

5-2016

# Teleportation in Misfit Mice - Combining Simulation and Attribute-Driven Animation

Alexander D. Rizeakos

Clemson University, arizeak@g.clemson.edu

Follow this and additional works at: [https://tigerprints.clemson.edu/all\\_theses](https://tigerprints.clemson.edu/all_theses)

---

## Recommended Citation

Rizeakos, Alexander D., "Teleportation in Misfit Mice - Combining Simulation and Attribute-Driven Animation" (2016). *All Theses*. 2395.

[https://tigerprints.clemson.edu/all\\_theses/2395](https://tigerprints.clemson.edu/all_theses/2395)

This Thesis is brought to you for free and open access by the Theses at TigerPrints. It has been accepted for inclusion in All Theses by an authorized administrator of TigerPrints. For more information, please contact [kokeefe@clemson.edu](mailto:kokeefe@clemson.edu).

# TELEPORTATION IN *Misfit Mice* - COMBINING SIMULATION AND ATTRIBUTE-DRIVEN ANIMATION

---

A Thesis  
Presented to  
the Graduate School of  
Clemson University

---

In Partial Fulfillment  
of the Requirements for the Degree  
Master of Fine Arts  
Digital Production Arts

---

by  
Alexander D. Rizeakos  
May 2016

---

Accepted by:  
Dr. Jerry Tessendorf, Committee Chair  
Dr. Donald House  
Dr. Timothy Davis

# Abstract

Conceptualizing, developing, and producing controllable effects is both a technically and artistically challenging endeavor. This paper explains the approach and final implementation of the teleportation effect seen in the short film, *Misfit Mice*. The concept was developed based on reference and specific intentions of the effect. The final implementation for the motion of the particles was created in two distinct steps: advection by a simulated fluid and attribute driven animation. Particle rendering was also driven by attributes. The methods and approach can be applied to the development of other effects and the final implementation has some potential for reuse.

# Acknowledgments

I would like to thank all of the students that participated in the 8600 course during the spring of 2015: Wyndham Batton, Kevin Boggs, Jordan Gestring, Gina Guerrero, Karl Jahnke, Dylan Swift, Chuqiao Wang, and Junyan Wang. Without your feedback and support, nothing would have been made that semester. In addition, I would like to thank Jon Barry for his feedback and support during the development and implementation of the teleportation effect. I would also like to thank the mentors from Moondog Animation, in particular Ben Davis, whose guidance helped shape the short film and insight made it more successful. Finally, I would like to thank my advisor, Dr. Jerry Tessendorf, and my committee, Dr. Donald House and Dr. Timothy Davis for their instruction and guidance throughout my time at Clemson.



# Table of Contents

Title Page . . . . .	i
Abstract . . . . .	ii
Acknowledgments . . . . .	iii
1 Introduction . . . . .	1
2 Fluid Simulation . . . . .	6
3 Particle Advection . . . . .	13
4 Attribute-Based Animation . . . . .	18
5 Rendering . . . . .	26
6 Conclusions and Discussion . . . . .	32
References . . . . .	37

# Chapter 1

## Introduction

The design and implementation of any effect can be a large undertaking. The first priority is to fully understand the intent of the effect. To do this, it is important to understand its context. The tone of *Misfit Mice* was intended to be dark and suspenseful. In a word, dangerous. In Figure 1.1 a render of the final film and below is the back story can be seen:

The mice have been assembled by a young scientist/inventor that has issues in his personal life. Perhaps he is bullied at school or comes from a broken home or both, but whatever it is, it causes him to retreat from the real world and into the world of super heroes. His obsession with comic books and super powers comes from his need to exert some control over his life. Each of the superpowers of the mice fall into two categories: fight or flight. The youngster develops the ice, fire, super strength, and telekinesis powers in order to physically conquer the problems in his life and they symbolize the power and aggression he wishes to possess, while the teleportation mouse is an extension of the way the creator feels inside and his tendency towards withdrawal and retreat.

The dark tone of the short film led to the idea that the teleportation effect would seem violent or explosive, almost as if it were ripping the mouse apart. The main two pieces of reference used for the design of the effect were the Nightcrawler BAMF effect from the film, *X2 - X-Men United* [1], and the apparate effect from *Harry Potter and the Deathly Hallows: Part 1* [2] films. The BAMF effect has some of the explosive qualities that were desired, while the apparate effect clearly indicates teleportation. Figure 1.2 shows Nightcrawler directly before teleporting and figures



Figure 1.1: Misfit Mice



Figure 1.2: Nightcrawler Before Teleport [1]



Figure 1.3: Teleport 1 [1]



Figure 1.4: Teleport 2 [1]



Figure 1.5: Harry Apparate [2]



Figure 1.6: Hermione Apparate 1 [2]



Figure 1.7: Hermione Apparate 2 [2]

1.3 and 1.4 show the particle system that is left directly after his teleportation. Figures 1.5, 1.6, and 1.7 show the apparate effect that inspired the swirling vortex .

It was clear based on the reference that the teleportation effect had to be broken into two distinct actions and implementations. In the end, simulation proved the best way to get an explosive look and attribute-driven animation provided the control necessary to visually sell the act of teleportation.

The final implementation contained four steps: fluid simulation, particle advection, attribute based animation, and rendering. The fluid simulation was sourced from animated geometry and the fluid's density and velocity grids were cached and used to advect a particle system. The advected particles were then animated using a combination of attributes and keyframes. Finally, the particle system was rendered additively. See Figure 1.8 for a visual explanation.

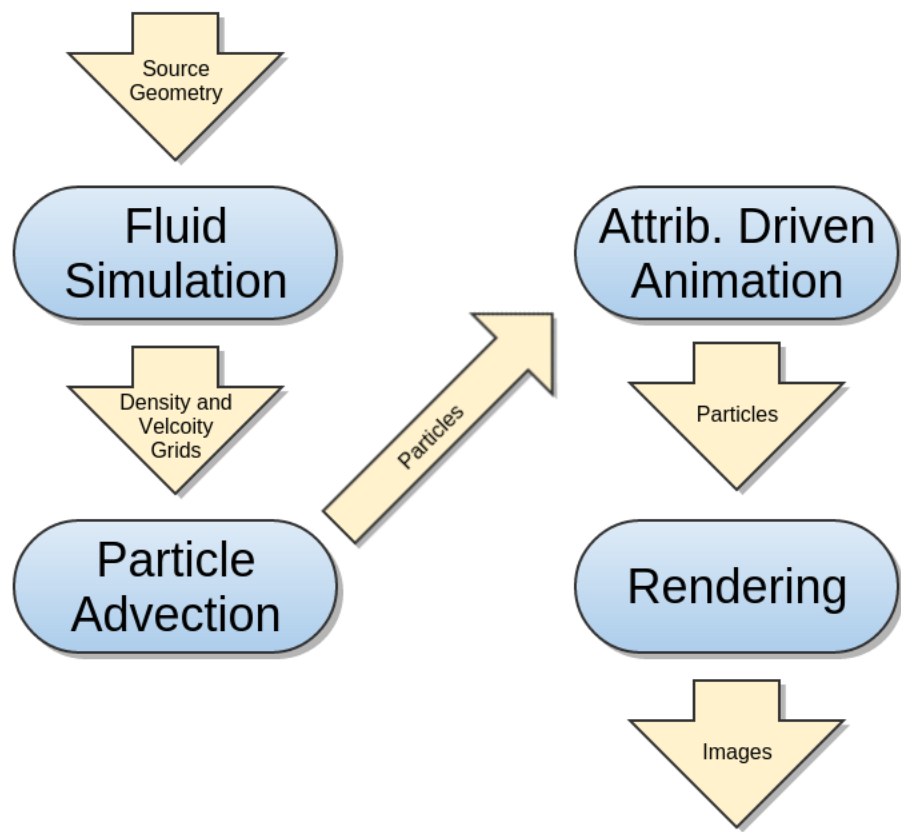


Figure 1.8: System Overview

## Chapter 2

# Fluid Simulation

Before any particles were simulated, a fluid was sourced and simulated. Since the intent of the initial burst of particles was meant to feel explosive, a fluid simulation was a natural starting point. The process of creating the fluid can be thought of in three parts: defining the source of the volume, simulating the movement of the fluid, and caching the output volumes for later use.

Fluid sourcing is the process of creating volumetric data from some form of input. The input is typically geometry, functions, or other volumes. During simulation, a fluid can be thought of as a collection of volumes. There is a wide range of volumes that can be sourced and used when simulating fluids, the main contributors for the desired effect were the density and velocity fields. Density is the visible field of a simulation, but it also has a dramatic effect on the motion during simulation. A uniform density results in a more uniform motion and uninteresting shape development. It is very common to use noise to break up the shape of the initial volume source in order to force a more interesting development of shapes. The velocity field has a direct effect on the motion of the volume because it defines the direction the fluid is moving and how fast it is moving in that direction.

The fluid density, temperature, and velocity were all sourced from a set of points generated from the mouse geometry interior, as points from the surface of the geometry gave a much less dense and less explosive look. Figure 2.1 shows the source geometry and figure 2.2 shows the points generated from the mesh. Sourcing from scattered points rather than the geometry itself allows for an additional layer of control since density and velocity attributes can be created and controlled per particle as well as each particle's area of influence when stamped onto the corresponding field. The



Figure 2.1: Source Geometry

velocity of the simulation needed to burst away from the center of the mouse, so the velocity of each particle was defined by the equations:

$$\begin{aligned}
 d &= P - C \\
 v &= d / \|d\| \\
 v &= vK
 \end{aligned}
 \tag{2.1}$$

where  $P$  is the particle's position,  $C$  is the center of the mouse, and  $K$  is a user-defined constant. The implementation within Houdini allowed for offsetting  $C$  and utilized VEX for attribute creation and manipulation. The velocity visualization can be seen in Figure 2.3

Once the source particles were stamped into the density, temperature, and velocity grids, to create a fluid source a layer of time varying turbulent and cell noise were multiplied in to the fields to break up the regularity of each. The density source and velocity streamers can be seen in figure 2.4.

Making sure that simulation time stays at a minimum is crucial because it limits the number



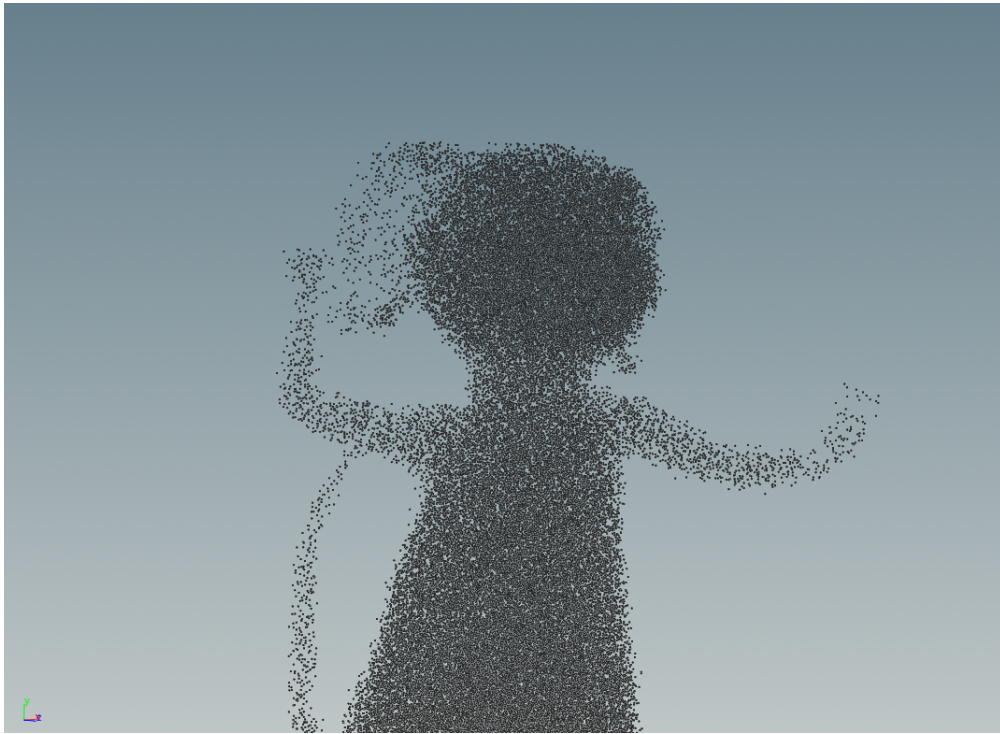


Figure 2.2: Source Points

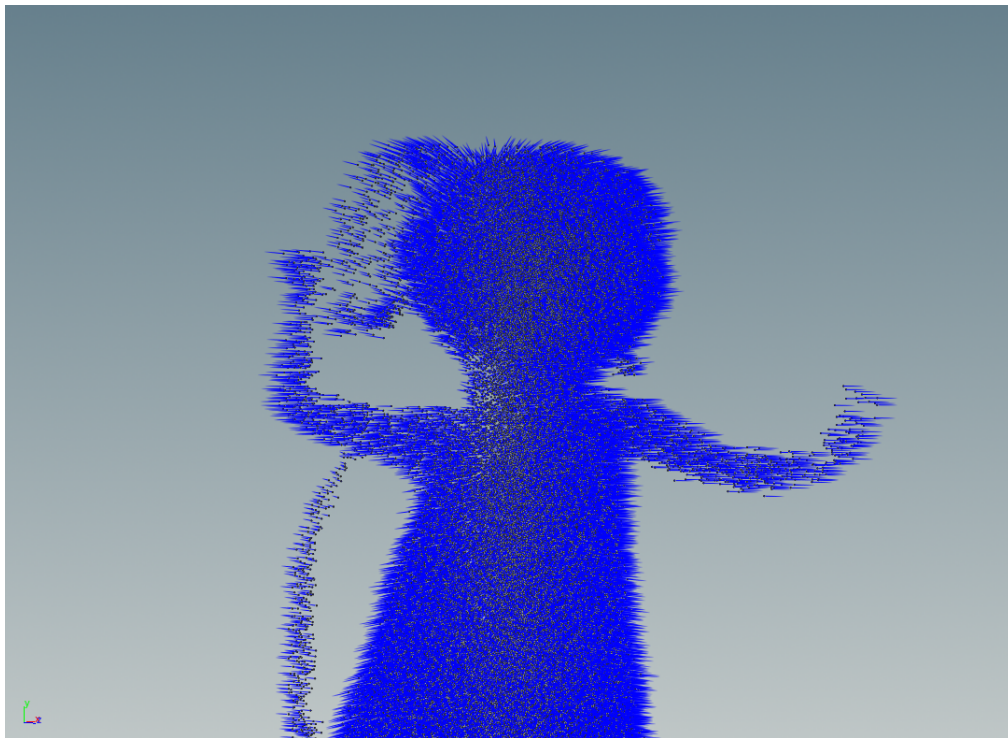


Figure 2.3: Source Streamers

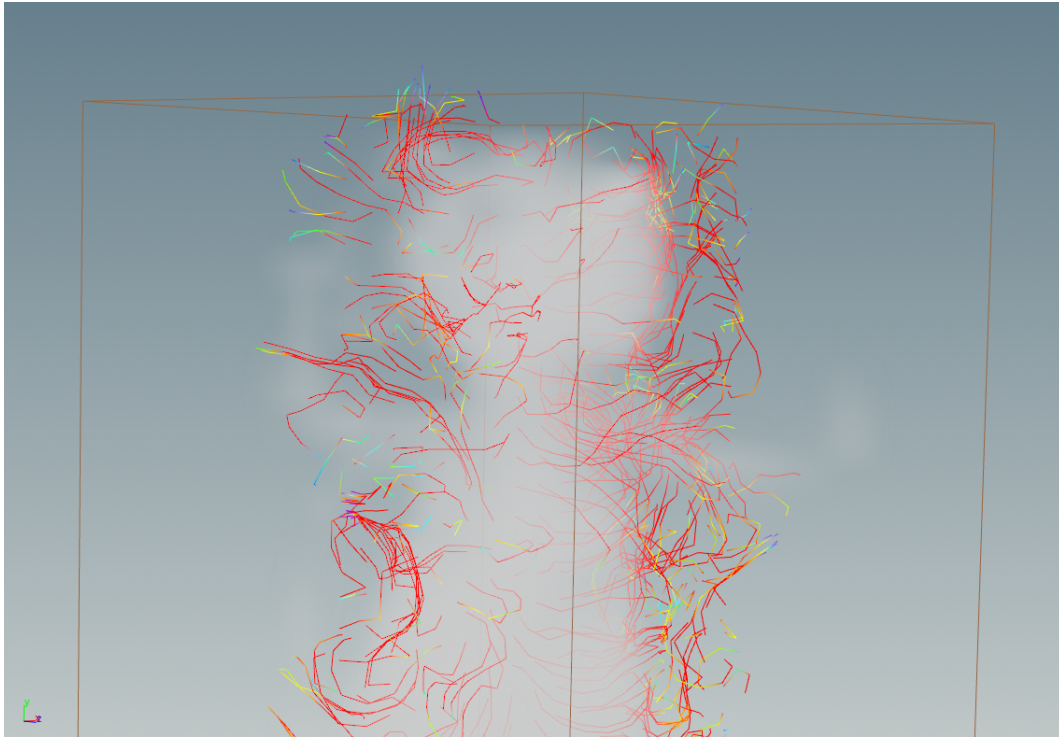


Figure 2.4: Density and Velocity Streamers

of iterations that are achievable, and more iterations means more chances to get it right. Collisions for the fluid simulation had potential to be very costly, since the fluid moved quickly initially. Collisions can be handled by designating scene geometry as collision objects or defining collision volumes, and depending on the resolution of the geometry or volume, the calculation can be slow. Speedy fluids can also require calculating the simulation multiple times per frame for accurate collisions, which compounds the problem. These factors indicated that collision detection would be the most time consuming portion of the simulation, which meant speeding up the collision calculation drastically improved simulation time.

Since the camera was very close to the microwave, as you can see in Figure 2.5, the only objects that needed collision detection were the microwave floor and front glass plate, which were both axis aligned. This allows for easy approximation with planes. The approximation planes used in the simulation can be seen in figure 2.6. Collision detection with planes is very fast, and since the planes could be reused for collision detection when simulating the particles, the manual approximation was well worth the time.

Simulations within Houdini are done within the Dynamics context. Operators within this



Figure 2.5: Collision Planes

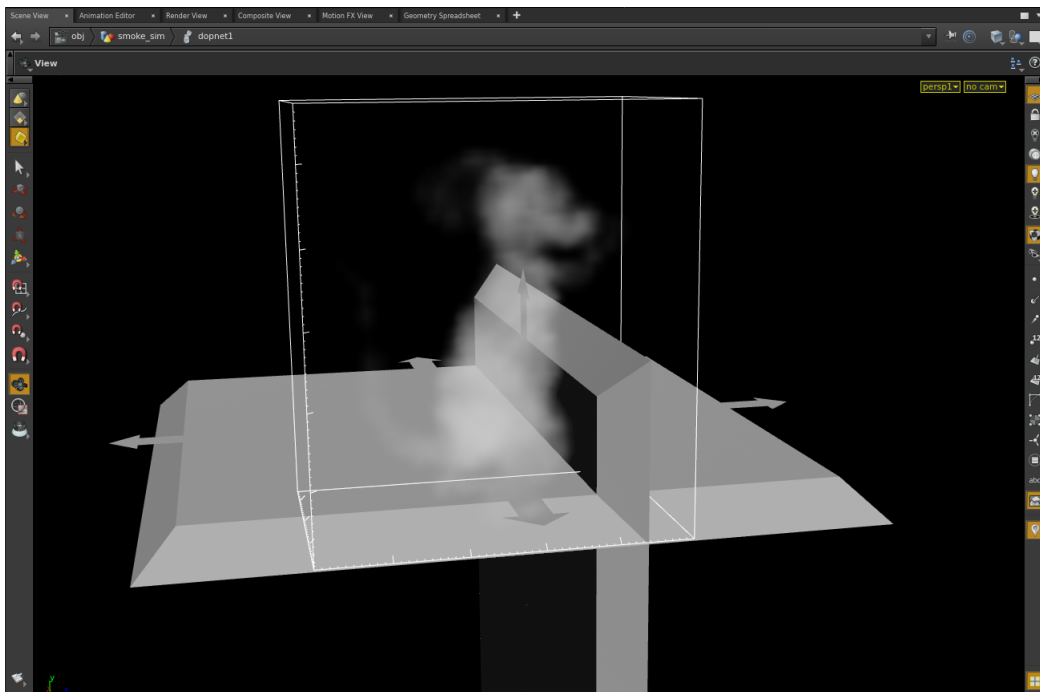


Figure 2.6: Collision Planes

context are connected together in the order that the user defines and different operators with varying orders can achieve a range of results. In addition to this, each operator has a set of controls that can be set and animated with expressions or keyframes. All of these options necessitate keeping simulation time low, as testing and reviewing many variations of operators, orders, and timing is necessary during the development of the effect. The final system incorporated keyframing of the sourced density and velocity fields, drag force, and three micro-solvers over a dynamic domain.

Keyframing the incoming fields was the fastest and most direct means of control. A large, fast burst was achieved by sourcing a large amount of velocity and density that quickly tapered off over the first five frames of simulation. Drag force was also used to quickly slow down the burst, which helped separate the motion of the burst and collapse. Additional DOPs nodes were used to introduce turbulence, vortex confinement, and disturbance into the system. Turbulence produces noise within the density of the volume and vortex confinement pushes the velocity field to introduce swirls. Disturbance was employed to help displace the mushroom clouds that typically develop in smoke simulation; it deforms the boundary of the density field to make the edges of the field less uniform.

Each of the added operators helped make the burst feel more explosive. Keyframing the source and the drag force worked together to develop the timing and bulk motion of the fluid simulation while the additional DOPs nodes helped add detail and interest to the simulation. The final DOP network is shown in Figure 2.7.

For particles advection, only the velocity fields are typically needed. However, during the development of the particle simulation, it was discovered that other fields could be used to add visually interesting results. The final cache included both the density and velocity fields.

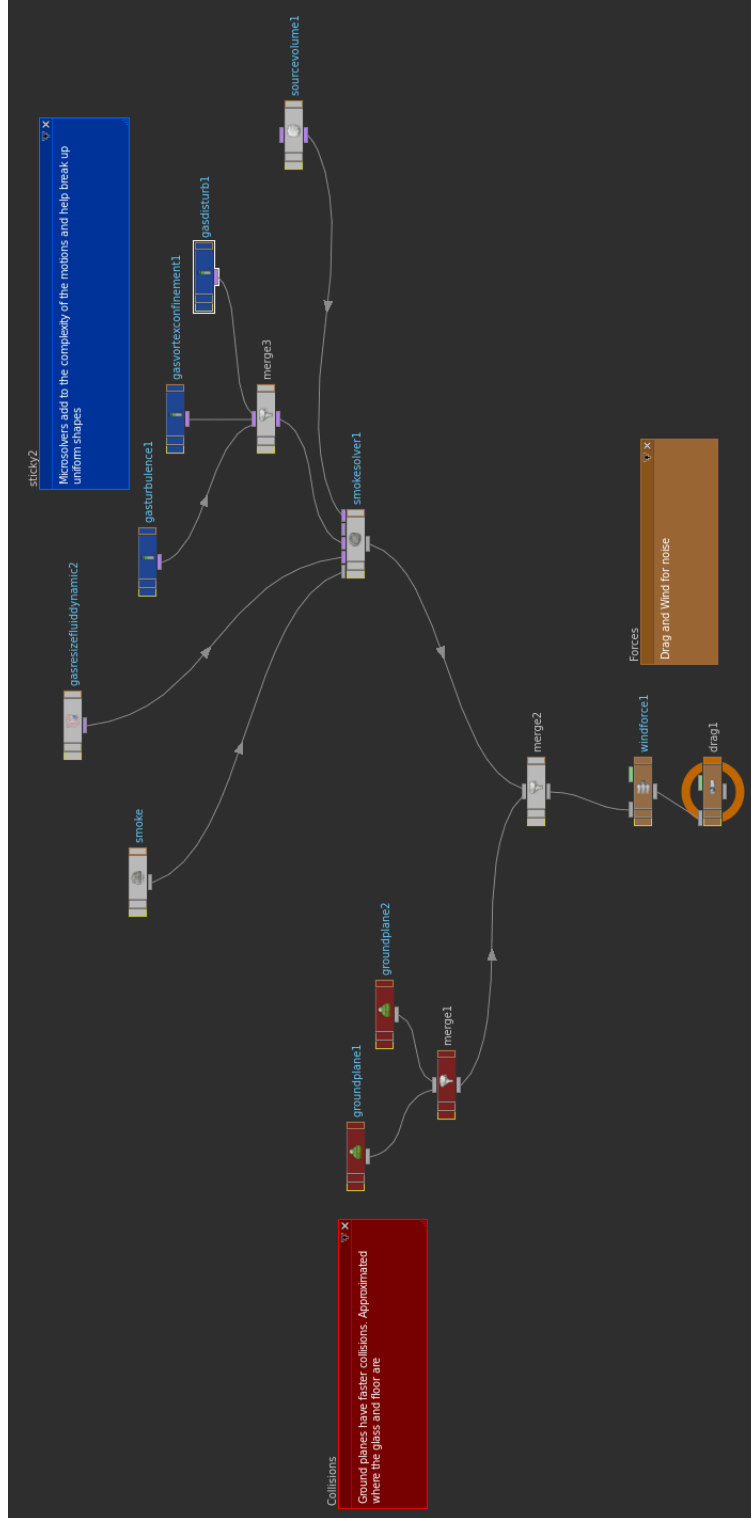


Figure 2.7: DOP Network

## Chapter 3

# Particle Advection

The next stage of producing the effect involved advecting a large number of particles by the previously simulated fluid. Using particles instead of the simulated fluid was necessary because the last step of animation for this effect involves procedural animation, which is very controllable for a set of particles. The goal of this stage was to keep the shape and timing of the fluid simulation and add to the structural detail.

The SOP (Surface Operator) network that includes the particle simulation can be seen in Figure 3.1. The particle source is a simpler version of the source used for the fluid simulation; particles were generated on the interior of the mouse geometry. A start color was projected on the particles, which is discussed in more detail in the rendering section. On the right side, the fluid's grids are separated for use within the particle simulation. After the simulation, unnecessary attributes were discarded, several octaves of Perlin noise is generated per particle per frame for use during procedural animation, and the system is cached to disk.

The DOP (Dynamics Operator) network used for the implementation of the simulation can be seen in Figure 3.2. This network is contained within the purple node named *particle\_advection* in Figure 3.1. First, the particles are sourced from the points generated from the interior of the mouse geometry, as they had been during the fluid simulation. The number of particles generated per frame was driven by the expression on the impulse count attribute of the source node:

```
npoints(opinputpath("../",0)) / ch("../substep") * ch("../key/sourceKey")
```

The sourceKey attribute was keyframed from zero to one, so it controlled the percentage of the

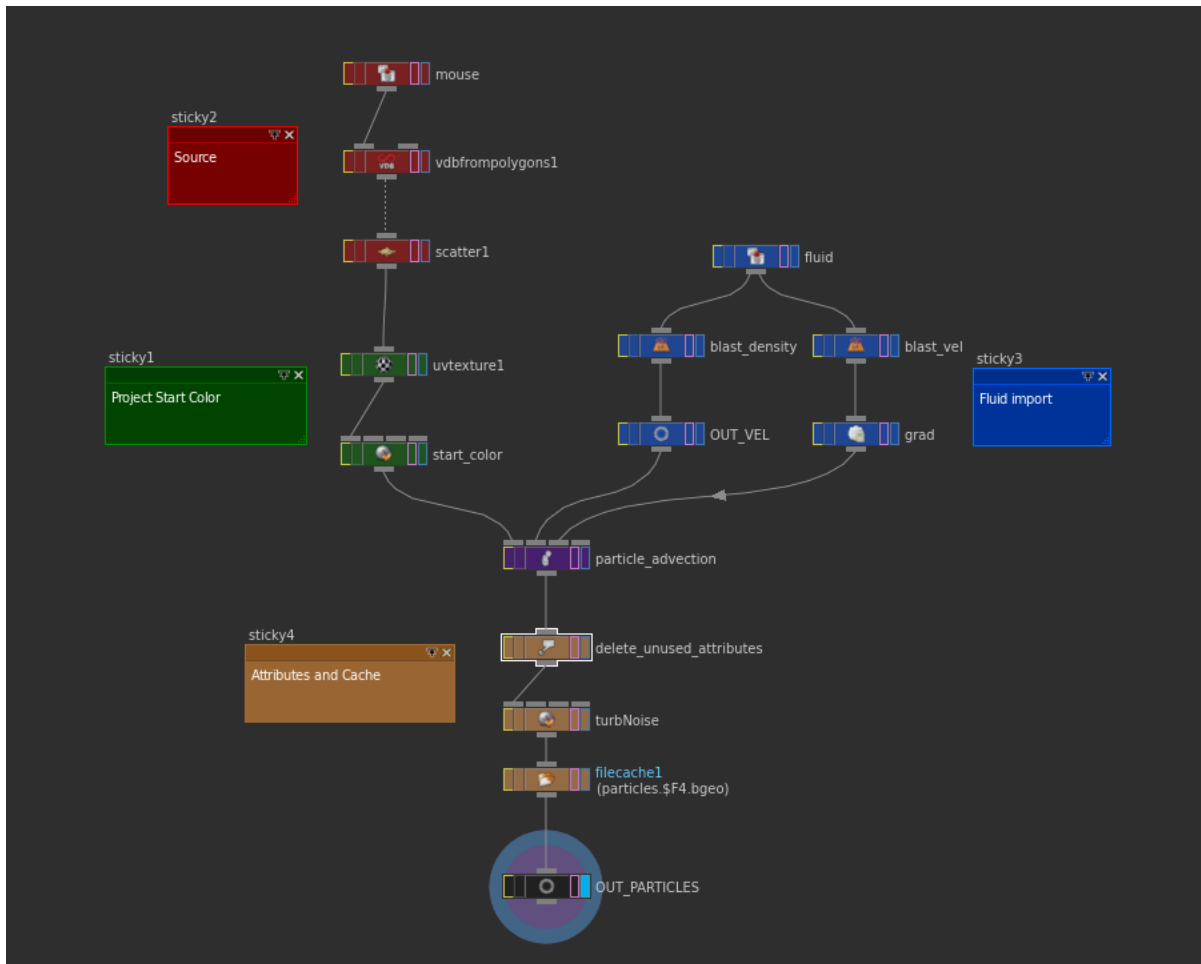


Figure 3.1: SOP network

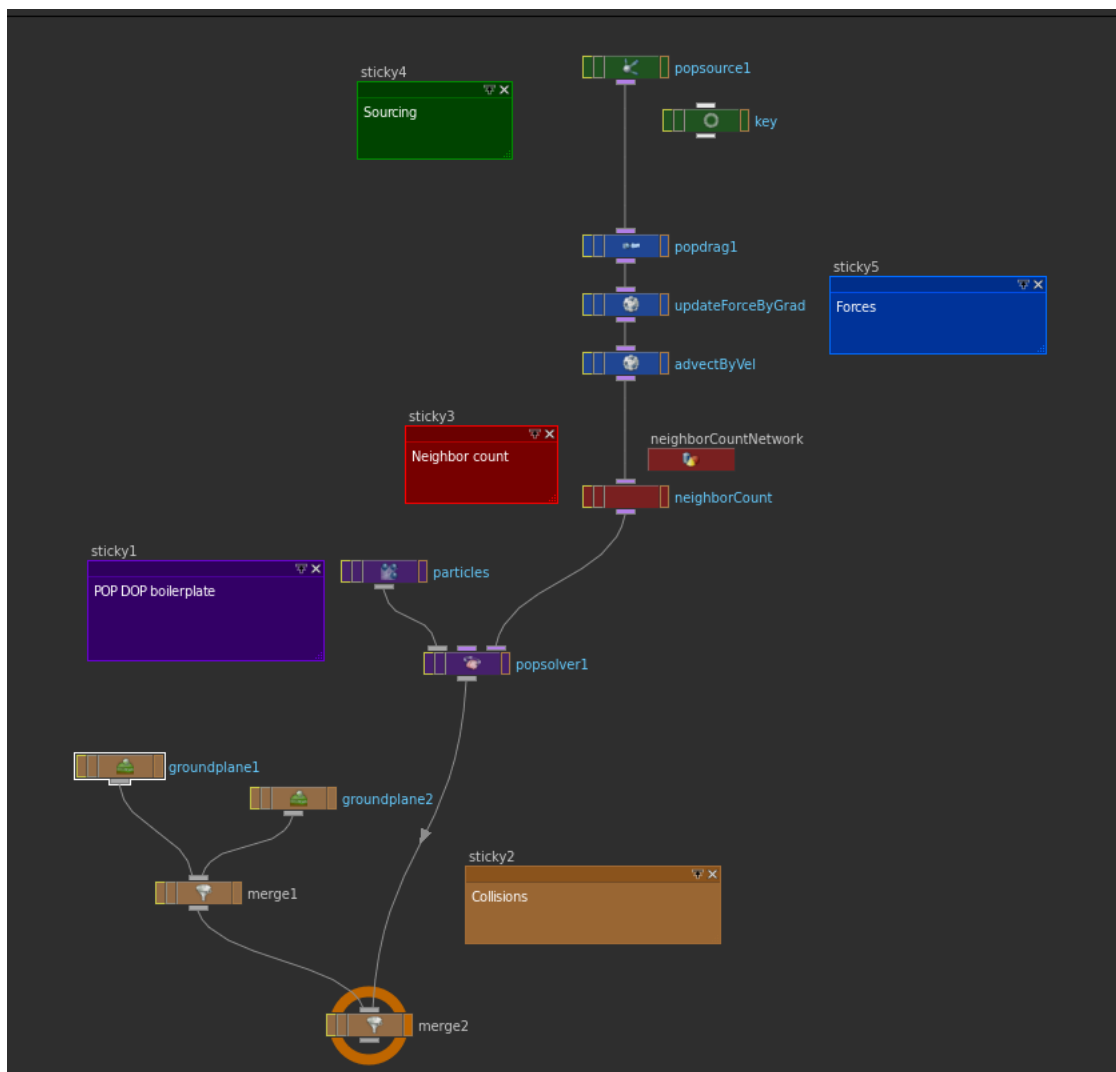


Figure 3.2: DOP Network



particles generated in the SOP source that would actually enter the simulation. This allowed adjustments to be made to the total number of particles entering the system without having to adjust every keyframe.

The next three nodes in the stream directly affect the particle system's motion. There is two-stage advection with a small amount of drag force. The first stage of advection updated the force on each particle by the gradient of the fluid's density field. This pushed the particles into denser areas of the fluid, making the structural detail in the simulation more pronounced. The second stage was updating the velocity of the particles by the velocity field of the fluid. Balancing these two stages was necessary in order to keep the system stable as well as keeping the timing of the fluid simulation intact. Too much gradient force caused the system to become unstable, but too little made the structural detail less apparent. Too much velocity update would overwrite the force update, but too little would drastically change the timing of the explosion.

During the rendering stage, an attribute was used to control the color of dense areas of particles independently. Calculating the number of neighbors can be a very expensive operation, so the attribute was calculated during simulation since the simulated particles would be cached to disk. The most efficient way to generate this attribute that was discovered was:

1. Create a field based on the bounding box of the points in the simulation
2. Stamp each particle into the voxel that contains it
3. At each particle, sample the corresponding voxel and store the value as an attribute

The first two steps occurring in the SOP network called *neighborCountNetwork* and the VEX node called *neighborCount* accomplished the sampling step.

The purple nodes are POP DOP (Particle Dynamics Operator) nodes used in every POP DOP simulation: a particle object to attach data and a solver node to solve the data. Finally, the same collision objects that were used in the fluid simulation are included. Figures 3.3 and 3.4 show the fluid simulation and corresponding particle system.

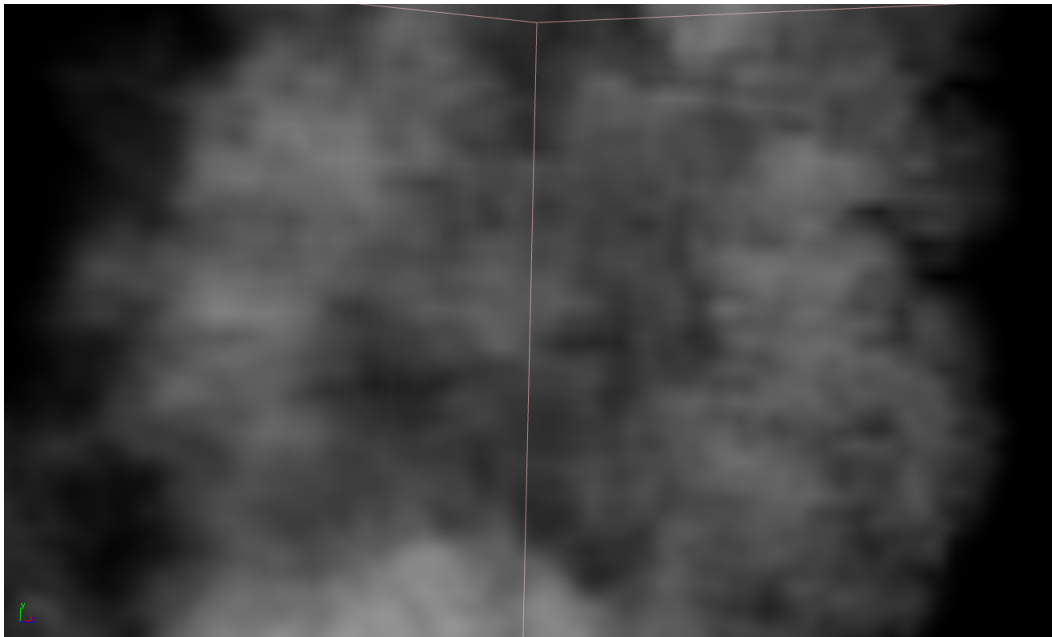


Figure 3.3: Fluid

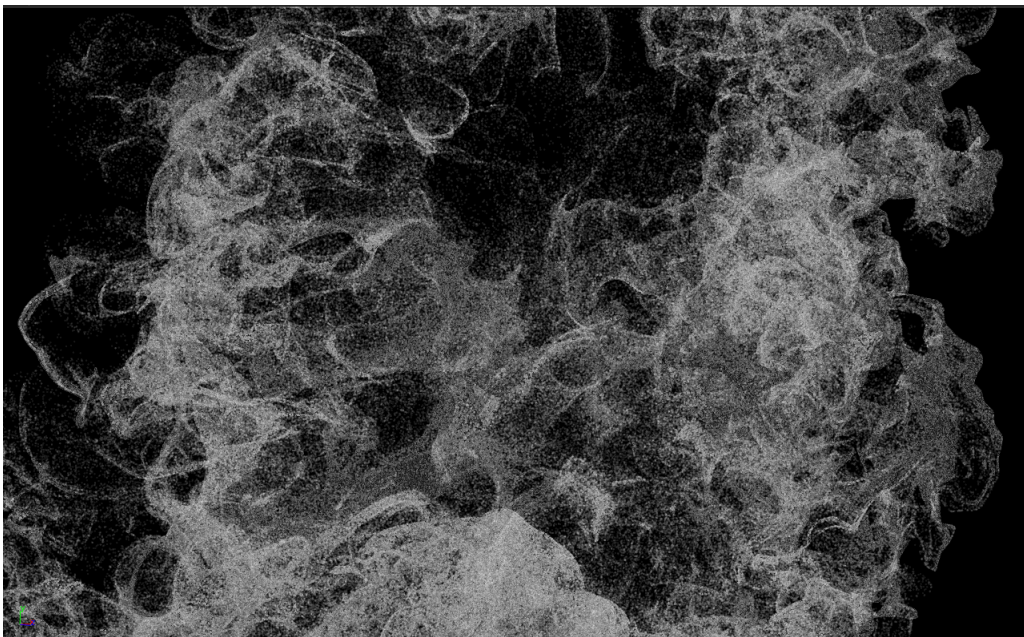


Figure 3.4: Particles

## Chapter 4

# Attribute-Based Animation

The implementation of the swirling vortex system combined per particle attributes and keyframe animation to drive motion. The system offered more control than simulation since timing was directly controlled by keyframes and individual strands of the vortex could be directly affected through certain attributes. Keyframe animation allowed for the precise timing of the bulk motion while per particle attributes gave the ability to vary the motion. Motion variation included timing and positional offsetting and helped make the motion much more appealing.

The first attribute used in the animation of the particles was dubbed the distance weight, based on the distance between the particle and a point in space. The variable,  $D$ , was defined by the distance between a user defined position,  $S$ , and the position of the particle itself,  $P$ . Turbulent noise,  $T$ , was driven by the particle's position to help break up the motion's uniformity.

$$D = \|P - S\| + T \tag{4.1}$$

This turbulent noise was generated after particle advection and stored on disk because computing octave noise for a large number of particles every frame proved to be time-intensive. Storing this attribute on the cached particles was more time-efficient.

The final piece of the attribute involved keyframe animation and a fit function. The distance variable,  $D$ , was fit to the clamped range  $[0,1]$  from a variable range that was keyframe-animated. The range was chosen because the attribute was to be used as a weight on a user-defined transformation. The animated range controls started as a very low value and were keyed to higher values,

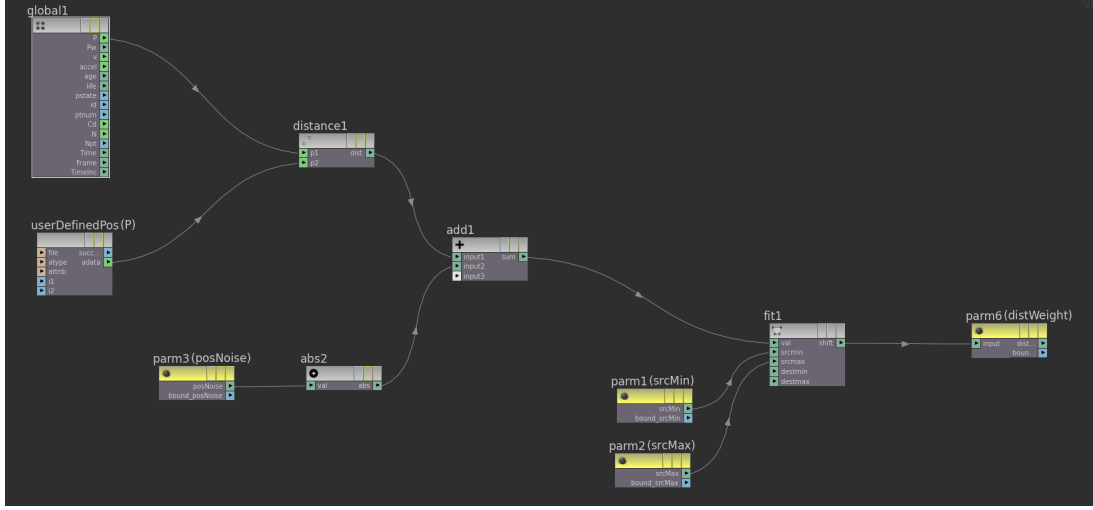


Figure 4.1: VOPs Distance Weight

which caused the attribute for each particle to start at 1 and end at 0. A value of 1 meant that a particle retained its advected position, any particle near zero was considered collapsed, and any other particle was being collapsed. This was accomplished by fitting  $D$ , the distance weight, from a user-specified range to the range  $[1,0]$  with the following equation:

$$D = (a - D)/(b - a) + 1 \quad (4.2)$$

where  $a$  is the user defined minimum and  $b$  is the user defined maximum. The resulting value is clamped on the  $[1,0]$  range, which can be formalized as:

$$\begin{cases} 1 & D < a \\ D = (a - D)/(b - a) + 1 & a < D < b \\ 0 & D > b \end{cases}$$

The final VOPs (Vex Operator) implementation can be seen in Figure 4.1

Before the discussion of how to apply the distance attribute to create the bulk motion of the particles, it is necessary to cover an additional set of attributes that allowed for control of the strands of the vortex that can be seen as the particles swirl to its center. This system gave control over the placement and tightness of the strands during the swirl collapse. Without this system, the

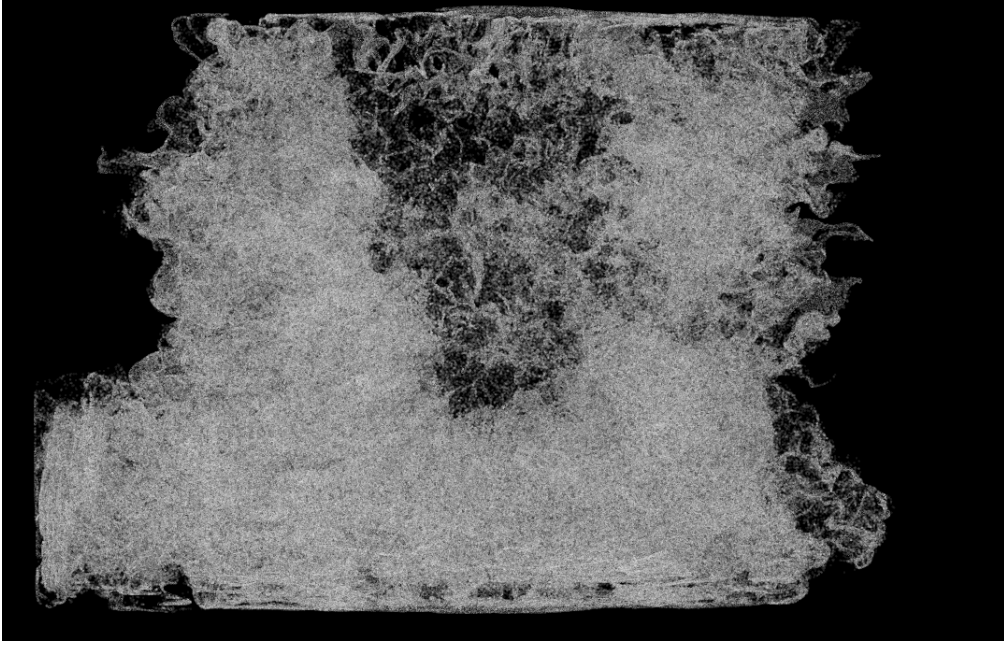


Figure 4.2: Particle System

position of the advected particles dictated the look of the swirl that followed. In other words, the look of the vortex was dictated by the shape of the simulation and not the artist.

The system created dense areas of particles that became visible as strands, giving the animation an atomic or quantum impression. It used a set of attributes to create an alternate set of positions for the particles that would not be visible, but would provide a clustered position that could be used when applying the transformation. First, a number of points were scattered on a designated frame of the fluid simulation called the cluster points. Each advected particle at the same frame looked up the closest cluster point and stored its I.D. and position as attributes. An attribute called cluster position was created by scaling all of the particles in towards their cluster points. Each strand could be identified by its cluster I.D. or cluster point. The particle system, cluster points, and cluster positions can be seen in Figures 4.2, 4.3, and 4.4.

The second stage of the attribute-based animation process was to apply the weight to a transformation. This transformation can be conceptualized as a blend shape and the previously described distance attribute as the weight between the advected particles and its transformed counterpart, hence the name for the attribute. However, a key difference between this method and a typical blend shape is that the transformation itself can be animated to add another layer of motion.

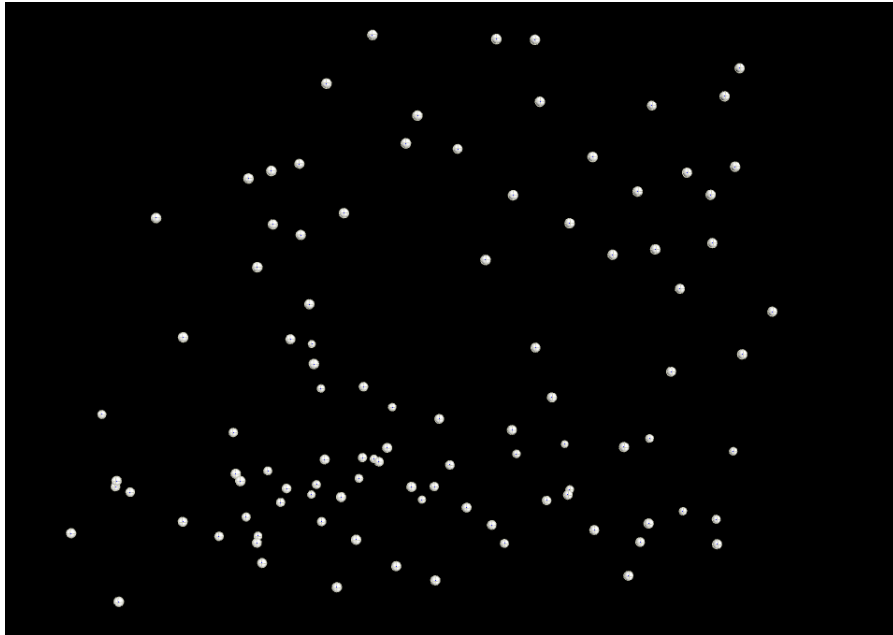


Figure 4.3: Cluster Points

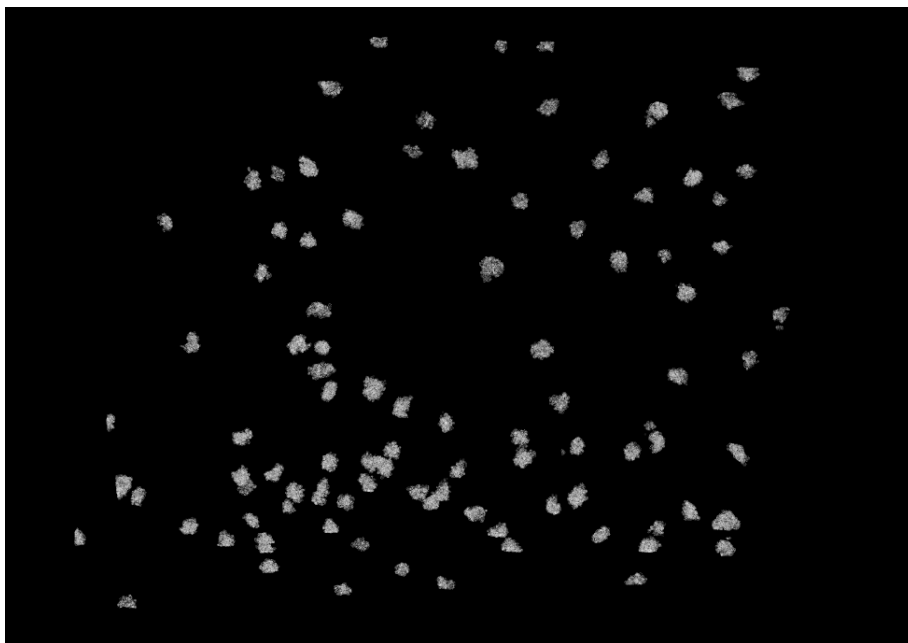


Figure 4.4: Cluster Position

The implementation required the assembly of a matrix from scale and rotation attributes. The scale scalar,  $S$ , and rotation vector,  $R$ , were defined as:

$$S = s * D^q \quad (4.3)$$

$$R.x = 0 \quad (4.4)$$

$$R.y = 0 \quad (4.5)$$

$$R.z = rD^w(rand(C) * 10 + .5) \quad (4.6)$$

where  $s$  and  $r$  were animated controls used to set the amount of scale and rotation,  $q$  and  $w$  gave the motion some easing from the explosion state to the swirl state, and  $C$  was the cluster point. Multiplying by a random amount based on the cluster point created a variance rotation per strand. The corresponding implementation in VOPs can be seen in Figure 4.5 and 4.6

The matrix was applied to a position blended between its actual position and cluster position. The blended position was:

$$U = PD^e + L(1 - D^e) \quad (4.7)$$

where  $P$  is the cached particle position,  $D$  is the distance weight,  $L$  is the cluster position, and  $e$  is an exponent to control the easing from the initial particle position into the strand.  $e$  also controls the shape of the funnel that can be seen at the areas between the initial particle position and the strands. For the VOPs implementation, see Figure 4.7. Finally, the matrix transformation was applied at the same user-defined position in space used to calculate the distance weight attribute. The VOPs implementation of the matrix construction and application can be seen in Figure 4.8. For the entire VOPs network, see Figure 4.9

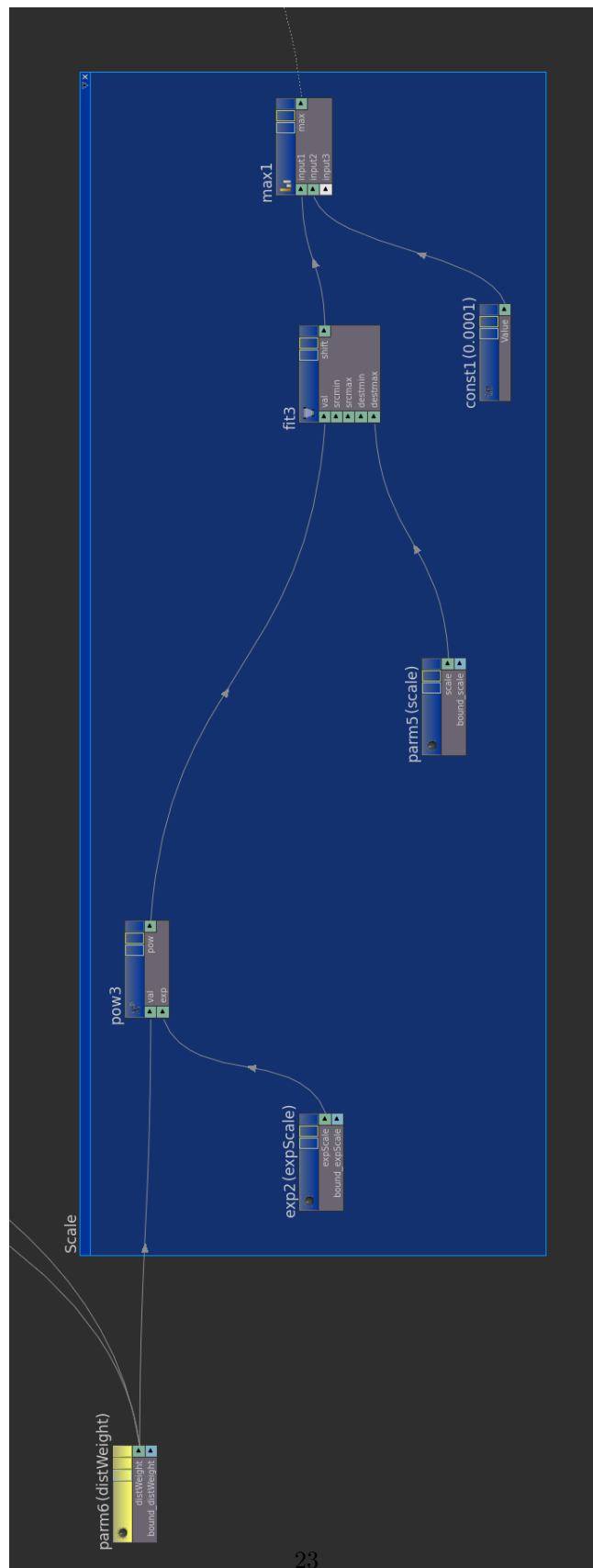


Figure 4.5: VOPs Scale



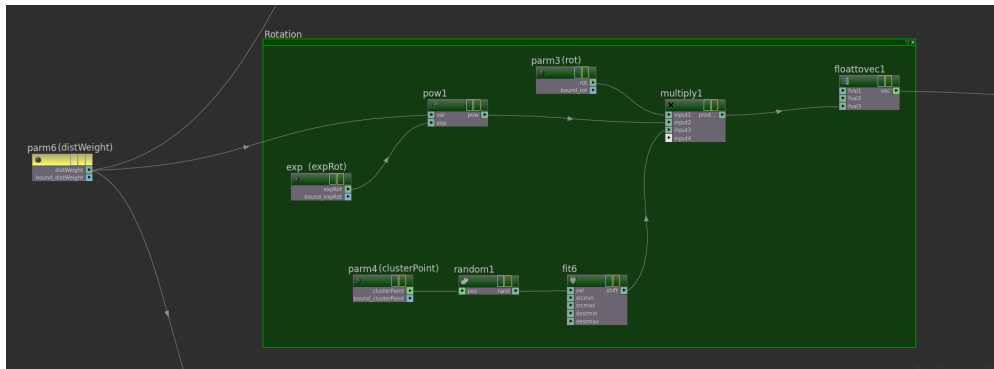


Figure 4.6: VOPs Rotation

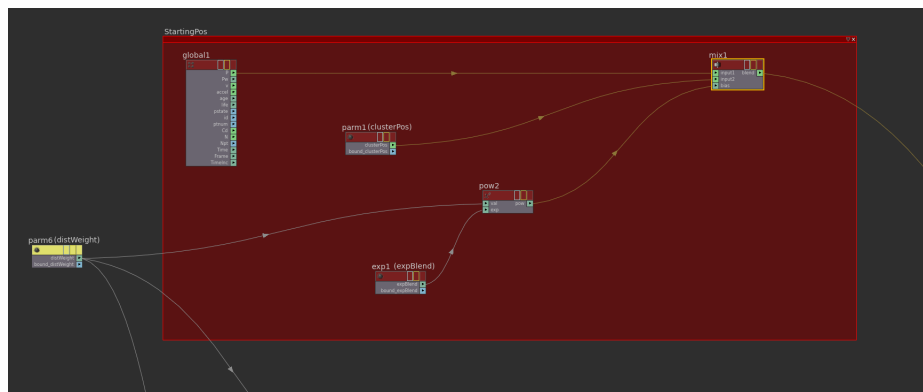


Figure 4.7: Position Blend

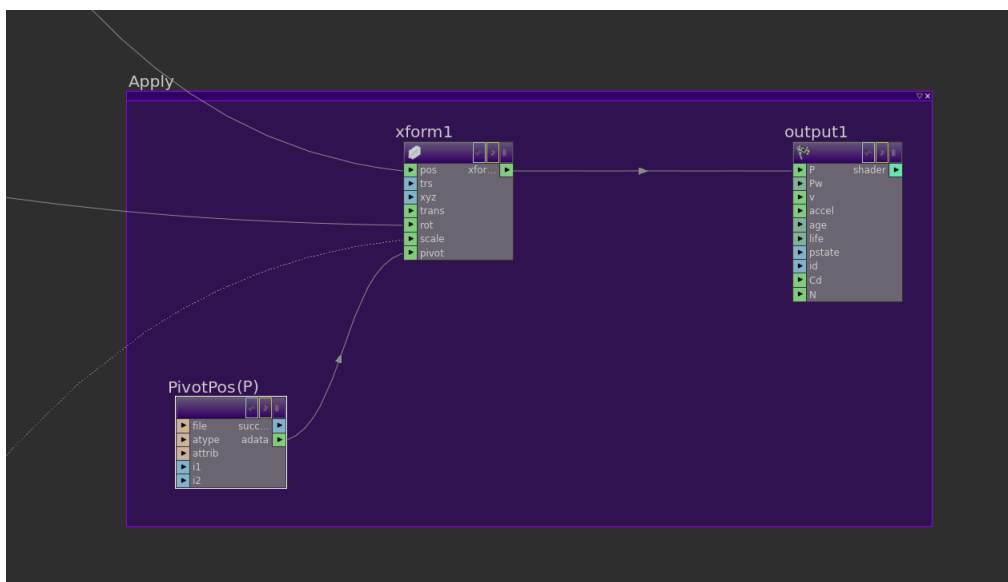


Figure 4.8: VOPs Apply

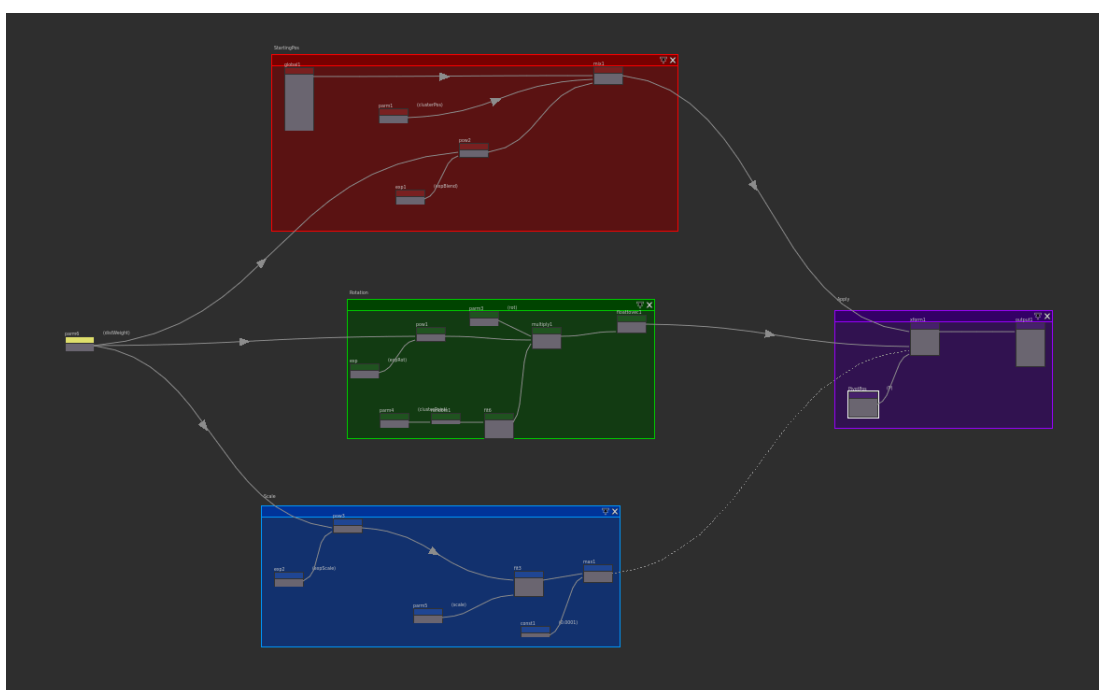


Figure 4.9: Transformation VOPs Network

## Chapter 5

# Rendering

A number of rendering methods were tested, including shaded point rendering and volume rendering. Additive point rendering was the easiest to manipulate and gave the least amount of noise. For additive point rendering, a color and opacity, or alpha, is set for each particle. At render time, each particle that is within a given pixel contributes its pre-multiplied color to that pixel, so denser areas of particles are more visible than less dense areas and outliers were almost invisible. Shaded point rendering had more visible outliers, which made the image appear more noisy. Rasterizing the points into a high-resolution volume was time-consuming and there was a noticeable loss of detail.

It was important to ease the transition from the mouse into the teleportation effect. To accomplish this, the rendered frame of the mouse immediately before the simulation begins was exported and used to project color onto the particles at birth. The color was then blended to their final color over about one third of a second from the particle's birth. This dramatically eased the transition into the effect. You can see the start frame of the teleportation in Figure 5.1.

The first layer of color for all of the particles was driven by two layers of Perlin noise based on the world space position of the particles. The user was allowed to select a base color, a low-frequency color, and a high-frequency color. These layers of color can be seen in Figures 5.2, 5.3, and 5.4. To reduce the amount of complexity for the operation, the frequency of noise was locked down once the simulations had gone through their initial iterations since the scale was blocked in at that point. The color was decided early on in the development process, so there was no need for extensive of experimentation.

The next layer of color was to make the denser areas appear hotter. During the advection



Figure 5.1: Projected Start Color

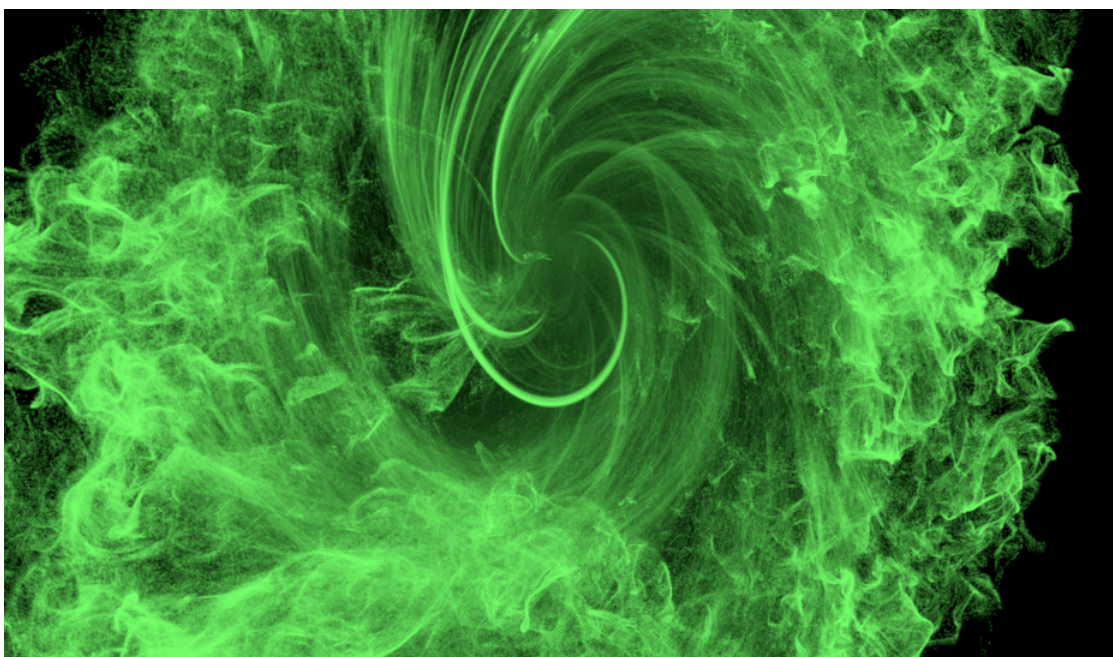


Figure 5.2: Base Color

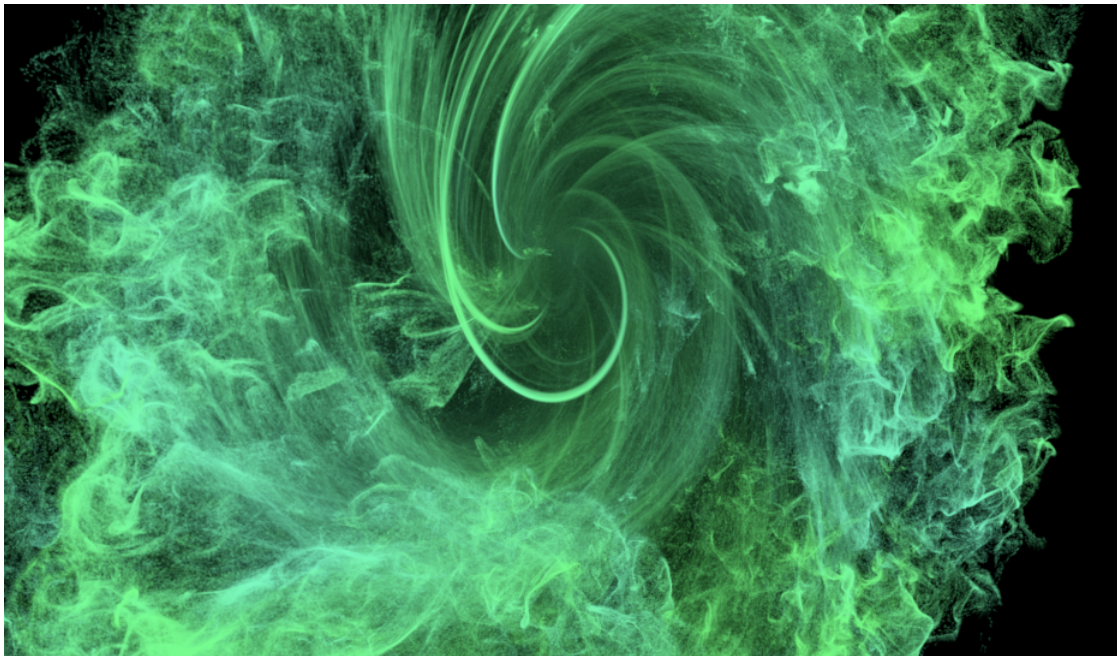


Figure 5.3: Low -Frequency Color

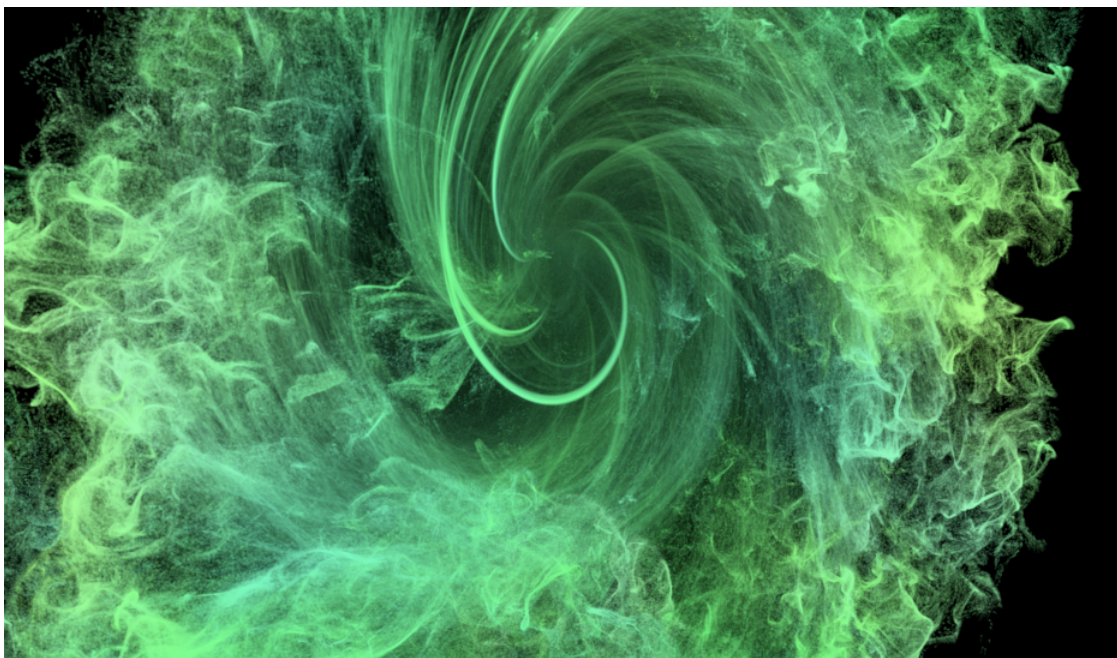


Figure 5.4: High-Frequency Color



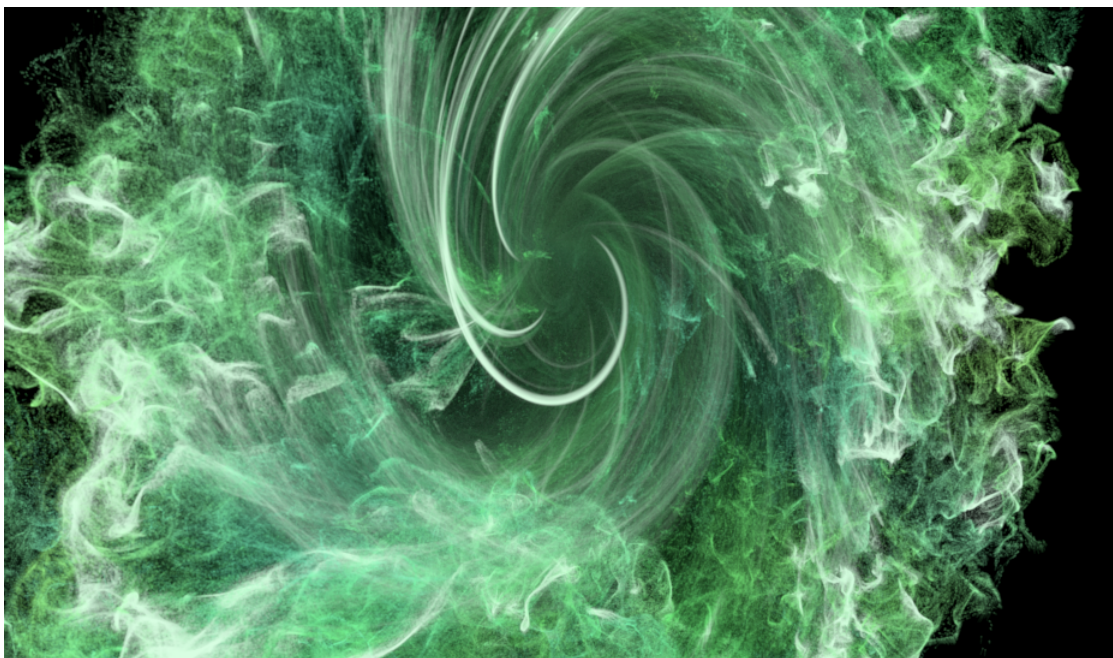


Figure 5.5: Density Color

stage, an attribute called neighbors was created to approximate the number of neighbors for each particle. This attribute was used to blend an additional color on top of the color mixing that was accomplished with Perlin noise. The attribute was fit to the zero to one range from a user-specified range. This attribute was used to linearly interpolate between the particle's initial color and a color close to white. This additional layer of color can be seen in Figure 5.5. A major limitation of this technique is that the particle count was tied to the color mix, so the range needed adjustment when it came time to increase the number of particles. Normalizing the attribute during the advection stage when the attribute was created would have helped solve this problem.

The final layers of color were added during the compositing stage, and the node graph used for the teleportation effect can be viewed in Figure 5.6. Any color corrections that occurred when dealing with semi-transparent pixels must be un-premultiplied before color correction and then premultiplied after. This ensures that the calculation is done correctly. Small color adjustments were done in this way to make sure that the effect remained visible when put in context with the other elements of the scene. The other major adjustment done during the compositing stage involved adding two layers of glow. The first was a glow for only the dense areas that rendered white due to the color addition due to the neighbor attribute and the second was a general glow with a much

lower intensity. The white glow helped the bands of dense particles become more pronounced and become visibly hotter.

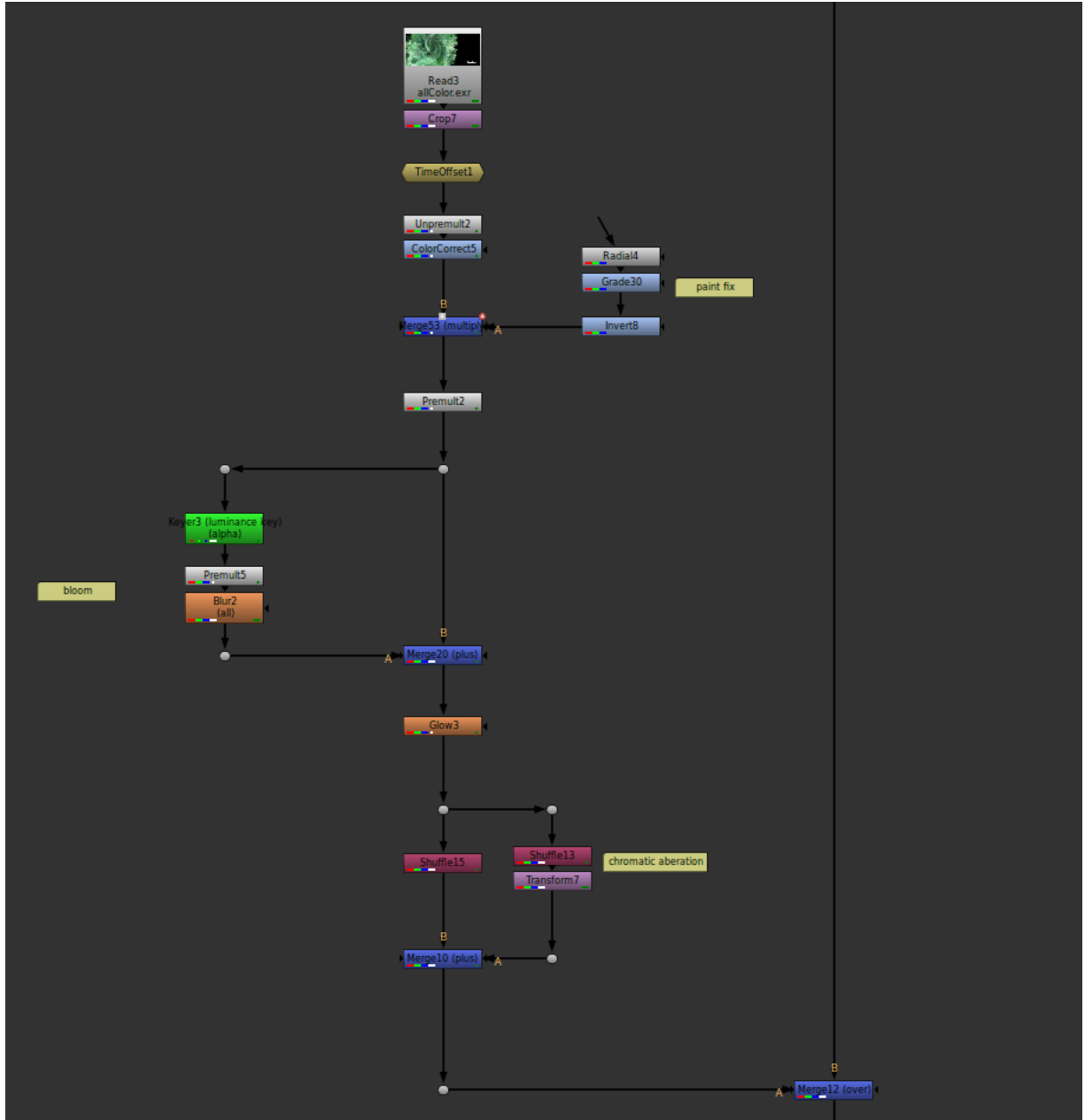


Figure 5.6: Nuke nodes



## Chapter 6

# Conclusions and Discussion

The technique used to design, implement, complete the teleportation effect exposed enough control to accomplish the goals set out in an efficient way. However, after production was completed, a number of improvements and additions were noted that could improve efficiency or add artistic control.

During the design and implementation stage, there should be an effort to disallow the particle count from having a drastic effect on the motion and color of the effect. This was eventually accomplished for the motion, but only after experimentation and a few iterations of each step of the process. In the end, the hot bands of color driven by the neighbor count attribute was still directly tied to the particle count, which caused some slow down between re-simulating and rendering on later iterations. If a little more time was spent early in the implementation process to separate the particle count and motion, a lot of time could have been saved late in the production.

A problem occurred late in production when FX switched from using layout assets to animation assets. The mouse animation had some timing and positional tweaks, and since the source of the particles and fluid simulation were coming directly from the imported animation, updating the animation changed the simulation. When the switch was made, the timing of the explosion stayed the same, but the shape completely changed. Since the look of the explosion had already been reviewed and approved, this was not acceptable. The end solution was to roughly match the layout model using some offsets in time and space. The initial explosion happens so quickly that it was not noticeable, but it might not be wise to source fluid simulations directly from animated models in the future.

For a short time there was experimentation with procedurally animating around multiple pivot points rather than rotating and scaling about one in the center and possibly animating those pivot points. While a naive implementation proved to be simple to setup, it was clear to see that the amount of control needed to complete the effect would require too much time to implement and harness. The naive approach had a set of points that acted as pivots for the simulated particles. Each simulated particle had the closest point in the set stored as an attribute which was used instead of the global pivot point when the weighted transformation was applied. However, there was no artistic control to adjust which pivot point a particular set of particles would follow and when a particle switched pivot points the resulting motion was jarring.

An early note was that the lines of the strands appeared too clean. Adding some noise to the position of the particles as they traveled along the strands was suggested and attempted for a brief amount of time in the late weeks of production. There was not much time for experimentation or many iterations for the idea, so in the end it was not used in the final version of the effect.

Each particle had a cluster id that represented its associated strand. This could have been used to leverage fine control over an individual strand. The position, timing, or shape could be adjusted, or an entire strand could be removed. In addition, the cluster positions were all editable points in space, so the starting points of a single strand could be adjusted without fear of changing the others.

Even though there are some improvements that could be made to the solution used to create and control the teleportation effect, the process was efficient and controllable enough to be used in a production environment and deliver on the goals set for the look and feel of the effect. Figures 6.1 through 6.7 are images from the final film.



Figure 6.1: Frame 92



Figure 6.2: Frame 96



Figure 6.3: Frame 100

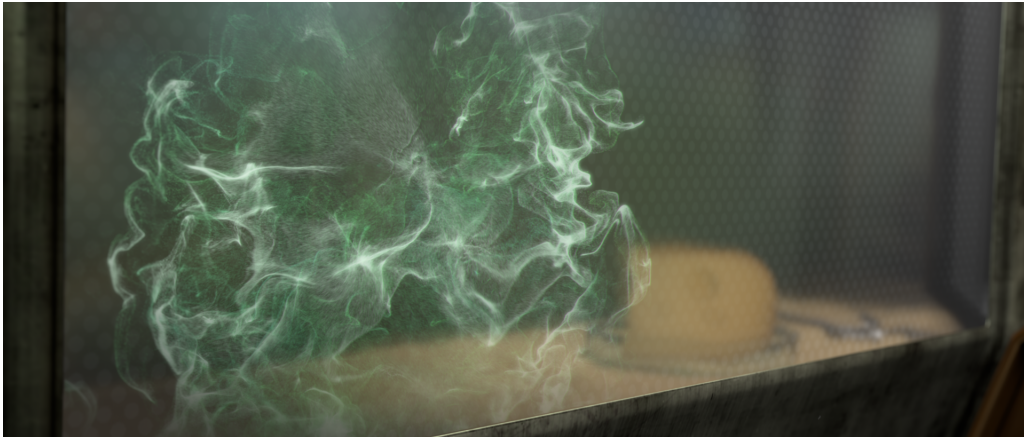


Figure 6.4: Frame 110

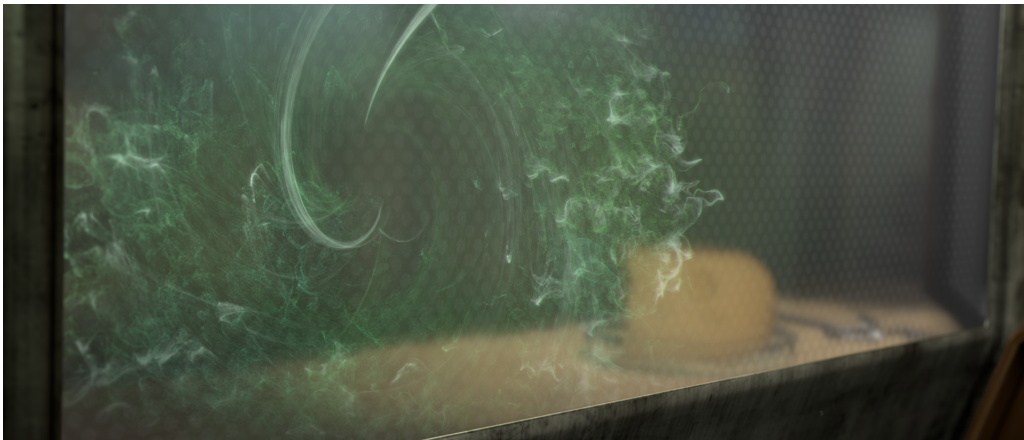


Figure 6.5: Frame 132

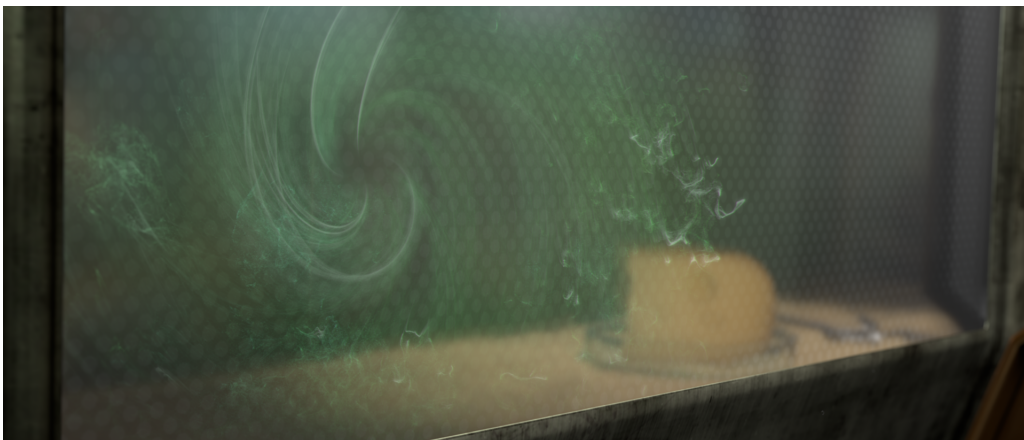


Figure 6.6: Frame 142

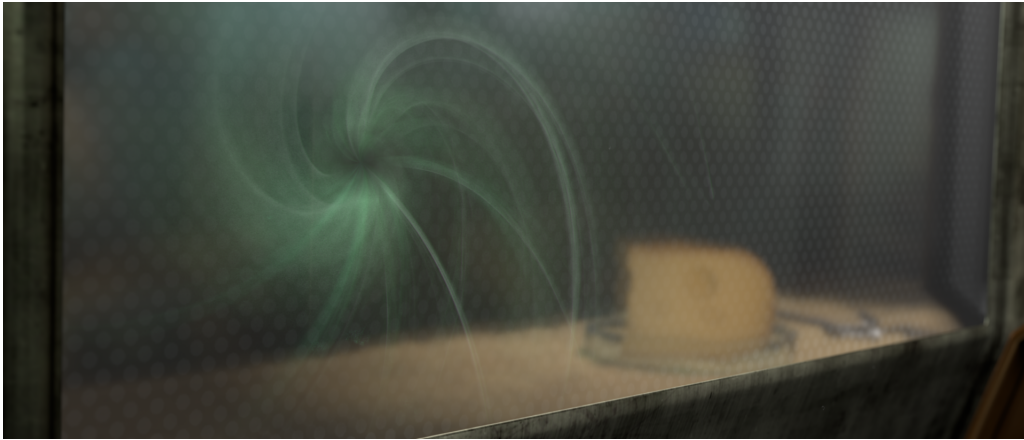


Figure 6.7: Frame 145

# References

- [1] Bill La Barge, Jerry Tessendorf, and Vijoy Gaddipati. "Tetrad Volume and Particle Rendering in X2". Siggraph sketch, 2003.
- [2] Yates, David (Director). Harry Potter and the Deathly Hallows: Part 1. Warner Bros., 2010.