Master's Project

Shadow Consuming Effect HDA

Tang Yeh

MSc Computer Animation and Visual Effects
National Centre for Computer Animation

Bournemouth University

14th September 2022

Abstract

This paper documents the method and implementation of a shadow like effect that has branches growing on a geometry and eventually consume the entire model.

L-system has been a popular method to generate tree-like branches for the past few decades, along with diffusion-limited aggregation. The final result of this project implements the latter method in Houdini. Two custom HDAs are created in order to achieve the final result, one creates the shadow geometry and the other simulates the shadow consumption of the model and decompose into particle dusts. VEX language and the ray SOP node in Houdini are also important parts in the effect creation process. The developed asset provides users to simulate shadow effect with different branch patterns on the model that is input into the digital asset, as well as the simulation of shadow consuming/spreading the whole model and decompose into particle dusts.

Keywords: Diffusion-limited aggregation, procedural, VEX, Houdini, ray SOP, digital assets.

Acknowledgements

I would like to take this opportunity to thank my course leader, Jon Macey, for his support and guidance not only on this project but also throughout this year during the program.

I would also like to thank my supervisor, Jian Chang, and Ian Stephenson, for their advices and suggestions on this assignment. As well as all my lecturers and the staff at the National Centre for Computer Animation (NCCA) for their help and support over the year.

Finally, I am grateful to all my friends and classmates from the NCCA 21-22 masters for their encouragement and support.

Content

1 Introduction

Different types of visual effects spreading across models have been presented in many projects in this industry, including both 2D and 3D projects. The patterns of the effects and the method to grow and attach them on geometries varies. For generating shapes with branches, tree-like algorithms are often chosen to be the ones to implement. There have been many studies in the computer graphic field for simulating tree-like patterns. For instance, L-system is a very popular and widely used algorithm to create tree-like designs, along with diffusion-limited aggregation. The methods mentioned are able to generate realistic patterns and fractals similar to frosts and tree branches. All methods could be done procedurally and a procedural approach to create complex and repeat patterns is less time consuming.

By understanding the L-system and diffusion-limited aggregation, this project focuses on the latter method to create the shadow patterns that are grown on the models. The main soft-ware that is used in this project to achieve the effects is Houdini, and Houdini Digital Assets (HDA) and particle system are the approach to create the effects. Other techniques that are also used in this project is VEX wrangles, which is an important part of this assignment, and Solvers. This paper will first review previous works on the algorithms mentioned and the technical backgrounds used in the project. Follow by the implementation of those methods and techniques, and some issues and problems in the developing process. Finally, the overall result and conclusion will be provided, as well as ideas for future development.

2 Previous Work

Spreading effects could be seen in many films and shows, such as *Rise of the Guardians* from DreamWorks Animation and and Japanese animation show *Naruto*. To generate patterns with branches, the common algorithm used is the L-system. There are also other popular methods could this type of effects, that is the diffusion-limited aggregation method. The two different ways will be reviewed in the following sections.

2.1 L-System

L-system, or Lindenmayer-system, is perhaps the most famous and common technique when it comes to creating tree-like growth patterns. The algorithm was introduced by a Biologist named Aristid Lindenmayer (1968), and it is created as a biological development to model the growth processes of plants. It has been further developed to generate a set of repetitive rules and create fractal-like plant models in *The Algorithmic Beauty of Plants* (Prusinkiewicz and Hanan 1990).

The algorithm is a type of formal language and a parallel rewriting system, including alphabet symbols to make strings, rules to expand every symbol to larger string of symbols, initial construction called “axiom” and converting strings into geometric structures. The initial axiom will form a string and then construct further with repeated applications of the replacement rules. The production rules of the system are very versatile and it has gone beyond the generation of plants. It can be used to create intricate fractal structures due to the recursive nature of the algorithm.

Although the L-system is widely used, this project has chosen to focus on the other methods to create a system that is simpler. Another reason to concentrate on other methods is that Houdini has a built in L-system node, therefore it would be more challenging to implement other methods.

2.2 Diffusion-Limited Aggregation

Another method of creating tree-like structures is named Diffusion-limited aggregation, also known as the DLA method. The theory was first proposed in 1981 by Witten and Sander (1981), and it is a kinetic phenomenon that can be applied to any systems as long as diffusion primary transport in the system. This is a process of particles spawned in random area going through random walk because of Brownian motion and make contact with one or more initial seeds that are called aggregates. When the connection is established in the particle will stop wandering around randomly and remain in the same position, it has been added to the aggregate and the aggregate grows.

Bourke (2014) suggested numerous ways to simulate the DLA process by computer, the most common way is to start a single black pixel in the centre as the aggregate in a plain white image. Follow by spawning new points at the borders and walk them randomly through the image until they are attached to the black pixel and grow the aggregate, this is the point attractor shown as Figure 2.1.

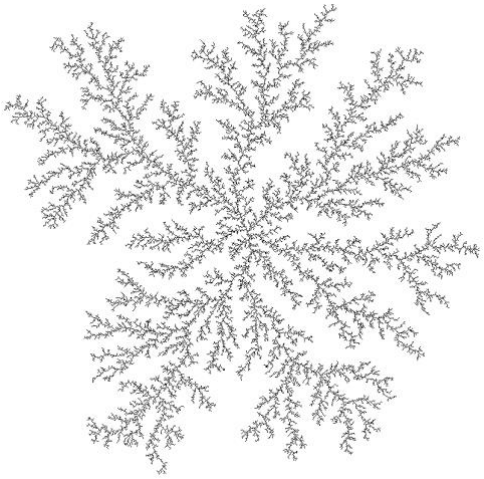

Figure 2.1: Point attractor of Diffusion-limited aggregation

Another attractor could be a line, this approach is to set a number of black pixels on the bottom edge of the image and spawn points on the top edge and make them wander down. The result of this attractor is illustrated in Figure 2.2.

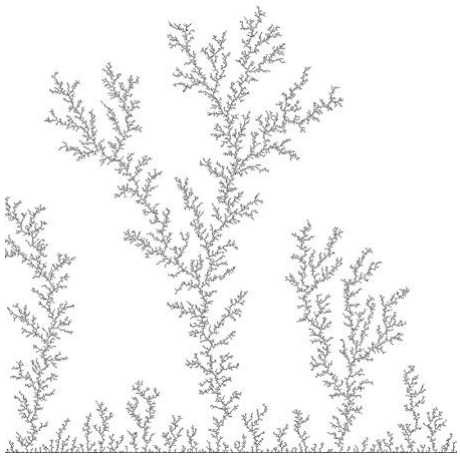

Figure 2.2: Line attractor of Diffusion-limited aggregation

Bourke (2014) also encouraged to manipulate the stickiness attribute, which controls the probability of the particle sticking on to the aggregate. Figure 2.3 and 2.4 shows the outcome of different stickiness value.

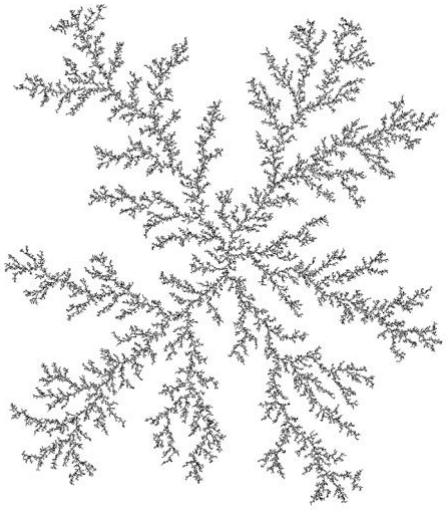

Figure 2.3: Stickiness = 0.2

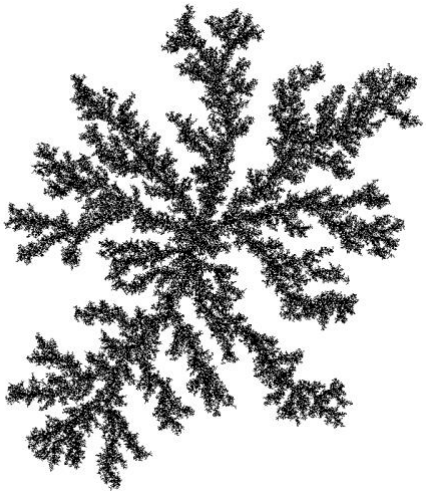

Figure 2.4: Stickiness = 0.01

This process is simple, but it could still create complex and tree-like structures. As a result, this method is chosen to be implemented in this project.

3 Technical Background

The effect tool described in this paper is accomplished with the 3D soft-ware Houdini. It is a powerful soft-ware that can create models and effects procedurally and make them easier to manipulate than other 3D soft-wares. The following sections will be introducing the nodes and tools in this soft-ware that are used to achieve the final result.

3.1 VEX

The major parts to finish the effect tool are done in attribute *wrangle* nodes, which are written in VEX, a Houdini native scripting language. VEX is quite versatile and can be used throughout Houdini, including modelling, rendering, and particle systems. It is a expression language based on C, C++ and RSL, and the language is relatively easy to understand with any previous experience of computer languages (SideFX 2022).

Understanding VEX is extremely useful when using Houdini, especially relate to custom nodes, manipulating node attributes and using *wrangle* nodes. Particularly the *wrangle* node, it is the core of this effect tool. The node is used to initiate the growth of the shadow pattern and the control of consuming simulation. This will be further explained in section 4 for more detail.

3.2 Ray SOP

The variety of nodes in Houdini is extremely versatile, they can accomplish many types of effects and amazing results, and the *ray* SOP is one of them. This Houdini built in node is for the purpose of projecting one surface onto another (SideFX 2022). It provides two types of projection methods, one is *Project Rays* and another is *Minimal Distance*. There are other attributes that could be

adjusted in order to manipulate the outcomes. This is a powerful node that could be used to satisfy different sorts of needs.

3.3 Particle System

The particle system in Houdini is part of the DOPs, which states “dynamics nodes set up the conditions and rules for dynamics simulations” (SideFX 2022). Because the particle system is part of the DOPs, this makes it easier to combine different dynamic simulations in a single DOP network. The *popnet* node is one of the DOP networks and it has established some basic particle set-up inside the network, and this makes the node perfectly for creating particles. Forces and attributes could be added to the network to control the particles further, and it could be customised by the user therefore this system is very flexible.

3.4 Houdini Digital Asset

The ability to execute procedural power in Houdini is very impressive, it allows user to custom nodes into digital assets that can be reused, which are named Houdini Digital Asset. These custom assets empower creators to wrap complex node networks into a single node and also build custom user interfaces on the node. The creators could also add notes to the built-in help cards to assist tool users to understand the HDA more. However, creating a clean and sleek network for the HDA could be quite dull and repetitive. But the outcome and the convenience brought by the HDA is rewarding.

4 Implementation

The shadow effect tool was established in Houdini, and most of the nodes used are *wrangle* nodes written in VEX. The VEX codes in *wrangle* nodes are used in order to handle the points of generated pattern curve. They also control the spreading of the effect across the whole model. The generated pattern was projected on to the model for the purpose of creating shadows on the surface of the given geometry. As for model consuming and decomposing effects, both are completed with particles.

4.1 Curve Generation

The generation of the pattern curves was written in VEX within the *wrangle* node in Houdini as mentioned in the previous paragraph. The *wrangle* node is mostly in the solver node, for the purpose of execute every frame, and for the reason that the status of the points of the curve will be affected by their previous status.

Users could select the pattern they would like to create and the starting points of the shadows. There are two patterns to choose, one is based on crystal-like method and another is diffusion-limited aggregation. For the starting points that the patterns grow from, users could choose random starting points or specific points they selected and grouped. They could also grow patterns from the model edge they selected and grouped.

4.1.1 Crystal-Like Method

The method used in the first pattern is to form crystal-like shapes, and the basic structure of the nodes was inspired by Horikawa's frost tutorial (2018), which is a 3D implementation of a 2D frost solver video from Grabovskiy (2015). The algorithm Grabovskiy (2015) used in the video was from Yuryevich's website (2012), which is about the pattern of ice crystals and snowflakes.

The approach to this implementation mainly focuses on the point positions and directions, and then combine the points into curve primitives. The tool will create points on the surface of the given model with the *scatter* node, these are the reference nodes for later creation.

First, users decide the starting points of the growth and choose the increment number of branches. Second, the *solver* will randomly grow branches to different directions to form the main branch, it is done by taking the position and direction of the nearest reference node and generate new branch points from them. Third, there will also be *wrangle* nodes in the *solver* growing sub branches, which are named as *fork* group, on the main branch, and the way to create new points for sub branch is the same as main branch. Last, adjusting the *pscale* attribute to control the size of points for converting curves to model primitives by the *polywire* node after the *solver* node's creation.

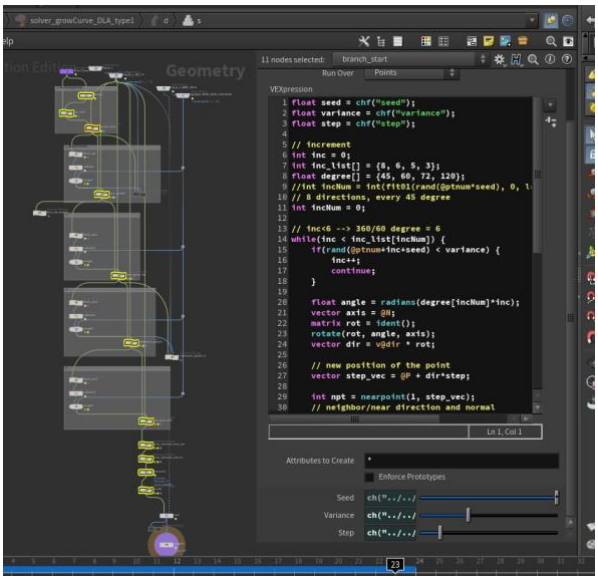

Figure 4.1: Crystal-like method network

The selected nodes in Figure 4.1, which are the yellow ones, illustrates the implementation. And the result of generated shadow curves and primitives are shown in Figure 4.2.


Figure 4.2: Result of Crystal-like method

4.1.2 Diffusion-Limited Aggregation

The technique for the second pattern is called diffusion-limited aggregation, also known as DLA. The reference of the execution of this method are mostly 2D implementation, one is written in C++ by Macey (2020), one is written in Python by QuantitativeBytes (2020), and another is written in Processing and JavaScript by The Coding Train (2016). However, there are some 3D references from a website which is originally a paper (Bourke 2014), and their outcomes are the reference for the final result of this project.

The execution of the DLA method is adjusted, instead of having particles wander around randomly until they attach to the main structure, particles growing randomly from the main structure is the approach in this project. The particle creation is also done in the *solver* node and *wrangle* nodes. Similar to the first pattern, the main branch is created first and each point on the main structure has a chance, which users could control, to grow sub branches. And the points on sub branches could also grow more sub branches the same way as the main branch. The points will be connected and become curve primitives. There are also points created on the surface of the given model as references for the generation of the shadow curve points, in order to match the shape on 3D surfaces. The *pscale* attribute of the generated points are adjusted as well, for the purpose of controlling the thickness of later created shadow model.

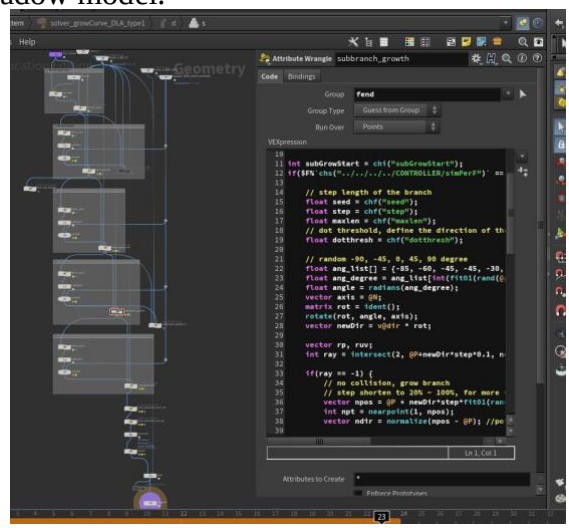

Figure 4.3: Diffusion-limited aggregation network

Figure 4.3 shows the nodes to create the shadow geometry, and Figure 4.4 illustrates the generated results.

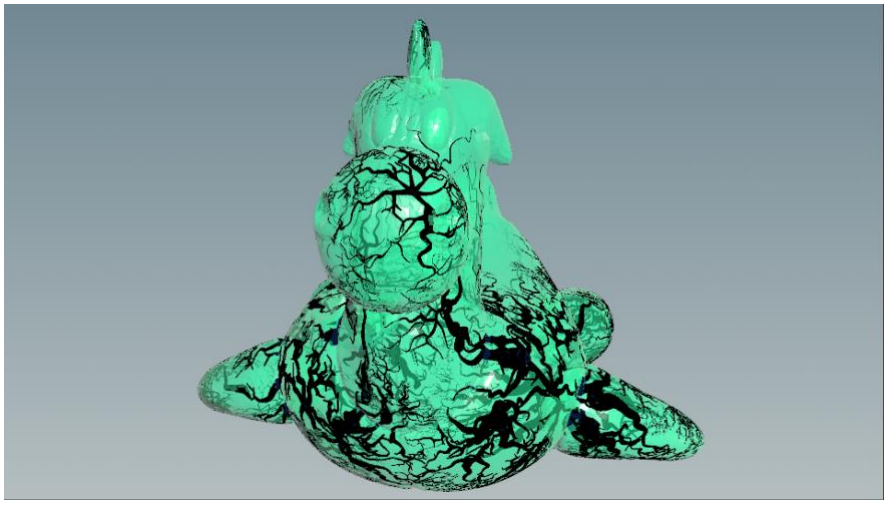

Figure 4.4: Result of Diffusion-limited aggregation

4.2 Projection

In an attempt to project the generated shadow geometries on to the model to achieve the effect that the shadow is growing on the model. The key node to accomplish this is the *ray* SOP node, the function of the node is to project the given surface onto another one (SideFX 2022). The *ray* SOP node is placed as the last node before the *output* node in the *solver*. This could ensure every newly created curve points of the shadow are correctly projected onto the model. The *Method* attribute in the node needs to be changed to *Minimal Distance* and the *Point Intersection Normal* attribute needs to be turned on. Figure 4.5 is the *ray* SOP node in the *solver* node.

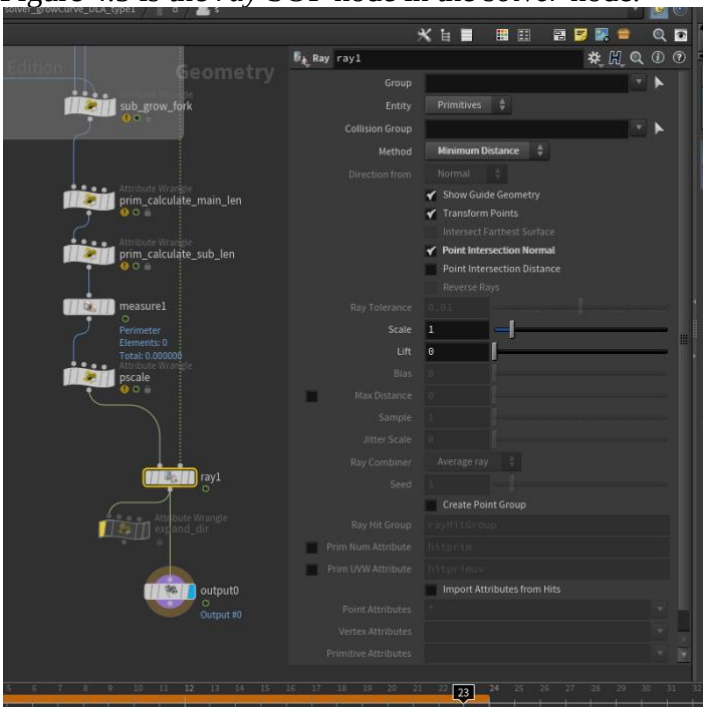

Figure 4.5: Ray SOP node settings in solver node

Another *ray* SOP node is added outside of the solver node, after the curve is converted into model primitives by the *polywire* node. This is to assure the shadow geometry attach to the surface of the given model as shown in Figure 4.6 below.

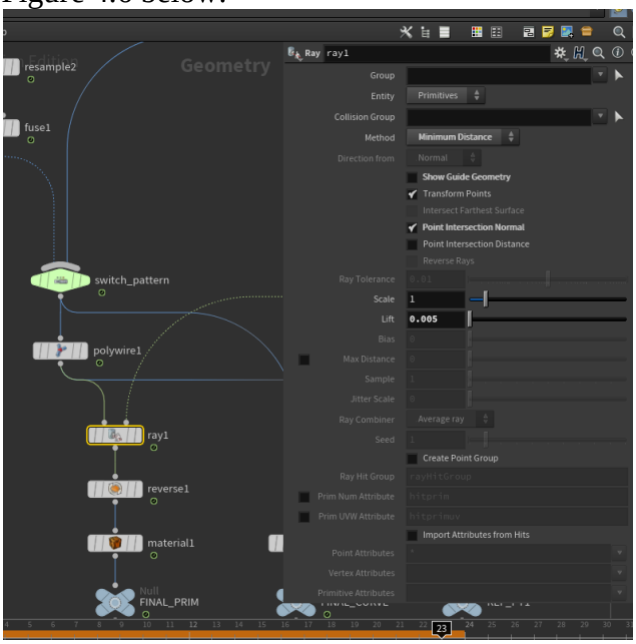

Figure 4.6: Ray SOP node settings outside of the solver node

4.3 Spreading

For the consuming effect in this project, it is managed by the *solver* node. It is an implementation of the combination of Yaşar's (2018) solver tutorial and Voxyde's (2022) attribute growth tutorial. The basic structure is to create points on the entire model surface with *scatter* node, and use VEX function *nearpoints()* in *wrangle* nodes to search points around the shadow shape and spread the effect across the given model. However, there are few attempts to create the starting points of the shadow shape.

First is to use the *group* node to select and transform points with in the shadow geometries. The second is to use the shadow curves and spread the effect to the points around the curves. The final attempt is to use the *pointsfromvolume* node to have the points that are shaped into the shadow geometry and then spread out across the given model. The final method has the best-looking shape as shown in Figure 4.7.

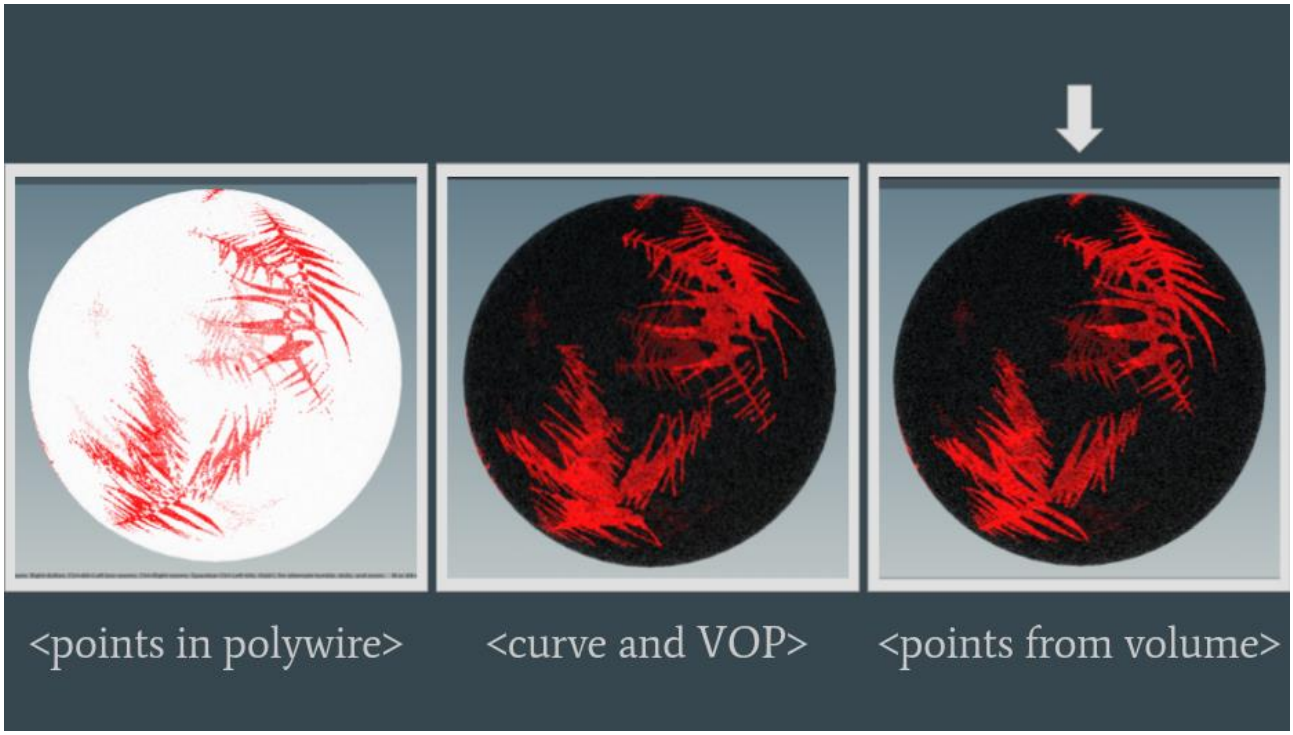

Figure 4.7: All attempts of particle spreading

4.4 Decomposition

As for the decomposing effect, the particle system in Houdini, which is the *popnet* system, was chosen. The initial idea after the given model decompose into particle dusts is that the particles will be wandering around the original place of the model. Therefore, creating swirling particles is the routine that was took in this project. The basic structure of the implementation is based on a swirly particle video uploaded by Entagma (2021).

The key point in this part is to add *velocity* attribute to the particles, and use the *mountain* node and *volumetrail* node to create the swirling path for the particles to follow. Nonetheless, the trail points for the *volumetrail* node is adjusted to creating points on the surface of the given model instead of within the model. This is to create the effect of the points are wandering around the original surface.

Figure 4.8 next page illustrates the final result of this effect.

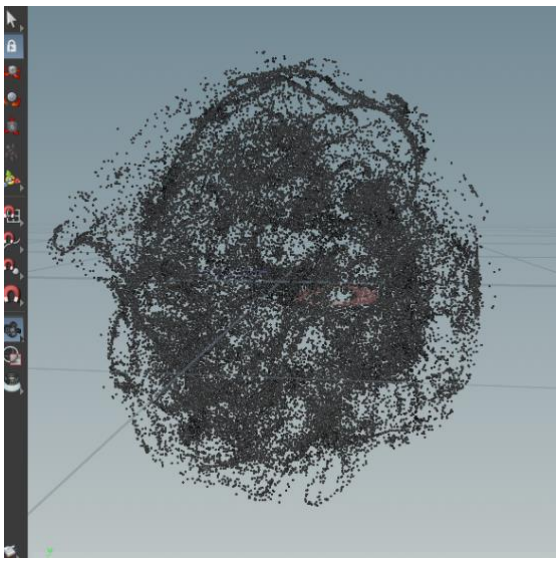

Figure 4.8: Swirling particle effect

5 Known Issues and Solutions

In the developing process of this project, there are some issues arise and cause some problems to the result. For instance, the branches of the pattern will intersect and collide with each other, and the simulation speed of growing patterns and spreading is quite limited. There are also some errors with creating shadow patterns on unclosed objects and moving objects. In the following sections, the problems mentioned above and the solutions to adjust them will be discussed.

5.1 Branch Collisions

Creating branches that grows in random directions will eventually lead to one result, that is intersection, the branches will collide and cross over each other. In order to fix the branch collisions, the *polyExtrude* node is added to node chain, along with a attribute *wrangle* node.

Firstly, the attribute *wrangle* node is added with the aim of moving all the points of the pattern curves backwards, and the direction of their movement is the normal of each point. Secondly, the function of the *polyExtrude* node is to extend the points along their normal direction and transform them into primitives. The *Extrusion Mode* attribute of the *polyExtrude* node should be set to *Point Normal* and *Existing*. And the *Distance* attribute could be set to twice of the length of the points movement, which is moving backwards in this case. Lastly, in the branch growth *wrangle* node that is created in section 4.1, a condition should be added before the creation of new branches.

There is a function in VEX named *intersect*, and the purpose of this function is to “compute the first intersection of a ray with geometry” (SideFX 2022). The function will return -1 value if the given ray does not intersect with any geometry. The geometry given into the function is the one created by the *polyExtrude* node, which is the pattern branches that are generated. Therefore, the *wrangle* node could only create new branches if the *intersect()* function returns -1 to ensure the new branch does not collide with the old ones.

Figure 5.1 below shows the chain of the nodes and the geometry created by the *polyExtrude* node for the purpose of computing intersections.

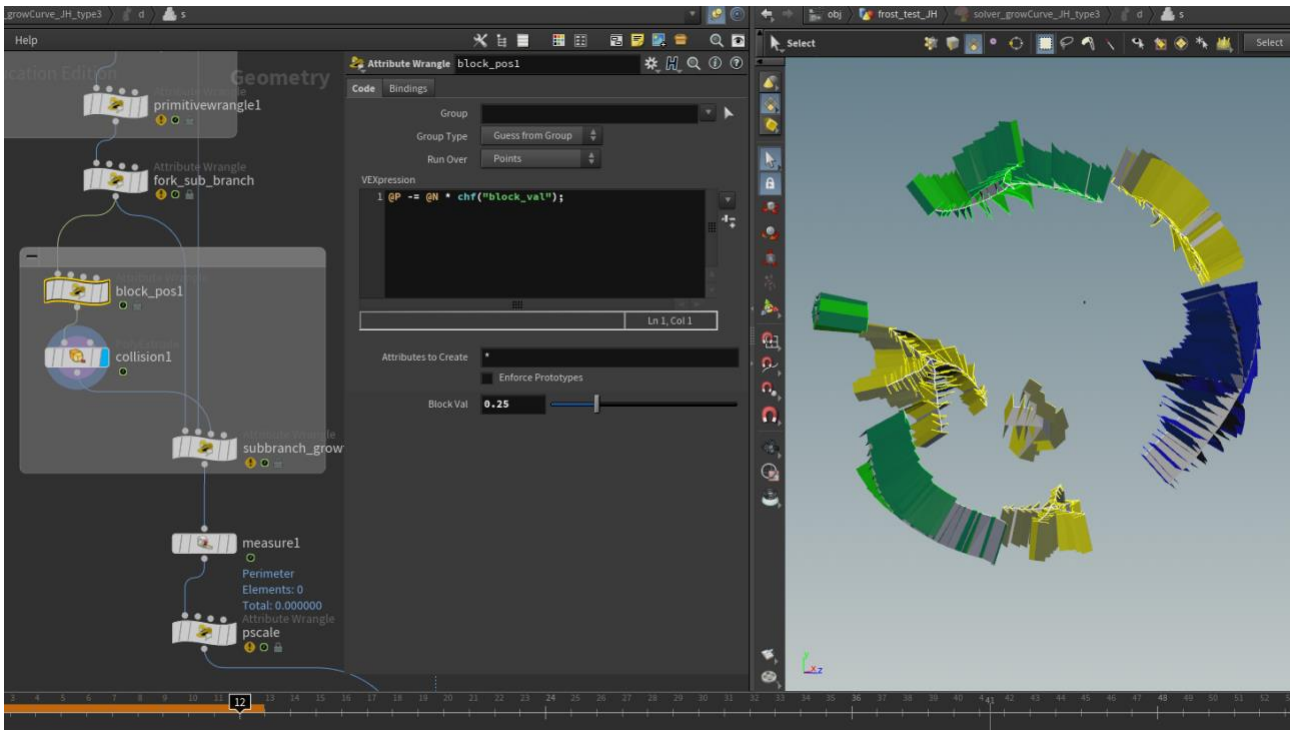

Figure 5.1: Node chain and geometries of branch intersection

5.2 Growth and Spreading Speed

The simulation of the pattern growth of the shadow and the consuming effects are all operated in the *solver* node. The speed of the simulations could be control by the *Sub Steps* attribute, which could decide “how many sub steps to break each frame into”. However, this attribute is not able to slow down the simulation, it can only accelerate the simulation by splitting each frame into smaller sections. In order to slow down the simulation, running every four or five frames for example, could not be done with the *Sub Steps* attribute in the *solver* node.

The solutions that could be found on the internet are usually contains the *retime* node or adjust the fps rate in the Global Animation Options, and others are fixed in the *popnet* system. Those approach could not fit in this project, it is using the *solver* node to simulate, not the *popnet* system. And changing the global fps rate is definitely not the correct attempt. The final method to adjust and slow down the simulation speed is actually not that complicated, it is accomplished by add a condition in all the *wrangle* nodes in the solver.

The key condition is to run the simulation every designated frame by the users, and the result is done by modulus. The mathematics operator *%* is used in the condition, using the current frame to modulo the designated frame, the number would be five for example if the user wish to run the simulation every five frame. If the result of the modulo is zero, then the *wrangle* node could proceed to run the simulation.

```
if($F%`chs("../../../../CONTROLLER/slowF")` == 0) {
    <run simulation codes>
}
```

5.3 Unclosed Object

For unclosed objects, there are chances that the creation of the shadow pattern could cause some issues. The generation might create new points of the pattern curve in the empty space as shown in Figure 5.2 below. And thus create the shadow geometry across the open space of the given model and create weird looking of the result.

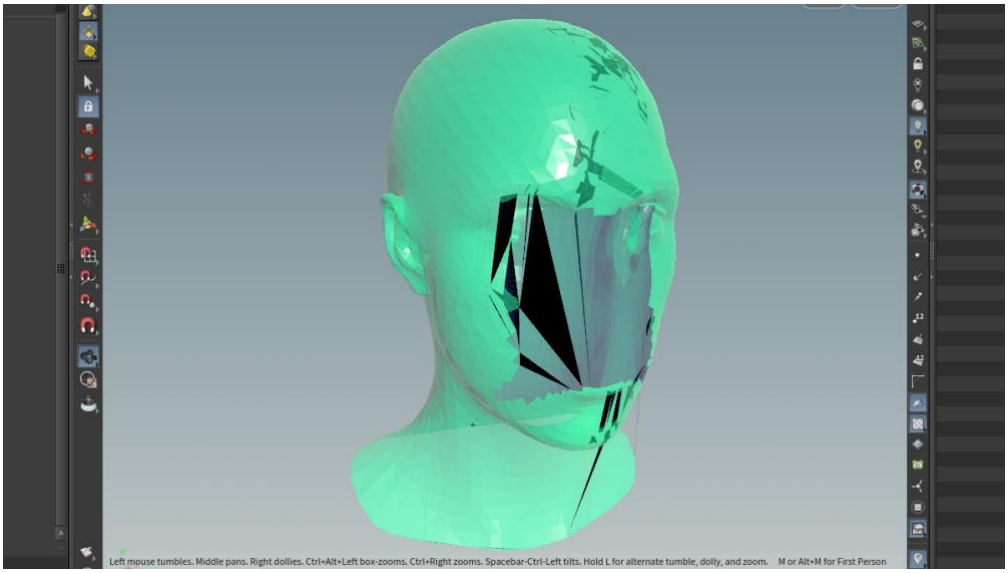

Figure 5.2: Misplaced pattern on unclosed object

The approach to solve this issue is by creating a geometry boundary to contain the generation of the curve points. The *group* node is used in this case, the *Base Group* is disabled and the *Include by Edges* is enabled, and the *Unshared Edges* should be turned on. The next step is to delete all other points and edges that are not on the unclosed/unshared edges. Follow by the step to extend the remain edges into boundary walls, the method here is the same as detecting branch collisions, which is using the *polyExtrude* node and the *intersect()* function.

The nodes used are illustrated in Figure 5.3.

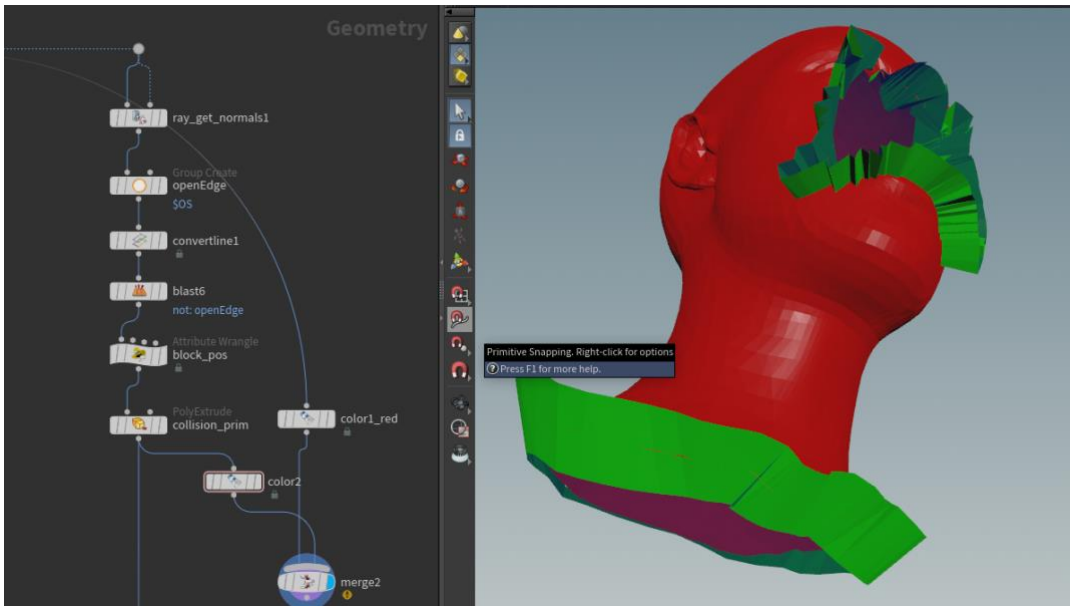

Figure 5.3: Visual implementation of solution. Red part is the original model, green part is the geometry to compute intersections

5.4 Uniform Scaling and Moving Object

The uniform scaling of the given model would not affect the result of the shadow geometries created by the tool. However, the shadow itself could not have any form of uniform scaling due to the reason that it depends on the given model entirely. And the shadow is projected onto the given model so adjusting the scale of the given model would be enough for this project.

Unfortunately, the shadow pattern will be misplaced when the given model is moving. This is because of the movement of the generated pattern does not follow the given model, and the nodes and implementations currently in this project could not fix this issue. As a result, the effect could only be used on static objects currently, and moving objects should be avoided. This part would be dealt with in further development in the future.

6 Results and Future Work

The finished shadow effect tool is separated into two digital assets, one is the creation of shadow geometry and the other is the generation of consuming effect and decomposing effect. The correct node chain is illustrated in Figure 6.1. The given model is the input for *ShadowPattern HDA* and the second, third and fourth outputs of the *ShadowPattern HDA* connect to the first, second and third inputs of *shadowConsumingAndDecomposition HDA* respectively.

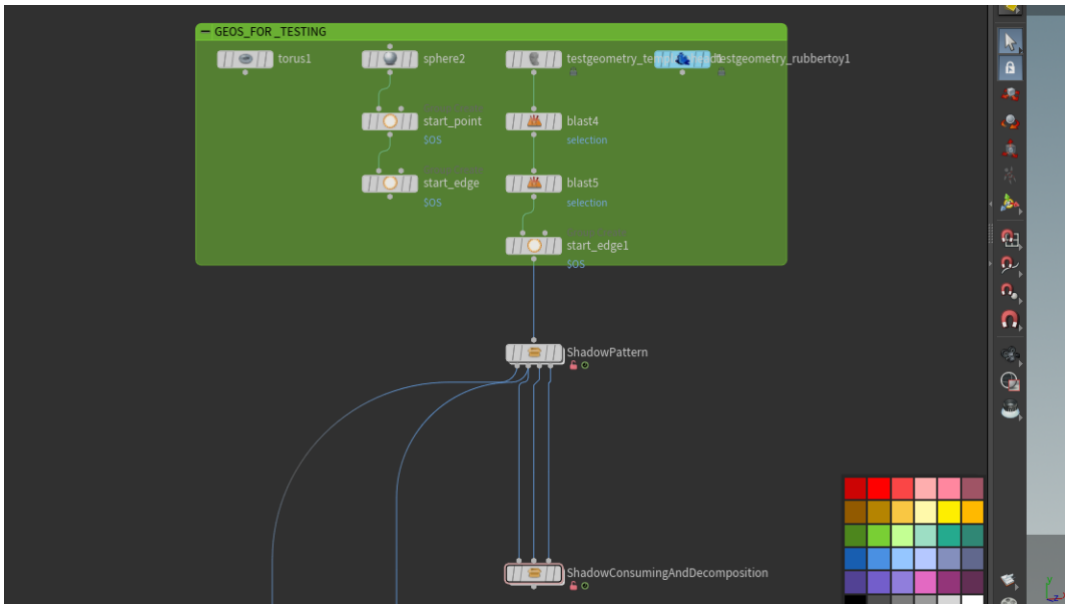

Figure 6.1: The correct connection of the HDAs

6.1 Shadow Pattern

The given model in the demonstration is the rubber toy test geometry in Houdini. First, the pattern inspired by the DLA method is shown in Figure 6.2, and it grows curves from random points or selected points. The shadow geometries shown in Figure 6.3 are also an implementation of DLA, but it grows from selected model edge, which is an execution of line attractor of DLA method.

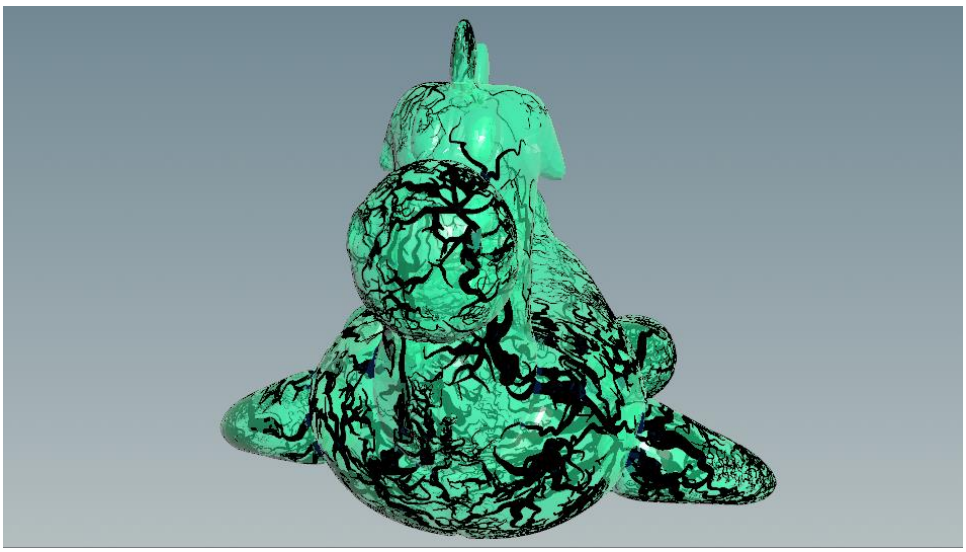

Figure 6.2: Result of DLA grow from random points

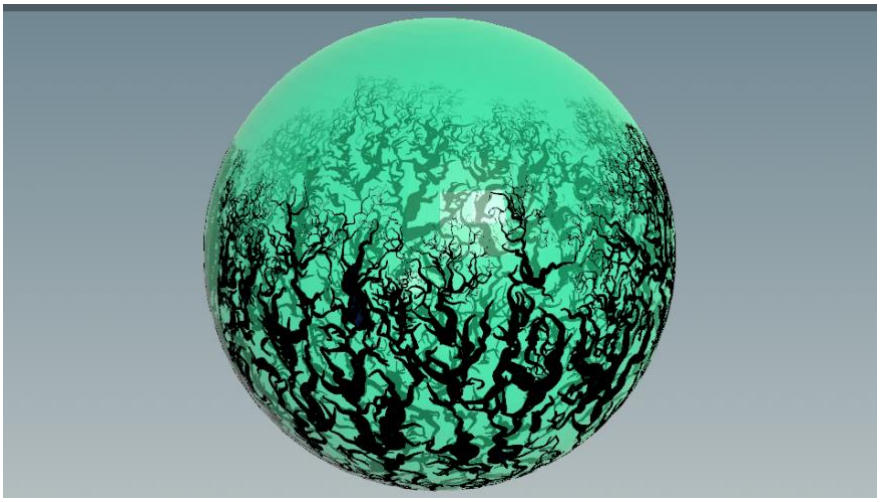

Figure 6.3: Result of DLA grow from selected edge

Second, the idea of the other pattern is inspired from crystals and snowflakes as shown in Figure 6.4 below. The basic structure of this method is an execution of Horikawa's frost tutorial (2018), and there are some adjustments added to it. For instance, users can select the increment number of branches.

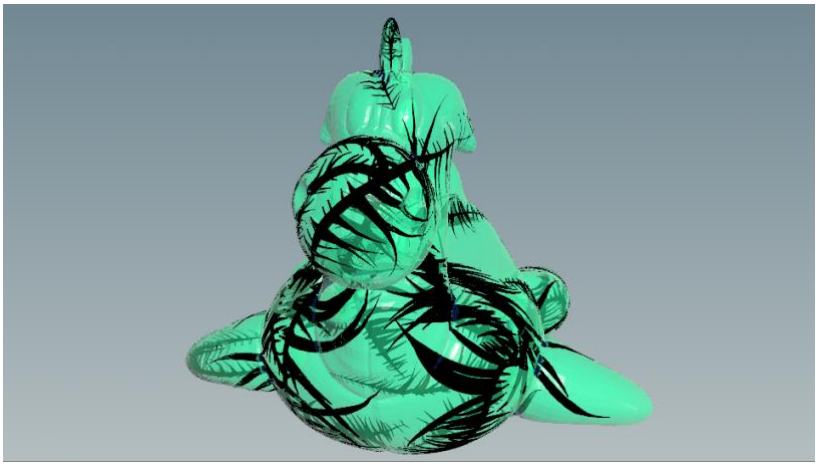

Figure 6.4: Result

Last, Figure 6.5 illustrates the implementation of the shadow effect on other models that have different shapes.

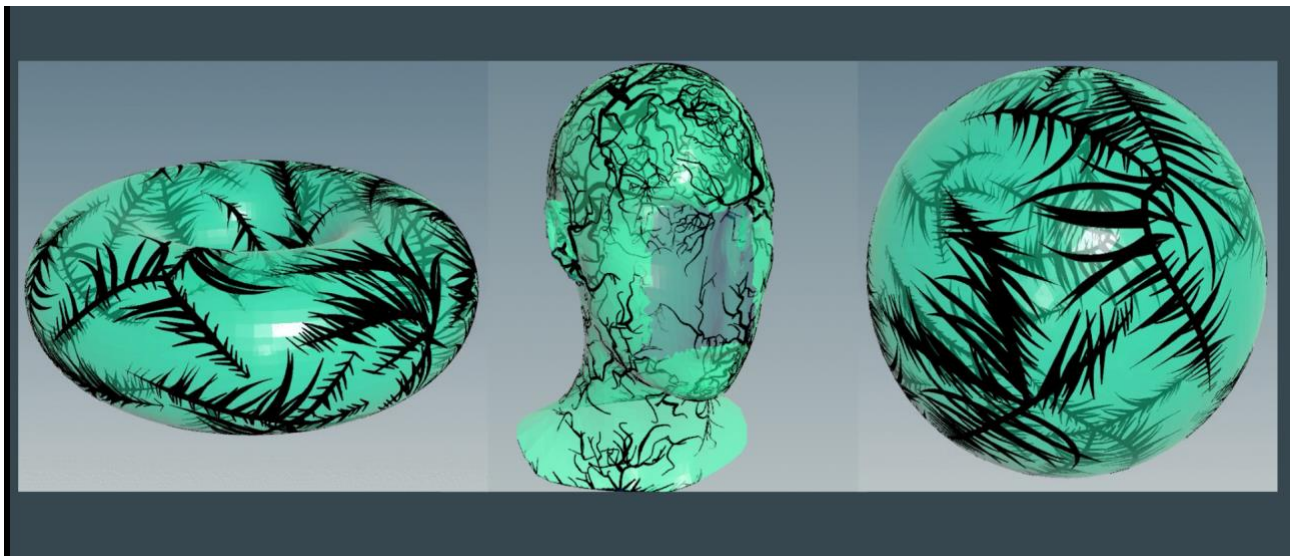

Figure 6.5: Result on other models

6.2 Consume and Decomposition

For the spreading and decomposing effects, both are accomplished with particle in Houdini. The consuming effect is done with the *solver* system and the decomposing effect is particle system, which is the *popnet* system. The given model in the demonstrations below is also the rubber toy test geometry in Houdini. Figure 6.6 shows the effect of particles spreading across the entire model in the shape of shadow patterns.

Figure 6.6: Result of red particles spreading with the shadow geometry and given model

As for decomposing effect, the *velocity* attribute was added to the particles in order to create the swirling movement. This is inspired by Entagma's tutorial of swirly particles (2021). Instead of creating particles within the geometries, the approach in this project is to create particles on the surface of the given model to make the effect looks like it is dissolving from the surface. The result is in Figure 6.7 below.

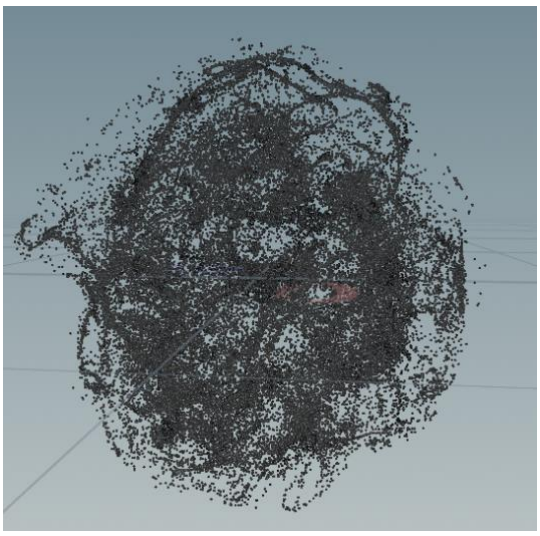

Figure 6.7: Result of swirling particles

6.3 Future Work

Although the assets are complete and finished with portable HDAs, there are several areas and potential future work could be done to improve the shadow consuming effects tool. There are also some minor flaws need to be fixed for better performance.

More patterns could be added to this tool to grow the shadows. Glass fracture is beautiful shape and it could make the given geometry looks broken and more interesting. Iterated function system, also known as IFS fractals, is another algorithm that could construct self-similar fractals that would be fascinating to try.

For the issue that need to be fixed, it is that the current asset could not work on moving object, it is best to create shadow geometries on static object. The shadows generated by the asset could not follow the moving object completely, mainly due to unchanged position and grow direction of the pattern curve. Therefore, the structure of the shadows would be manipulated into weird shapes, as shown in Figure 6.8 below. Adding a *vector* attribute to record previous position to each point of the curve could be a possible solution. Another method that could solve this is to record the points' positions at the start frame of simulation and the distance difference every frame, similar to acceleration, and adjust the shadow position with those data.

Figure 6.8: Result of wrong shadows caused by moving object

7 Conclusion

In general, this project could be considered as a success due to the achievement of initial design of creating digital assets to generate shadow effects. That includes generating shadow geometries with branch patterns and the effect spreading across the model to consume it entirely and decompose into particle dusts. The developed digital assets could be separated into two parts. First, the digital asset *ShadowPattern HDA* is the simulation of growing shadow patterns, and users are able to choose between two different overall patterns and three different starting shapes. Second, the other digital asset, which is *ShadowConsumingAndDecomposition HDA*, is to create the consuming effect by spreading particles match with the pattern shape across the entire model and generate swirling particles as the decomposition result. There are some properties that allow users to control, such as simulation time step, random simulation points, spreading size and particle swirl velocity. The assets also provide the choice to show only the shadow pattern curves and show shadow geometry after simulation, this could reduce the time for users to find the shape they want.

Both HDAs have their own help card to assist users' understanding of the effect tool, and the overall result of the tools could create shadow geometries with branches and consuming and decompose effect on the given model.

Although the complete development was successful, there are still some limitations and flaws with the assets. The main issue in this effect tool is that the assets currently could only work on static objects, it is not a good idea to use the assets on moving objects. The generated shadow pattern will not 100% follow once the model starts to move, and this will create weird shadow shape on the given geometry. The growing direction of the pattern is also unchanged when the geometry is moving. The method could be a possible solution to improve the developed assets in this case is to add another *vector* attribute to each point of the pattern curve to record the previous position. Therefore, the asset would be able to calculate the distance difference between current frame and previous frame and could possiblely adjust the position difference cost by moving the object. Another approach is to record the distance different directly, similar to acceleration, and add it to the position and grow direction of every point. However, this might not work with model rotation and this need some further research and testing. Overall, these will be part of the future development to extend and complete the shadow tool.

Bibliography

Bourke, P., 2014. *DLA - Diffusion Limited Aggregation* [online]. Available from: http://paulbourke.net/fractals/dla/ [Accessed 20 August 2022].

Entagma, 2021. *Guest Tutorial: Simon Fiedler - Controlling Swirly Particles* [video, online]. YouTube. Available from: https://www.youtube.com/watch?v=yqM_3goH4J8 [Accessed 19 August 2022].

Grabovskiy, A., 2015. *Houdini frost solver intro* [video, online]. Vimeo. Available from: https://vimeo.com/141886207 [Accessed 09 August 2022].

Horikawa, J., 2018. *[Houdini Tutorial] 0033 Frost Wrapper (Slow version)* [video, online]. YouTube. Available from: https://youtu.be/4I3zVgGuSmw [Accessed 09 August 2022].

Kishimoto, M. (Writer), 1999. *Naruto* [manga, tv shows]. Pierrot Co., Ltd. [Accessed 15 July 2022].

Lindenmayer, A., 1968. *Mathematical models for cellular interaction in development, Parts I and II*. Journal of Theoretical Biology, 18:280–315.

Macey, J., 2020. *Lab 4 Diffusion Limited Aggregation | Jon Macey's WebPages* [online]. Jon Macey's WebPages. Available from: https://nccastaff.bournemouth.ac.uk/jmacey/msc/ase/labs/lab4/lab4/ [Accessed 21 August 2022].

Prusinkiewicz, P., Lindenmayer, A., 1990. *The Algorithmic Beauty of Plants*. New York: Springer-Verlag.

QuantitativeBytes, 2020. *Diffusion-limited aggregation and modelling frost (Python)* [video, online]. YouTube. Available from: https://www.youtube.com/watch?v=BD6s_gCAiAU [Accessed 22 August 2022].

Ramsey, P. (Director), 2012. *Rise of the Guardians* [Film]. DreamWorks Animation. [Accessed 23 July 2022].

SideFX, 2022. *intersect VEX function* [online]. Available from: https://www.sidefx.com/docs/houdini/vex/functions/intersect.html [Accessed 10 August 2022].

SideFX, 2022. *Ray geometry node* [online]. Available from: https://www.sidefx.com/docs/houdini/nodes/sop/ray.html [Accessed 05 August 2022].

SideFX, 2022. *Solver geometry node* [online]. Available from: https://www.sidefx.com/docs/houdini/nodes/sop/solver.html [Accessed 11 August 2022].

SideFX, 2022. *Vex* [online]. Available from: https://www.sidefx.com/docs/houdini/vex/index.html [Accessed 05 August 2022].

The Coding Train, 2016. *Coding Challenge #34: Diffusion-Limited Aggregation* [video, online]. YouTube. Available from: https://www.youtube.com/watch?v=Cl_Gjj80gPE [Accessed 24 August 2022].

Voxyde, 2022. *VFX Essentials - Attribute Growth / Propagation - Houdini Tutorial* [video, online]. YouTube. Available from: https://www.youtube.com/watch?v=0FX7tcrYR_0 [Accessed 14 August 2022].

Witten Jr., T. A., Sander, L. M., 1981. *Diffusion-Limited Aggregation, a Kinetic Critical Phenomenon*. Phys. Rev. Lett.47, 1400 – Published 9 November.

Yaşar, S., 2018. *Houdini Growth System With Sop Solver (Turkce)* [video, online]. YouTube. Available from: https://www.youtube.com/watch?v=hFWHtM1f5bo [Accessed 14 August 2022].

Yuryevich, S. V., 2012. *Illustrations for " The Riddle of Water. Ice Crystals, Snowflakes "* [online]. Available from: http://samlib.ru/img/s/skosarx_wjacheslaw_jurxewich/zagadkaproishozhdenijawody-1/ [Accessed 10 August 2022].