start)
{
    v@v = point(1,"v",@ptnum);
}
```

- All parts of the top level are fed into a multi-solver to handle the RBD and SOP inputs.

**Appendix B.9 RBD Simulation DOP Network Collisions.**

(The following list will be transcribed flowing from left to right.)

- DOP Create Collisions
  - Fetch the created heightfield using a Terrain Object node.
    - Use Heightfield = True.
  - Feed the Terrain Object node into a Static Solver node.
  - Connect the Static Solver as the first input of a merge node with the previous level multi-solver as the second input. The new merge node now connects the two data stream levels.

**Appendix B.10 RBD Simulation DOP Network Constraints and Forces.**

- DOP Constraints
  - Create a Constraint Network Node. Connect the previous merge into the first input.
    - Inside the Constraint Network node.

- Relationship Tab.
  - Constraint Objects = Name of the object having its constraint geometry attached.
- Data Options Tab.
  - Set the SOP path to the needed piece constraint geometry.

- Create a Glue Constraint Relationship node.
  - Inside Glue Constraint node.
    - Set Data Name to the constraints name, surface_glue_fixed.
- Repeat the glue constraint process for any other needed glue-based constraints.
- Merge the constraint relationship nodes and feed them into the second input of the Constraint Network node.
- Each above step is repeated for all simulation objects, lunar surface, impact rocks, and solar panels.

- DOP Forces and Output
  - Feed the constraint networks into a gravity node and set it to the gravity constant on the moon of -1.62.
  - Connect the gravity into the DOP network's final output node.

**Appendix B.11 Houdini RBD Caching.**

- Read in the cached RBD simulation points.
- Separate the point based on their spawn objects using a blast SOP.
  - The solar panels' points are separated from the points representing the lunar

surface and impacting rocks.

- The points are fed into the second input of a Transform Pieces SOP, with the packed geometry being feed into the first input.

**Appendix B.12 RBD Export Preparations, Active and Inactive.**

- ACTIVE PIECES.
  - The pieces are then separated based on their @active status. A value of 1.0 is active, while a value of 0.0 is inactive.
  - An Attribute Delete SOP removes all attributes other than what is needed for Unreal.
    - Point attributes = * ^P ^v ^w
  - A Group Delete SOP removes all but the inside and outside face groups.
  - The geometry is then unpacked, and the velocity (v) and rotation (w) attributes are transferred.
- INACTIVE PIECES.
  - Separate the active from inactive pieces.
  - Feed the geometry into the Labs Destruction Cleanup SOP.
    - Simulation Rang = 1-1.
    - Insides.
      - Remove Insides = True.
      - Fuse Distance = 0.002
    - Cusping.
      - Cusp Polygons = True.
      - Cusp = Inside/Outside Together.

- Cusp Angle = 30.

- Advanced.

    - Transfer Attributes = uv N.

**Appendix B.13 Poly Reduction of Inactive Geometry.**

- Unpack geometry.
- The remaining inside faces are grouped and promoted to points.
- A PolyReduce SOP then reduces the model to the lowest amount possible.
    - Group = extrudeBase.
    - Target = Percentage of Input Polygon Count.
    - Percent To Keep = 61.
    - Use Only Original Point Positions = True.
    - Attributes = * ^N.
    - Hard Points = ray inside.
- An attribute transfer returns all the attributes from the unpacked geometry to the newly reduced mesh.
- An attribute delete and group delete, then clean up the mesh for export.

**Appendix B.14 Export RBD Alembic (ABC) File.**

- Merge the active and inactive simulation pieces.
- Clean any unnecessary attributes and groups.
- A transform node then resizes the model to a uniform scale of 100 to account for Houdini's world-scale being 1m while Unreal is 1cm.
- The simulation is then written out using a rop_alembic SOP.

**Appendix B.15 RBD Unreal Import/Setup and Texture Baking.**

- Drag the alembic file into Unreal's content browser.
- Inside the dialog box, set the following.
    - Alembic
        - Import Type = Geometry Cache (Experimental).
    - Sampling
        - Sampling Type = Per Frame
        - Frame Start = 1.
            - Must be changed to 1 since Houdini does not consider frame 0 as the starting frame.
        - Frame End = End of Sequence
    - Geometry Cache
        - Compressed Texture Coordinates Number of Bits = 16.
            - Improves UV and Lightmaps.
        - Motion Vectors = Import Abc Velocities as Motion Vectors.
            - Used in post processing volume motion blur.
    - Conversion
        - Preset = Autodesk 3ds Max
            - This is on account of the differences in Houdini and Unreal's coordinate systems[85].
- Drag the new Unreal asset into the scene and zero out its transformations.

---

[85] "Coordinate Systems," *Houdini Engine for Unreal*.

- Inside the sequencer, drag the new alembic object and attach the geometry cache.
- Assign a material to the RBD element in Unreal.
- Inside Houdini, use the BakeTexture Render Operator (ROP) to bake textures from a high poly model to a low poly model. Shown are just a few of the texture maps baked out by the BakeTexture ROP, BaseColor, Custom Noise, and Normals generated by the noise pattern, and below them, the lunar surface UVs.

**Appendix B.16 Unreal Light Building and SWARM Application.**

- Set the light build quality within Unreal.
- Click the drop-down menu next to Build.
- Select Build Lighting Only.
- Once Unreal shows that it has begun the light build, look for the swarm icon in the computer tray.
- Open SWARM and adjust the total number of cores/processors allowed to use.
- Reducing the number of processors allocated to SWARM fixed the heat issue but did increase build time.

**Appendix C.1 Installing Unreal Houdini Niagara Plug-In.**

- Ensure the latest version of SideFX Labs is installed, version 440 minimum.
- Download the version of the Niagara plug-in from SideFX's Github.
- Unzip the folder.
- On Windows, go to C:\Program Files\Epic Games**INSERT UE VERSION** \Engine\Plugins\FX
- Delete the "Houdini Niagara" folder that already exists.

- Copy and paste the newly downloaded “Houdini Niagara” folder into this location.

**Appendix C.2 Define Particle Source Base.**

- Object merge in the RBD simulation static mesh.
- Assign each piece an @pscale value based on their distance from the impact points.
- Remove any unwanted attributes using an attribdelete SOP.
- Remesh Branch.
    - Inside a ForEach SOP.
        - Unpack transferring all attributes.
        - Feed the unpacked geometry into a remesh node.
        - Link the remesh into input one of a primitive wrangle, with the original unpack node into the second input.

```
s@name = point(1,"name",0);
```

        - Feed the primitive wrangle into the first input of a point wrangle and the original unpack node into the second input.

```
f@pscale = point(1,"pscale",0);
```

    - Group the primitives under the name “debrisSource” using a Group SOP.
- Point Branch.
    - Blast the inactive geometry with a blast SOP.
    - Unpack the geometry transferring @name and @pscale.
    - Use a measure SOP set to primitives and perimeter to get the perimeter of each piece.
    - Promote the perimeter attribute from type primitive to point using an attribpromote SOP.

- A scatter node then creates points on the surface using the perimeter as a guide in the Density Attribute parameter.
- A point wrangle then gives each point an @id attribute.

  i@id = @ptnum;

  i@totalPTS = @numpt;
- Any unneeded attributes are removed.
- The points are put into the point group “pointSource” using a Group SOP.
- Both sources are then merged and cached.

**Appendix C.3 Create Animated Particle Source.**

- The cached RBD simulation points are read using an object merge SOP.
- Debris Source.
  - All but the remeshed model is removed using a blast SOP.
  - The model is feed into the first input of a transformpiece SOP with the RBD sim points into the second input.
  - The newly animated geometry is then passed to a debrissource node.
    - Edge Debris = 1,000,000.
    - Surface Debris = 0.
    - Search Radius = 0.5.
    - Search Points = 10.
    - Life Span = 1.
  - The resulting points are then grouped into “debrisPoints.”
- RBD Points.
  - All but the active scattered points are removed using a blast SOP.

- The points are feed into the first input of a transformpiece SOP with the RBD sim points into the second input.
- A pointjitter SOP adds noise into the animation point position of each frame.
- The points are grouped into “rbdPoints.”

- The two resulting point sources are merged together.
- A wrangle node compares the dot product of the point normal and their velocity. The points are then reaped based upon the resulting value. This culling method ensures that points only spawn from the rear-facing portion of the RBD piece.
- The points are then reaped based on their velocity. Ensuring points only spawn above a minimum velocity.

**Appendix C.4 Separating Point Sources and New Velocity.**

- The two-point groups are separated for further control on spawning and sim behavior.
- Each point source is passed through a pointreplicate SOP to have a larger point count to pull from at the end of the workflow.
- An attribute VOP uses an anti-alias flow noise to generate a “newVel” attribute.
- The “newVel” attribute is combined with the original point velocity to add noise and irregularity to the point sources.
- All particles are given the same defined mass, but this could be altered if needed.
- Declare Unreal @dead attribute using an attribute wrangle.

```
f@dead = 0.0;
```

**Appendix C.5 POP Simulation DOP Network Point Sourcing and Solver.**

With the @dead attribute created, the point sources are ready to pass into the particle DOP network.

(The following list will be transcribed flowing from left to right.)

- DOP Fetch Point Source and Create Solver
    - Create containing object using a POP Object node.
    - Feed POP Object node into the first input of a POP Solver.
    - Import point sources using POP Source node. Use multiple nodes for multiple sources.
        - Source
            - Emission Type = All Points.
            - Geometry Source = SOP = Input path to point source.
        - Birth
            - Life Expectancy = 5. (How long the particles should live)
            - Life Variance = 0.5. (How much, in seconds, to vary life)
            - Jitter Birth Time = Positive.
            - Interpolate Source = Back.
            - Interpolation Method = Use Point Velocity.
    - Merge the two POP sources.
    - Merge the POP source merge into the first input of a second merge node.
    - A POP Speed Limit is wired into the second input of the merge.
        - Maximum Speed = 250.
        - Maximum Spin = rad(95).
    - A POP drag is wired into the third input of the merge node to enhance the visual look of the sim.

- Air Resistance 0.005

- The second merge is then directed into a POP Force node.
  - Force = {1,1,1}.
  - Amplitude = 2.
  - Swirl Size = 5.
  - Roughness = 0.25.
- The POP Force then feeds into the third input of the POP Solver.

**Appendix C.6 POP Simulation DOP Network Collisions.**

(The following list will be transcribed flowing from left to right.)

- DOP Create Collisions
  - Fetch the created heightfield using a Terrain Object node.
    - Use Heightfield = True.
  - Feed the Terrain Object node into a merge node.
  - Grab the RBD simulation using a Static Object node.
    - Use Deforming Geometry = True.
    - Re-evaluate SOPs to Interpolate Geometry = True.
  - Connect the Static Object into the merge node.
  - Pass the merge node into a Static Solver.
  - Connect the Static Solver as the first input of a merge node with the previous level POP Solver as the second input. The new merge node now connects the two data stream levels.

**Appendix C.7 POP Simulation DOP Network Forces.**

- DOP Gravity and Output
    - Feed the merge node into a gravity node and set it to the gravity constant on the moon of -1.62.
    - Connect the gravity into the DOP network's final output node.

**Appendix C.8 POP Caching and Dead Culling.**

- The cached POP sim is passed through a uvtexture node to generate UV's from the X-axis in case they are needed.
- The result is feed into the SOP Solver node.
- Inside SOP Solver.
    - A probability is calculated using a point cloud vex function based on point density by connecting a point wrangle from the Object Merge Input One.
    - The probability wrangle is then passed into the next point wrangle.
        - The wrangle generates the point number of the POP Solver @id.
        - Then using the current point number and other attributes, it decides to set the @dead attribute.
        - The point wrangle then feeds into the first input of a second point wrangle. At the same time, the previous frame import is connected to the second input.
            - This wrangle then compares the @dead attribute to the previous frame, and if a point was dead on the previous frame and this frame, it is added to a point group named "remove."
        - The final result is returned to the solver's output node.

**Appendix C.9 POP Unreal Export.**

- The resulting @dead solver is connected to a Labs Niagara ROP node.
    - Start/End/Inc = 1,120,1.
    - Output Path = $HIP/FILEPATH/FILENAME.hbjson
    - Keep Attributes = P id time N v life type Cd Alpha dead pscale age uv prob

**Appendix C.10 POP Unreal Import.**

- Drag/Import the .hbjson point cache into the Unreal content browser.
- Right-click in the content browser and select, FX -> Niagara Emitter -> New emitter from a template -> Empty.
- Give the emitter a name.
- Right-click the emitter and select “Create a Niagara System.”

**Appendix C.11 Unreal Niagara System.**

- Niagara System Overview Node.
    - Create User Parameter, Houdini Point Cache Info.
        - Link point cache into user parameter.
    - System State.
        - Loop Behavior = Infinite.
        - Loop Duration = 5.
- Particle System.
    - Emitter Properties Stage.
        - Local Space = True.
        - Determinism = True.

- Sim Target = GPUCompute Sim.
- Fix Bounds = True.

- Emitter State Stage.
  - Life Cycle Mode = System.
  - Scalability Mode = System.
- Spawn Particles from Houdini Point Cache.
  - Houdini.PointCache.
    - Reference the System Overview User Parameter Houdini Point Cache.
- Particle Spawn Stage.
  - Sample Spawned Houdini Point Cache.
    - Reference the System Overview User Parameter Houdini Point Cache.
  - Set Particle Position.
    - Particle, Houdini, Position.
  - Set Particle SpriteSize.
    - Vector 2Dfrom Float.
    - Add Float.
      - Particle, Houdini, Pscale.
      - Add Float.
        - 150.
        - Random Range.
          - -100.
          - 75.

- Set Particle SubImageIndex.
    - Random Range Float.
        - 0.
        - 15.
- Particle Update Stage.
    - Particle State.
        - Kill Particle When Lifetime Has Elapsed = True.
    - Sample Houdini Point Cache.
        - Reference the System Overview User Parameter Houdini Point Cache.
    - Set Particle Position.
        - Particle Position = Particle, Houdini, Position.

**Appendix C.12 Houdini Attributes inside Unreal using a Niagara Module Script.**

- Right-click in the content browser and navigate to FX -> Niagara Module Script.
- Click the plus sign in the Map Get node and select New Houdini Point Cache Info.
- Drag off from the newly created Houdini Point Cache Info and select Get Point Value at Time by String.
- Type “dead” into the attribute string name of the Get Point Value at Time by String node.
- Click the plus icon for the Particle Attribute section and create an int32 attribute.
- Give the new int attribute the name “NID.” * The name is case sensitive *
- Rick-click the attribute name, and navigate to Change Namespace Modifier.
- Select the “Custom” namespace modifier and give it the name “Houdini.” * The name

is case sensitive *

- Click the plus icon in the Map Get node and now add the new NID attribute and connect this into the PointID input of the Get Point Value at Time by String node.
- Click the plus icon in the Map Get node, add the Emitter Age, and connect this into the Time input of Get Point Value at Time by String node.
- Create a new float particle attribute named “dead” with the “Houdini” namespace modifier.
- Click the plus icon on the Map Set node, add the newly created (Houdini) dead attribute, and connect its input to the Get Point Value at Time by String node.
- Finally, under the Script Detail panel, turn the Expose to Library to True.
- Apply and Save.

**Appendix C.13 Using the Houdini Dead Attribute to Kill Unreal Particles.**

- Particle System.
    - Particle Update.
        - Kill Particles.
            - Set Bool by Float Comparison.
                - A = Particles, Houdini, dead.
                - B = 0.
                - Comparison Type = A Greater Than B.

**Appendix C.14 Adding Particle Variety Inside Niagara.**

- Jitter Position.
    - Jitter Amount = 2.

- Jitter Offset = Random Vector.
- Jitter Delay = 0.

- Sprite Rotation Rate.
  - Rotation Rate.
    - Make Float from Vector = Particles, Houdini, w.
    - Channel = X.
- Dynamic Material Parameters.
  - Opacity.
  - Pos Rand Adjust (Used to add a random value to shaders).
- Set (Particles) SpriteAlignment.
  - Add Vector.
    - A = 0,0,1.
    - B = Rand Range Vector
      - 0,-0.5,0.
      - 0,0.5,0.
      - Evaluation Type = Spawn Only.

**Appendix C.15 Niagara Render Settings.**

- Particle System.
  - Sprite Render.
    - Material = POP Sprite Instance
    - Source Mode = Particles.
    - Alignment = Custom Alignment.
    - Facing Mode = Face Camera.

- Sub UV = 4x4.
- Visibility.
    - Enable Camera Distance Cull = True.
    - Min Camera Distance = 0.
    - Max Camera Distance = 25,000.

**Appendix D.1 Creating Volumes (Pyro) Movement Simulation Source.**

- POP SIM Source.
- Import the previous POP simulation result using an Object Merge node.
- Remove all point attributes except those necessary for the next steps.

    * ^age ^P ^uv ^id ^dead ^pscale ^prob ^v

- The points are culled based on their age and y position using a point wrangle.
- The points are given a new @pscale attribute to be used during volume rasterization.

    f@pscale = fit(f@pscale,0,1,chf("Min_Scale"),chf("Max_Scale"));

    f@pscale *= fit(f@prob,0,1,0.25,1);

- After generating the new attributes, the points are trailed.
- An add node converts the points to curves.
    - Polygon Tab.
        - By Group.
            - Add = By Attribute.
            - Attribute Name = id.
- The curves are resampled using a resample SOP.
- A UVTexture node, set to Rows and Columns, generates a UV attribute along the curve.

- Finally, a @ramp attribute is generated using the UV using an attribute wrangle SOP.
- POP SOURCE.
    - The source used during the POP simulation is brought in using an Object Merge SOP.
    - A new @pscale is generated using the before mentioned method.
    - The points are trailed using a trail SOP.
- The two sources are merged into one source.
- An SDF is generated from the points using the vdbfromparticles SOP.
- VDBreshapesdf is used to dilate the SDF.
- VDBSmooth softens the SDF volume.
- VDBReshapesdf then erodes the volume the same amount as the dilate function.
- The new SDF is feed into a pyrosource node.
    - Group Type = Points.
    - Mode = Volume Scatter.
    - Particle Separation = 0.2.
    - Particle Scale = 2.
    - Attributes = 1.
        - Attribute = Density

**Appendix D.2 Restoring Point Attributes.**

- Return needed attributes using attribute transfer SOP.

  Points = v ramp id
- Volume rasterize the density and velocity attributes.
    - Voxel Size = 0.1.

- Particle Scale = 0.75.
- Minimum Filter Size = 0.25.
- Velocity Blur = True.

- A primitive node converts the VDB from 32 to 16-bit float, saving a large amount of cache space.
- Cache the volume source.

**Appendix D.3 Adding Velocity Noise.**

- The volume source is passed through a volume vop SOP to add noise to the velocity.
    - An anti-alias flow noise uses position and time to generate noise added to the current velocity and multiplied by a constant scale.

**Appendix D.4 Volume Simulation DOP Network Sourcing and Solver.**

(The following list will be transcribed flowing from left to right.)

- DOP Fetch Volume Source and Create Solver
    - Create containing object using a smokeobject_sparse node.
    - Feed the smokeobject_sparse node into the first input of a smokesolver_sparse.
    - Import the volume source into the third input of the solver using a volume source node.
        - Initialize = Source Smoke.
        - Check that Enlarge Fields to Contain Sources is enabled.
    - The following micro-solvers are connected to the fourth input of the solver.
        - Gas Dissipate.

        - Diffusion = 0.1.
        - Evaporation Rate = 0.95.
    - Gas Vortex Confinement.
        - Confinement Scale = 3.
    - Gas Wind (Used for visual interest counter of scientific accuracy).
        - Wind Direction = -0.3,0,0.5.
    - Gas Turbulence.
    - Gas Disturb.
        - Strength = 25.
        - Reference Scale = 0.2.
    - Gas Disturb.
        - Strength = 50.
        - Threshold Range = 0.1,0.
        - Reference Scale = 0.1.

**Appendix D.5 Volume Simulation DOP Network Collisions.**

- DOP Create Collisions
    - Fetch the created heightfield using a Terrain Object node.
        - Use Heightfield = True.
    - Feed the Terrain Object node into a merge node.
    - Grab the RBD simulation using a Static Object node.
        - Use Deforming Geometry = True.
        - Re-evaluate SOPs to Interpolate Geometry = True.
    - Connect the Static Object into the merge node.

- Pass the merge node into a Static Solver.
- Connect the Static Solver as the first input of a merge node with the previous level Pyro Solver as the second input. The new merge node now connects the two data stream levels.

**Appendix D.6 Volume Simulation DOP Network Forces.**

- DOP Gravity and Output
    - Feed the merge node into a gravity node and set it to the gravity constant on the moon of -1.62.
    - Connect the gravity into the DOP network's final output node.

**Appendix D.7 Volume POP Simulation Setup.**

- Duplicate the POP simulation DOP network from the last section.
    - Wire a POP Advect by Volumes node into the network before the POP Speed Limit.
    - Feed the pyro simulation into the SOP input.
    - Velocity Scale = 0.7.
    - Advection Type = Update Force.
    - Air Resistance = 0.3.
        - NOTE: Air resistance denotes how much influence the volume should have on the particles, not actual atmospheric resistance.

**Appendix D.8 Volume POP Unreal Export.**

- The POP sim is feed into the dead solver.

- Particles in the “remove” group are deleted.
- A volume wrangle node remaps the original pyro simulation’s density from zero to one.

```
f@density = fit(f@density,0,9.3,0,1) * chf("Density_Mult");
```

- The point simulation is feed into the first input of a point wrangle with the volume wrangle in the second input. The wrangle then checks the particle's position to the density of the volume and assigns a color attribute.

```
float min = chf("Density_Min");
float density = volumesample(1,"density",v@P);
float age = 1 - fit(f@age,0,chf("Age_Max"),0,1);
float randomValue = rand(i@id);
density = fit(density,0,1,min,1);
if(f@age >= chf("Age_Max"))
{
        removepoint(0,@ptnum);
}
v@Cd = clamp((age * density) + randomValue,0,1);
```

- A Labs Niagara ROP node writes out the simulation for use inside Unreal.

**Appendix D.9 Volume POP Unreal Niagara Setup.**

- Create a new emitter inside the Niagara System previously created during *section 4.3, Particle Operators*.
- Niagara System Overview Node.
    - Create a second User Parameter, Houdini Point Cache Info.

- Link volume point cache into the user parameter.

- System State.
  - Loop Behavior = Infinite.
  - Loop Duration = 5.

- Particle System.
  - Emitter Properties Stage.
    - Local Space = True.
    - Determinism = True.
    - Sim Target = GPUCompute Sim.
    - Fix Bounds = True.
  - Emitter State Stage.
    - Life Cycle Mode = System.
    - Scalability Mode = System.
  - Spawn Particles from Houdini Point Cache.
    - Houdini.PointCache.
      - Reference the System Overview User Parameter Houdini Volume Cache.
  - Particle Spawn Stage.
    - Sample Spawned Houdini Point Cache.
      - Reference the System Overview User Parameter Houdini Volume Cache.
    - Set Particle Position.
      - Particles, Houdini, Position.

**Appendix D.10 Volume POP Unreal Niagara Setup.**

- Set Particle SpriteSize.
  - Vector 2Dfrom Float.
  - Add Float.
    - Particle, Houdini, Pscale.
    - Add Float.
      - 2500.
      - Random Range Float.
        - -1000.
        - 1000.
- Particle Update Stage.
  - Particle State.
    - Kill Particle When Lifetime Has Elapsed = True.
  - Sample Houdini Point Cache.
    - Reference the System Overview User Parameter Houdini Volume Cache.
  - Kill Particles (Reuses the Niagara Module Script in *section 4.3, Particle Operators*).
    - Set Bool by Float Comparison.
      - A = Particles, Houdini, dead.
      - B = 0.
      - Comparison Type = A Greater Than B.

**Appendix D.11 Volume POP Unreal Sprite Alignment.**

- Set (Particles) SpriteAlignment.
    - Add Vector.
        - A = 0,0,1.
        - B= Rand Range Vector.
            - 0,-1,0.
            - 0,1,0.
            - Evaluation Type = Spawn Only.

**Appendix D.12 Volume POP Unreal Dynamic Material Parameters.**

- Dynamic Material Parameters.
    - Opacity.
        - Float remap Values.
            - Add Float.
                - A = Float From Curve.
                - B = Random Range Float.
                    - A = -10.
                    - B = 10.
                    - Evaluation Type = Spawn Only.
            - Mininput = -10.
            - MaxInput = 11.
            - MinOutput = 0.65.
            - MaxOutput = 1.1.
    - Pos Rand Adjust.
        - Random Range Float.

        - -50.
        - 50.
        - Evaluation Type = Spawn Only.
    - Time.
        - Random Range Float.
            - -10.
            - 10.
            - Evaluation Type = Spawn Only.

**Appendix D.13 Volume Sprite Simulation.**

- Create a sphere SOP to be the base of the volume source.
- A mountain SOP adds noise to the sphere each frame.
    - Height = 1.2.
    - Element Size = 1.09.
    - Offset = $FF /2.
- VDB From Polygons generates a fog volume.
- A scatter SOP puts points inside the new volume.
    - Force Total Count = fit(rand($FF),0,1,1,3).
- A point wrangle generates a new @pscale attribute.
- An attribute randomize SOP then creates a quaternion @orient attribute.
- A copy to point SOP then copies new spheres onto the points.
- These spheres are then given noise using another mountain SOP.
    - Height = 2.43.
    - Element Size = 1.09.

- Offset = $FF / 10.

- A pyro source node then generates the needed attributes for simulation.
- A point wrangle calculates the velocity of the point based on its distance from the center.

```
vector center = set(0,0,0);
v@v = normalize(v@P - center);
```

- Attribute VOPS using an anti-alias flow noise adds noise to the velocity.
- The points are rasterized into a volume and passed to a pyro simulation.
- The simulation uses the same settings as the previous driving the particle motion.
    - The only difference is the lack of collision objects.

**Appendix D.14 Loopable Simulation Texture.**

- Read in the cached volume simulation.
- Use a timeshift node to adjust the simulation's midpoint to the first frame.
    - Frame = $FF + ((ch("../Frames/sequence_length")) / 2).
    - Clamp = Camp To Last.
    - End Frame = 120.
        - The end frame is set to 120 to ensure that the timeshift never goes outside the total cached frames available.
- FIRST BRANCH.
    - A volume VOP uses the fit and power functions to adjust the total density of the volume.
        - The exponent of the power function is animated between one and five to slowly subtract the volume over time.

- The volume VOP is feed into the first input of a volume mix node. (TO BE CONTINUED)

- SECOND BRANCH.
  - The first timeshift node connects to a second timeshift, so the first frame begins at the midpoint.
    - $FF - (ch("../Frames/sequence_length"))
    - Clamp = Clamp To Last.
    - End Frame = 120.
  - The same volume VOP from the first branch is used, but the power exponent value is animated in reverse — the value starting at five and decreasing to one.
  - The volume VOP is feed into the second input of a volume mix node.
- The volume mix node is then used to blend between the two different volume branches to create the looping texture.
  - Mix Method = Blend.
  - The Blend value is animated between the two volumes near the end of one and the beginning of the other.
- A volume wrangle then adjusts the volume's total density value.

  f@density *= chf("Density");
- A standard pyro shader is applied to the volume.

**Appendix D.15 Unreal Material Time Component.**

- Particle Random Value generates a random number per Niagara particle.
- The random value is clamped between 0 and 0.5.

- The clamped value is then added to the exposed material parameter TimeScale.
- If Comparison.
    - A = TimeScale.
    - B = Float Constant, 0.
    - A > B = The randomized TimeScale.
    - A == B = TimeScale.
    - A < B = Float Constant, 0.
- The result from the If function is feed into another clamp between zero and ten.
- An Unreal Time node is multiplied by the new TimeScale value.
- A dynamic parameter node then adds any time changes from Niagara to the material.
- This value is then multiplied by the Houdini time value.
    - 24 FPS divided by 96 total frames of the animation.
- The update TimeScale is then run through a Frac node to gather just the zero to one value.

**Appendix D.16 Unreal Flipbook Function.**

- Flipbook.
    - Animation Phase = Frac node with the TimeScale.
    - Number of Rows = 12.
    - Number of Columns = 8.
    - UVs = TexCord(0) node.

**Appendix D.17 Unreal Smart Lighting Setup.**

- Create a Camera Direction Vector node.
- Break the camera vector apart using a Break Out Float 3 Components node.
- Feed the R and G channels into two dot product functions. R = X, G = Y, B = Z.
    - The B (Z) represents the Up vector component inside Unreal.
- Place down a vector parameter and name it “Light Position.”
- Normalize the output from the light position parameter.
- Break the light position vector apart using a Break Out Float 3 Componenets node.
- Feed the R (X) channel into the dot product function with the camera vector's R (X) channel.
- Feed the B (Z) channel into the dot product function with the camera vector’s G (Y) channel. This is due to Houdini being a right-handed Y-up coordinate system compared to Unreal’s left-handed Z-up system[86].
- Feed the G (Y) channel into a new dot product function comparing to a constant up value of {0,0,1}.
- Each dot product is then passed to a Remap Value Range node and re-written from the original value of minus one to one to a new value of zero to one.
- The returned value of the dot products is then used to lerp between each RGB channel of the texture. A value of zero meaning the light is in the opposing direction of the camera, thus behind the volume sprite.

**Appendix D.18 Unreal Material Artist Controls.**

- Each RGB lighting channel is multiplied by a user parameter to adjust the intensity of

[86] “Coordinate Systems,” *Houdini Engine for Unreal*.

each light.

- The lighting layers are reassembled using add nodes.
- Image contrast is remapped between zero and one based on user inputted min and max values.
- The sprite color is multiplied by the particle color node if Niagara needs to change the material.
- The resulting color is then multiplied again by a user inputted color.
- Lastly, the user can input a finite multiplier to control the sprite's intensity.

**Appendix D.19 Unreal Material Smart Opacity Setup.**

- The Texture Sample's alpha is brought in and multiplied by a Dynamic Parameter from the Niagara system.
- The alpha is then multiplied by the particle alpha.
- A user input parameter multiplies the alpha to adjust the intensity.
- Finally, the alpha is multiplied one last time by the camera distance culling.
- Camera Distance Opacity Cull.
    - A Dynamic Parameter from Niagara brings in a random value on each particle.
    - The random value is added to the particle's original position.
    - This position data is then subtracted from the camera's current position. The result is a vector with direction and magnitude, which can be measured.
    - The magnitude (length) of the vector is measured using a Vector Length node.
    - The resulting length is then returned as an absolute since being in negative or positive space makes no difference.

- The absolute is remapped between zero and one based on a minimum and maximum distance cutoff value.

- Any particle closest to the camera has its opacity multiplied by one, while every particle that moves further and further away gradually gets multiplied by a value closer to zero.

**Appendix E.1 Houdini Base FBX Export.**

Fabric_PanelsKB3D_LNB_BldgLG_F_BuildingA
{Fabric_Panels}{KB3D_LNB_BldgLG_F_BuildingA}

Material Name = Fabric_Panels
Building Name = KB3D_LNB_BldgLG_F_BuildingA

*Fig. E.1.1 Houdini Unreal Base FBX Name Attribute Hierarchy*

- Unpack the laid-out base elements.
- A primitive wrangle creates the first component of the @name attribute using the applied material.

```
string material = s@shop_materialpath;
material = strip(material,"/mat/");
setprimattrib(0,"name",@primnum,material,"set");
```

- Another primitive wrangle then takes the newly create @name attribute and combines it with the intrinsic primitive group.

```
string groups[] = detailintrinsic(0, "primitivegroups");
int loop = len(groups);
for(int i = 0; i < loop; i++)
```

```
{
        int check = inprimgroup(0,groups[i],@primnum);
        if(check == 1)
        {
                s@name = s@name + groups[i];
                break;
        }
}
```

**Appendix E.2 Using Unreal HDRI Backdrop.**

- Inside the Unreal project, open Edit > Plugins.
- Search for HDRIBackdrop.
- Enable the plug-in and restart the project.
- In Unreal's Actor browser, search for HDRI Backdrop.
- Drag the backdrop into the Unreal scene.
- Adjust the size of the sphere to cover the entire scene.
- Drag the star map HDRI into the backdrop's cubemap parameter.
    - The used HDRI was sourced from NASA's Scientific Visualization Studio Deep Star Maps 2020[87] and saved as an HDRI inside Photoshop.
- Adjust the project center z attribute to reduce HDRI stretching.

[87] Ernie Wright, "Deep Star Maps 2020," *NASA Scientific Visualization Studio*, September 9, 2020, accessed July 21, 2021, https://svs.gsfc.nasa.gov/4851.

**Appendix E.3 Creating Unreal Earth Model.**

- Create a sphere SOP set to Polygon Mesh.
- A uvproject node, set to Polar, unwraps the sphere so that the North and South poles are the seams.
- The normal SOP generates normals on the sphere.
- The sphere is transformed by a uniform scale of ten to account for the world scale difference between Houdini and Unreal.
- The Labs soften normals node adjusts the normals.
- The model is imported into Unreal, and the Earth material is applied.

**Appendix E.4 Lunar Rocks using Unreal Foliage.**

- Create rock models within Houdini.
- Import the static meshes into Unreal.
- Select Foliage under the Modes dropdown menu in Unreal.
- Drag and drop the static meshes from the content browser into the foliage type directory.
- Ensure the foliage tool is in Paint mode.
- Adjust the density and radius of the brush.
- Inside the paint settings of each model, set the following.
    - Painting.
        - Single Instance Mode Override = True.
    - Placement.
        - Z Offset = Min -0.3, Max -0.1.
        - Align to Normal = True.

- Align Max Angle = 10.0.
- Random Yaw = True.
- Random Pitch Angle = 10.0.

- Instance Settings.
  - Mobility = Static.
  - Affect Dynamic Indirect Lighting = True.
  - Light Map Resolution = 32.

**Appendix E.5 Unreal Exponential Height Fog.**

- Under the Place Actor panel, find the Visual Effects Tab.
- Left-click and drag the Exponential Height Fog into the scene.
- Position the actor to adjust the fog location and use the Fog Height Falloff to control the fog further[88].
- Adjust the Fog Density to have a very low value to be unintrusive.
- Set the start distance, so the fog does not directly influence the lunar surface but is applied to the HDRI Backdrop.
- Set the Directional Inscattering Color to create color within the volume.

**Appendix E.6 Unreal Lightmass Importance Volume.**

- Open the Unreal scene file.
- In the Actors panel under Volumes, drag and drop a Lightmass Importance Volume

[88] "Exponential Height Fog User Guide," *Unreal Engine Docs*, Ver. 4.26, last updated 2021, accessed July 26, 2021, https://docs.unrealengine.com/4.26/en-US/BuildingWorlds/FogEffects/HeightFog/.

into the scene.

- Adjust the position and scale of the volume to encompass the entirety of the FX.
- Click the drop-down arrow next to the Build button.
- Under Lighting Quality, set the Quality Level to production.
- Click Build and let Unreal build the lighting, reflection, and other needed elements.
- Total Build Time = ~00:05:30.

**Appendix E.7 Unreal Post Process Volume.**

- In the Place Actors panel, find the Volumes tab.
- Left-click drag a Post Process Volume into the Unreal scene.
- Turn on Infinite Bounds.
- Lens.
    - Exposure.
        - Metering Mode = Auto Exposure Histogram.
        - Exposure Compensation = 1.0.
    - Chromatic Aberration.
        - Intensity = 0.5.
        - Start Offset = 0.823.
    - Image Effects.
        - Vignette Intensity = 0.4.
    - Depth of Field.
        - Focal Distance = 500.0.
        - Depth Blur km for 50% = 6.0.
        - Depth Blur Radius = 5.0.

- Color Grading.
    - White Balance.
        - Temp = 7100.0.
    - Global.
        - Saturation.
            - H = 0.0.
            - S = 0.0.
            - V = 1.3.
            - Y = 1.0.
        - Contrast.
            - H = 0.0.
            - S = 0.0.
            - V = 1.27.
            - Y = 1.0.

**Appendix E.8 Creating Unreal LUTs Inside Nuke.**

- Open Nuke.
- Read in the Unreal neutral loop-up table .png.
- Use a vectorfield node to apply a .cube file LUT.
- Write out the new Unreal LUT as a .png or .tga file.

**Appendix E.9 Unreal Post Processing Utilizing a LUT.**

- Misc.
    - Scene Color Tint = 0.58.

- Color Grading LUT Intensity = 0.7.
- Color Grading LUT = True.

**Works Cited**

"Alembic File Importer." *Unreal Engine Docs*. Ver. 4.26. Last updated 2021. Accessed April 12, 2021. https://docs.unrealengine.com/4.26/en-US/WorkingWithContent/Importing/AlembicImporter/.

Ambrosiussen, Paul. "Optimizing Destruction Simulations for Real-Time Solutions." *SideFX Houdini*. Ver. 16. Last updated April 6, 2017. Accessed April 7, 2021. https://www.sidefx.com/tutorials/optimizing-destruction-simulations-for-real-time-solutions/.

"Assets." *Houdini Engine for Unreal*. Ver. 18.5. Last updated 2021. Accessed April 2, 2021. https://www.sidefx.com/docs/unreal/_assets.html.

"Bake Texture." *SideFX Houdini*. Ver. 18.5. Last updated 2021. Accessed July 22, 2021. https://www.sidefx.com/docs/houdini/nodes/out/baketexture.html.

Bath, Addison. "Virtual Production." In *The VES Handbook of Visual Effects Industry Standard VFX Practices and Procedures* 3rd ed. Edited by Jeffrey A. Okun and Susan Zwerman, 57-60. New York: Routledge, 2021.

Brucks, Ryan. "Authoring Pseudo Volume Textures." *Shader Bits*. October 18, 2016. Accessed June 12, 2021. https://shaderbits.com/blog/authoring-pseudo-volume-textures.

Brucks, Ryan. "Creating a Volumetric Ray Marcher." *Shader Bits*. November 16, 2016. Accessed June 12, 2021. https://shaderbits.com/blog/creating-volumetric-ray-marcher.

Cloward, Ben. "Flipbook Animation – UE4 Material 101 – Episode 5." December 19, 2019. Video, 16:48. https://youtu.be/ZWAF_f2aP9s.

"Coordinate Systems." *Houdini Engine for Unreal*. Ver. 18.5. Last updated 2021. Accessed July 25, 2021. https://www.sidefx.com/docs/unreal/_coordinate_system.html.

"Creating Looping Realtime FX Using Houdini | Free Webinar Replay." Rebelway. Accessed January 28, 2021. Video, 1:17:31. Accessed June 18, 2021. https://youtu.be/0NgNQlpmPlQ.

*Cube LUT Specifications Version 1.0*. Adobe, 2013. Accessed August 1, 2021. https://wwwimages2.adobe.com/content/dam/acom/en/products/speedgrade/cc/pdfs/cube-lut-specification-1.0.pdf.

Eriksen, Kaare. "Why Virtual Production Benefits Filmmaking in a Pandemic." *Variety*. May 13, 2021. Accessed September 3, 2021. https://variety.com/vip/why-virtual-production-benefits-filmmaking-in-a-pandemic-1234971940/.

"Exponential Height Fog User Guide." *Unreal Engine Docs*. Ver. 4.26. Last updated 2021. Accessed July 26, 2021. https://docs.unrealengine.com/4.26/en-US/BuildingWorlds/FogEffects/HeightFog/.

Failes, Ian. "How Previs Has Gone Real-Time." *VFX Voice*. April 1, 2020. Accessed September 27, 2021. https://www.vfxvoice.com/how-previs-has-gone-real-time/.

Farris, Jeff. "Forging New Paths for Filmmakers on 'The Mandalorian.'" *Epic Games*. February 20, 2020. Accessed October 3, 2021. https://www.unrealengine.com/en-US/blog/forging-new-paths-for-filmmakers-on-the-mandalorian.

"FBX Content Pipeline." *Unreal Engine Docs*. Ver. 4.26. Last updated 2021. Accessed March 3, 2021. https://docs.unrealengine.com/4.26/en-US/WorkingWithContent/Importing/FBX/.

The Focus. "Virtual Production 101: How Does It Work, How Can It Revolutionise VFX?" *Medium*. October 18, 2019. Accessed September 28, 2021. https://medium.com/@thefocus/virtual-production-101-how-does-it-work-how-can-it-revolutionise-vfx-1c7e80ade0f2.

"Foliage Tool." *Unreal Engine Docs*. Ver. 4.26. Last updated 2021. Accessed August 3, 2021. https://docs.unrealengine.com/4.27/en-US/BuildingWorlds/Foliage/.

Fowler, Deborah R. "Houdini Height Fields into Unreal." *Deborah Fowler Houdini Resources*. Last updated January 31, 2021. Accessed March 6, 2021. http://deborahfowler.com/HoudiniResources/HoudiniHeightFieldsToUnreal.html.

Fowler, Deborah R. "Phyllotactic Pattern." *Deborah Fowler Math for VSFX*. Last updated February 13, 2021. Accessed March 8, 2021. http://deborahrfowler.com/MathForVSFX/Phyllotaxis.html.

"Generating Lightmap UVs." *Unreal Engine Docs*. Ver. 4.26. Last updated 2021. Accessed July 28, 2021. https://docs.unrealengine.com/4.26/en-US/WorkingWithContent/Types/StaticMeshes/AutoGeneratedLightmaps/.

"GPU Profiling." *Unreal Engine Docs.* Ver. 4.26. Last updated 2021. Accessed July 31, 2021. https://docs.unrealengine.com/4.26/en-US/TestingAndOptimization/PerformanceAndProfiling/GPU/.

Harvey, Ian. "The Mill SDGM Real-Time FX Switching Gears to Niagara." *Ian Harvey*. Last updated March 9, 2020. Accessed May 24, 2021. https://www.ianharveyvfx.com/post-pme0g/the-mill-sdgm-real-time-fx-switching-gears-to-niagara.

"HDRI Backdrop." *Unreal Engine Docs*. Ver. 4.26. Last updated 2021. Accessed August 1, 2021. https://docs.unrealengine.com/4.26/en-US/BuildingWorlds/LightingAndShadows/HDRIBackdrop/.

Hogg, Trevor. “Virtual Production Takes a Big Step Forward.” *VFX Voice*. January 28, 2021. Accessed September 4, 2021. https://www.vfxvoice.com/virtual-production-takes-a-big-step-forward/.

“Houdini Engine, Unreal Plug-In.” *SideFX Houdini*. Accessed April 2, 2021. https://www.sidefx.com/products/houdini-engine/plug-ins/unreal-plug-in/.

“Houdini Niagara Plugin – UE4.26.” *Sideeffects Github*. Ver. 4.26. Last updated January 22, 2021. Accessed April 20, 2021. https://github.com/sideeffects/HoudiniNiagara/releases.

“Introduction to Alembic | Live Training | Unreal.” Unreal Engine. September 21, 2017. Video, 55:45. https://youtu.be/rDHxxk0y2D4.

Kadner, Noah. *The Virtual Production Field Guide*. Ver. 1. Epic Games, 2019. https://cdn2.unrealengine.com/vp-field-guide-v1-3-01-f0bce45b6319.pdf.

Kaufman, Debra. “New Virtual Technologies Remake VFX’s Future Pipeline.” *VFX Voice*. January 2, 2020. Accessed October 6, 2021. https://www.vfxvoice.com/new-virtual-technologies-remake-vfxs-future-pipeline/.

“Landscapes.” *Houdini Engine for Unreal*. Ver. 18.5. Last updated 2021. Accessed April 2, 2021. https://www.sidefx.com/docs/unreal/_landscapes.html.

“Landscape Technical Guide.” *Unreal Engine Docs*. Ver. 4.26. Last updated 2021. Accessed March 3, 2021. https://docs.unrealengine.com/4.27/en-US/BuildingWorlds/Landscape/TechnicalGuide/.

Lewis, Hamilton. “What Is Previs and Other Forms of Visualization.” In *The VES Handbook of Visual Effects Industry Standard VFX Practices and Procedures* 3rd ed. Edited by Jeffrey A. Okun and Susan Zwerman, 36-38. New York: Routledge, 2021.

“Lightmass Basics.” *Unreal Engine Docs*. Ver. 4.26. Last updated 2021. Accessed April 15, 2021. https://docs.unrealengine.com/4.26/en-US/RenderingAndGraphics/Lightmass/Basics/.

“LiViCi Presentation at Performance & XR Symposium.” Shocap Entertainment, Ltd. October 6, 2020. Video, 6:29. https://youtu.be/Od61KOfQkFE.

“LiViCi Presentation.” *Shocap Entertainment*. October 7, 2020. https://www.shocap.com/projects/livicipresentation.

Lyndon, Mike. “Creating a Niagara Emitter from Scratch.” SideFX Houdini. April 29, 2020. Video, 7:53. https://www.sidefx.com/tutorials/houdini-to-ue4s-niagara/.

Lyndon, Mike. “Houdini Niagara Installation.” SideFX Houdini. April 29, 2020. Video, 6:45. https://www.sidefx.com/tutorials/houdini-to-ue4s-niagara/.

Lyndon, Mike. “Houdini to Niagara Workflow.” SideFX Houdini. April 29, 2020. Video, 8:47. https://www.sidefx.com/tutorials/houdini-to-ue4s-niagara/.

Lyndon, Mike. “Sampling Custom Attributes.” SideFX Houdini. April 29, 2020. Video, 10:08. https://www.sidefx.com/tutorials/houdini-to-ue4s-niagara/.

“Materials.” *Houdini Engine for Unreal*. Ver. 18.5. Last updated 2021. Accessed April 2, 2021. https://www.sidefx.com/docs/unreal/_materials.html.

“Performance and Profiling.” *Unreal Engine Docs*. Ver. 4.26. Last updated 2021. Accessed July 30, 2021. https://docs.unrealengine.com/4.26/en-US/TestingAndOptimization/PerformanceAndProfiling/.

“Post Process Effects.” *Unreal Engine Docs*. Ver. 4.26. Last updated 2021. Accessed July 28, 2021. https://docs.unrealengine.com/4.26/en-US/RenderingAndGraphics/PostProcessEffects/.

“Profiling Common Trouble Areas with Particle Systems.” *Unreal Engine Docs*. Ver. 4.26. Last updated 2021. Accessed July 31, 2021. https://docs.unrealengine.com/4.26/en-US/RenderingAndGraphics/ParticleSystems/Optimization/Results/.

“Realtime FX for Games & Cinematics.” *Rebelway*. Accessed February 18, 2021. https://www.rebelway.net/realtime-fx-for-games-and-cinematics.

Seymour, Mike. “Art of LED Wall Virtual Production, Part One: ‘Lessons from the Mandalorian'.” *FX Guide*. March 4, 2020. Accessed October 9, 2021. https://www.fxguide.com/fxfeatured/art-of-led-wall-virtual-production-part-one-lessons-from-the-mandalorian.

Seymour, Mike. “Art of (LED Wall) Virtual Production Sets, Part Two: ‘How You Make One.’” *FX Guide*. March 9, 2020. Accessed October 9, 2021. https://www.fxguide.com/fxfeatured/art-of-led-wall-virtual-production-sets-part-two-how-you-make-one/?highlight=LED%20Wall%20Virtual%20Production.

Seymour, Mike. “Virtual Production Guide: Kaya Jabar, Third Floor.” *FX Guide*. July 25, 2019. Accessed August 29, 2021. https://www.fxguide.com/quicktakes/virtual-production-guide-kaya-jabar-third-floor/.

“SideFX Labs.” *SideFX Houdini*. Accessed April 5, 2021. https://www.sidefx.com/products/sidefx-labs/.

Świerad, Oskar. “Unreal’s Rendering Passes.” *Unreal Art Optimization*. Last updated August 8, 2019. Accessed July 31, 2021. https://unrealartoptimization.github.io/book/profiling/passes/.

"Technicolor's MPC Film and the Filmmakers of The Lion King Bring the Beloved Characters from a Disney Classic Back to the Screen – Like You've Never Seen Before." *Technicolor*. 2019. Accessed October 1, 2021. https://www.technicolor.com/thelionking#virtual-production.

*Technicolor Quick Reference Virtual Production Glossary*. Ver. 1. Technicolor, January 1, 2019. https://www.technicolor.com/sites/default/files/2019-07/20190726-Virtual-Production-Glossary-v1-web.pdf.

"Terrain Party." Last updated 2021. Accessed March 2, 2021. https://terrain.party/.

"The Lion King – Breaking Ground with Virtual Production." *Technicolor*. August 7, 2019. Video, 4:50. https://youtu.be/shD-dQ1yPdk.

"Unreal Niagara: Sub UV Index Particles." AndrewWeirArt. December 30, 2018. Video, 13:20. https://youtu.be/Hh4wxRocnQc.

"Unreal Online Learning Courses." *Epic Games Online Learning*. Last updated 2021. Accessed February 15, 2021. https://www.unrealengine.com/en-US/onlinelearning-courses.

"Unreal Swarm." *Unreal Engine Docs*. Ver. 4.26. Last updated 2021. Accessed July 30, 2021. https://docs.unrealengine.com/4.26/en-US/RenderingAndGraphics/Lightmass/UnrealSwarmOverview.

"Using Lookup Tables (LUTs) for Color Grading." *Unreal Engine Docs*. Ver. 4.26. Last updated 2021. Accessed August 2, 2021. https://docs.unrealengine.com/4.26/en-US/RenderingAndGraphics/PostProcessEffects/UsingLUTs/.

"Vertex Animation Textures in Unreal." *SideFX Houdini*. Updated September 7, 2021. Accessed September 17, 2021. https://www.sidefx.com/tutorials/vertex-animation-textures-for-unreal/.

"Virtual Production at DNEG." *DNEG*. Accessed September 23, 2021. https://www.dneg.com/virtual-production/.

"Virtual Production: LED Stage Creative Test at Dimension Studios." Dimension Studio. August 6, 2020. Video, 7:54. https://youtu.be/SKTg83Pgfas.

"Visibility and Occlusion Culling." *Unreal Engine Docs*. Ver. 4.26. Last updated 2021. Accessed May 9, 202. https://docs.unrealengine.com/4.26/en-US/RenderingAndGraphics/VisibilityCulling/.

Wright, Ernie. "Deep Star Maps 2020." *NASA Scientific Visualization Studio*. September 9, 2020. Accessed July 21, 2021. https://svs.gsfc.nasa.gov/4851.

Zerouni, Craig. “Particles.” In *The VES Handbook of Visual Effects Industry Standard VFX Practices and Procedures* 3rd ed. Edited by Jeffrey A. Okun and Susan Zwerman, 571-575. New York: Routledge, 2021.

Zerouni, Craig. “Rigid-Body Dynamics.” In *The VES Handbook of Visual Effects Industry Standard VFX Practices and Procedures* 3rd ed. Edited by Jeffrey A. Okun and Susan Zwerman. 576-578. New York: Routledge, 2021.