Pipeline Development for Winter Scenes and FX

Bournemouth University

Georgi Hristov Beshovski
MSc Computer Animation and Visual Effects
20th of August, 2018

Contents

1. Abstract

Simulating granular based materials has been considered a computationally heavy task in computer graphics for a long time. Given the limitations that a granular system presents, these materials appear unique properties that sometimes are difficult to be simulated with physical accuracy. Sand is a granular material that has offered some great effects in computer animation (Papadimitriou 2012).

One major drawback is the expensive simulation. There is a large number of particles needed to interact with each other in order to create photorealistic effects. In this master thesis 2 pipelines will be observed, designed and implemented. The first is a set of tools created in a pipeline to take any object or up to 4 objects and cover them with snow, enabling quick creation of winter scenes. It includes a tool to provide a lighter polygonal mold of how the snow would look. The Volume generator is included with artist friendly tools to allow for a wide array of customization in order to improve tool flexibility and usability. The second pipeline utilizes a paper on snow crystallization to create procedural snowflakes which are then mapped onto a particle system which behaves like falling snow to bring a higher level of realism to the snowfall effect.

2. Introduction

In recent years there has been a significant increase of interest in simulating the dynamic motion of granular materials, such as sand, soil, powders and grains. In computer graphics world there has been always the demand of animating such materials, but their unique natural properties have made this task a challenging procedure (Papadimitriou 2012).
In the Houdini set of pre-defined assets there is a wide variety for creating sand simulations, however there is a lack of tools that deal with development for snow-based systems. In this Project a pipeline development of tools which would enable artist to create winter scenes easier was designed. The main focus was on creating tools and systems which would enable artist to create winter scenery and snow-based effects with the least possible problems.

Especially, sand is a granular material that is met in many landscapes and its interaction with the environment has been researched for many years. Animating scenes with sand in computer graphics has become an interesting topic of research as the computational models for this should be highly configurable with rich visual behaviour and in as low as possible computational cost. (Papadimitriou 2012).

In production, methods to decrease production time are crucial and in this final year project tools that were deemed helpful for the creation of modules within the pipeline were encapsulated. This way should an artist have the need to create a winter scenery they can use the designed modules and decrease the development process significantly and reuse the modules as many times as required. In the next chapters the methods, elements and tools designed are discussed.

The first scenario considered is how to speed up a pipeline in which the artists have a scene or group of objects which need to be covered with snow for a winter scene. The pipeline takes the object and through 3 stages creates the desired effect.

- Stage 1 is the artist driven selection process. The artist imports the objects they wished to be worked on. Once setup of the object is complete the surface to be covered of the object with snow is determined including the area of coverage, the angle of coverage relative to the objects normal and for further flexibility a paint node to paint areas where no snow is desired. Once the area is decided the object has a polygonal mold created to show a relatively high detail approximation of the shape of the snow at the end. The mold comes with its own set of customizable tools to increase snow clumping or smoothness what noise algorithms to apply and how detailed the mold should be. This way the artist is provided with a lighter visual reference of what the end result will look like. For example, an object using 2-3 million grain particles can be represented fairly easily by 80-100-thousand-point polygon.

- Stage 2 is the conversion of the Polygon into grains in order to add the heavy physics calculations to the snow. The stage also introduces the surface shader.
- Stage 3 is the Volume shader which provides the essential effects for creating realistic looking snow by adding transparency and subsurface light scattering tin order to give the impression of the crystalized snow surface.

The second scenario is how to implement a raining snow simulation which utilizes snowflakes rather than spherical particles. This is done in 2 or 3 stages depending on the approach required.

- Stage 1 is the snowflake procedural generator asset which creates a snowflake based on equations found in the paper of [Reiter, C. (2004)] and the masterclass [Creating A Procedural Snowflake]
- Stage 2 (optional) Use a snowflake generator to create slight of the snowflake created in Stage 1
- Stage 3 either utilize the lighter system with one snowflake mapped to every particle in a particle simulation or the considerably heavier multiple particle system which in its current form uses a random generator to choose between 6 different snowflakes however can be expanded depending on production requirements.

In section 3, an overview of the previous work researched in the field is presented. In chapter 4, the main systems initially researched. In chapter 5, the different tools developed are analysed and illustrated. In chapter 6, a general conclusion is given in regards to the developed 2 pipelines, their applications advantages, disadvantages and possible improvements as well as difficulties encountered and problems with the current implementation.

3. Previous work

A lot of research into creating realistic looking granular based scenes has been done throughout the years. In computer graphics granular effects such as sand or snow have always been difficult to create at a massive production scale.

The previous work in this field can be divided into particle-based methods and mesh-based methods. (Papadimitriou 2012).

In previous master’s project there have been a number of projects simulating various types of grain materials such as sand and snow and utilized different methods such as Position Based Dynamics and Material Point Method. This however means that they focused on the internal forces and implemented solvers for Houdini creating sophisticated simulations utilizing not only aesthetic but also robust and physics accurate solutions. The main problem with them is that they are focused on the creation of a single concept and do not concern themselves with the rest of the pipeline.

This work is mainly in the starting point of the work pipeline and although essential in the development process is not the only one. In this thesis the next steps are observed, once you have a system in this case the Houdini Rigid Body Dynamics grain solver what tools can be created to speed up asset and scene creation.

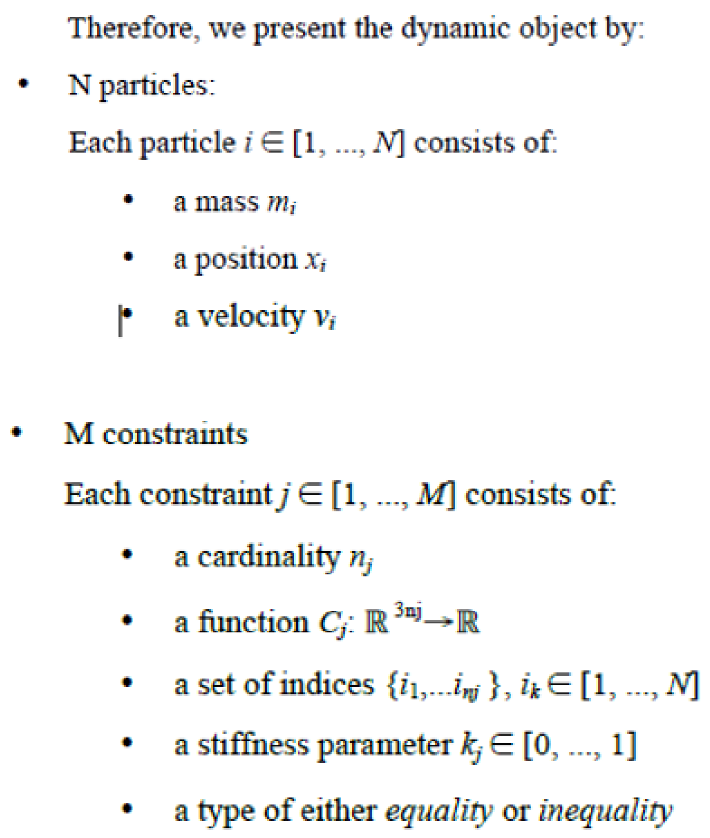
4. Technical background

4.1. Position Based Dynamics

When it comes to dynamics simulation most methods are implemented using force -based algorithms. They establish the internal and external forces to which the simulation is subjected to and calculate the acceleration through Newtons second law of Physics through an integration method of time in order to update the particle velocity. In Graphics algorithms are either focused on accuracy or illusion of realism. The first type is for example utilized in engineering scenarios in order to create realistic scenarios for real machinery to be tested in, these simulations are focused on stability, accuracy and robustness. The PBD implemented in Houdini is classified as the second type of method the ones who are focused on looking as aesthetically pleasing as possible. Houdini's standard assets have utilized Position Based Dynamics (PBD) which works directly on the positions of the particles in order to easily manipulate the simulation. The use of PBD is widespread in video games and interactive application thanks to its simplicity and stability. The internal components of the Houdini asset are built entirely in VEX which is a multi-threaded high-speed scripting language inbuilt in Houdini.

Position Based Dynamics is a technique used for simulating dynamics that computes the position of a group of particles or vertex over time. Then, the velocity is updated as the difference of positions. These positions change affected for the internal and external forces. (Orel 2016)
The major advantage of PBD is how simple the computation of the internal forces are to estimate such as elasticity, pressure or viscosity, into constraints enforced onto the positions of the particles.
The algorithm is then modified by external forces such as wind or gravity. The external forces determine possible final locations for the particles for each and every step and then compute the correct end based on the internal forces acting within the particle.

The constraints used depend on the effect that we want to
simulate. For example, the constraints used for grain materials are different from the constraints for cloth simulation. (Orel 2016)



Therefore, we present the dynamic object by:

- N particles:

  Each particle $i \in [1, ..., N]$ consists of:

  - a mass m_i
  - a position x_i
  - a velocity v_i

- M constraints

  Each constraint $j \in [1, ..., M]$ consists of:

  - a cardinality n_j
  - a function C_j: $\mathbb{R}^{3nj} \rightarrow \mathbb{R}$
  - a set of indices $\{i_1, ... i_{nj}\}$, $i_k \in [1, ..., N]$
  - a stiffness parameter $k_j \in [0, ..., 1]$
  - a type of either *equality* or *inequality*

The way Position Based Dynamics simulate the dynamic objects for each time step t are defined in the following pseudo code defined by (Müller, 2007):

```
(1)  **forall** vertices $i$
(2)     initialize $\mathbf{x}_i = \mathbf{x}_i^0, \mathbf{v}_i = \mathbf{v}_i^0, w_i = 1/m_i$
(3)  **endfor**
(4)  **loop**
(5)     **forall** vertices $i$ **do** $\mathbf{v}_i \leftarrow \mathbf{v}_i + \Delta t w_i \mathbf{f}_{ext}(\mathbf{x}_i)$
(6)     dampVelocities($\mathbf{v}_1, \ldots, \mathbf{v}_N$)
(7)     **forall** vertices $i$ **do** $\mathbf{p}_i \leftarrow \mathbf{x}_i + \Delta t \mathbf{v}_i$
(8)     **forall** vertices $i$ **do** generateCollisionConstraints($\mathbf{x}_i \rightarrow \mathbf{p}_i$)
(9)     **loop** solverIterations **times**
(10)       projectConstraints($C_1, \ldots, C_{M+M_{coll}}, \mathbf{p}_1, \ldots, \mathbf{p}_N$)
(11)    **endloop**
(12)    **forall** vertices $i$
(13)       $\mathbf{v}_i \leftarrow (\mathbf{p}_i - \mathbf{x}_i)/\Delta t$
(14)       $\mathbf{x}_i \leftarrow \mathbf{p}_i$
(15)    **endfor**
(16)    velocityUpdate($\mathbf{v}_1, \ldots, \mathbf{v}_N$)
(17) **endloop**
```

The algorithm executes the pseudo code in the following steps:
Lines 1-3: The features of the vertices *i* are initialised
Line 4: Loop the calculations for all parameters
Line 5: Compute v for all external forces
Line 6: Damp the velocity
Line 7: For all particles calculate the possible next positions
Line 8: For all particles generate collision constraints
Line 9-11: The solver manipulates the possible positions in order to satisfy the constraints.
Line 12-15: Update to the next position and the final velocity
Line 16: Update final velocity

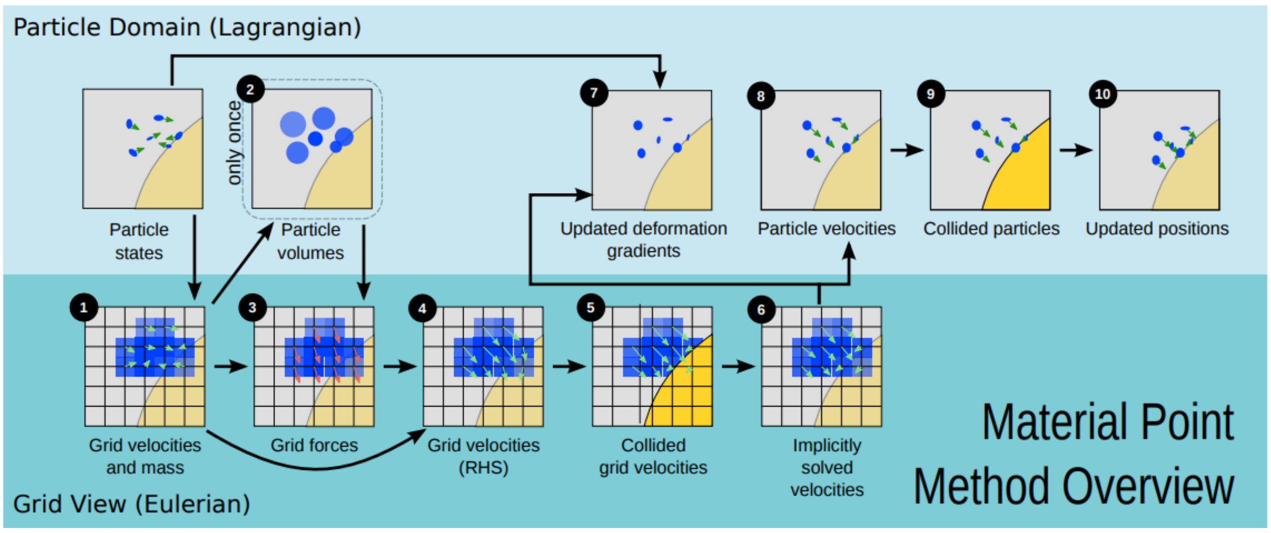
The algorithm uses Jacobi Iteration instead of the Gauss-Seidel

That means that the candidate positions are not updated immediately, and rather than projecting one constraint after the other, all constraints are computed based on the same position. (Orel 2016)

4.2. Material Point Method

The Material Point Method is a Eulerian/ Lagrangian hybrid method that uses particles in combination with grids. It uses particle as its the primary representation. It has position velocity and deformation gradient. Presented by Stomakhin et.al (2013) as a new innovative method to create snow for the demands of Wald Disney Animation Studios Frozen. The Material Point method is a hybrid model where the particles are influenced by grid forces and similarly to FLIP fluids the particles carry all of the data such as position, velocity and temperature. The governing equations are solved on a stationary Eulerian grid, and steps at the beginning and end of a calculation are taken to interpolate the particle data to the grid and to update the particle data from the grid. In this way, MPM is able to combine the advantages of the mesh-based, grid-based and particle-based methods. (Sorensen 2016)

MPM is capable of solving engineering simulations creating the possibility of utilising physically accurate simulations. The Eulerian grid allows easy implementation of the boundary conditions, it also eliminates the need to calculate inter-particle collisions. The collisions are removed thanks to the assumption that particle on particle collisions will not occur while their velocities are calculated from the velocity field of the Eulerian grid.



The particle-based representation of the material allows for a way to simply deform the material without causing mesh entanglement and the representation through particles allows splitting and merging of objects with different materials and behaviours. This makes it an appropriate method for simulations like melting.
The Material Point Method Consists of 10 steps:

1. Rasterize particle data to the grid. The first step is to transfer mass from particles to the grid
2. Compute particle volumes and densities.
3. Compute grid forces
4. Update velocities on grid
5. Grid-based body collisions
6. Solve the linear system
7. Update deformation gradient
8. Update particle velocities
9. Particle-based body collisions
10. Update particle positions

(Stomakhin 2013)

4.3. Technical Background Possible Method Analysis

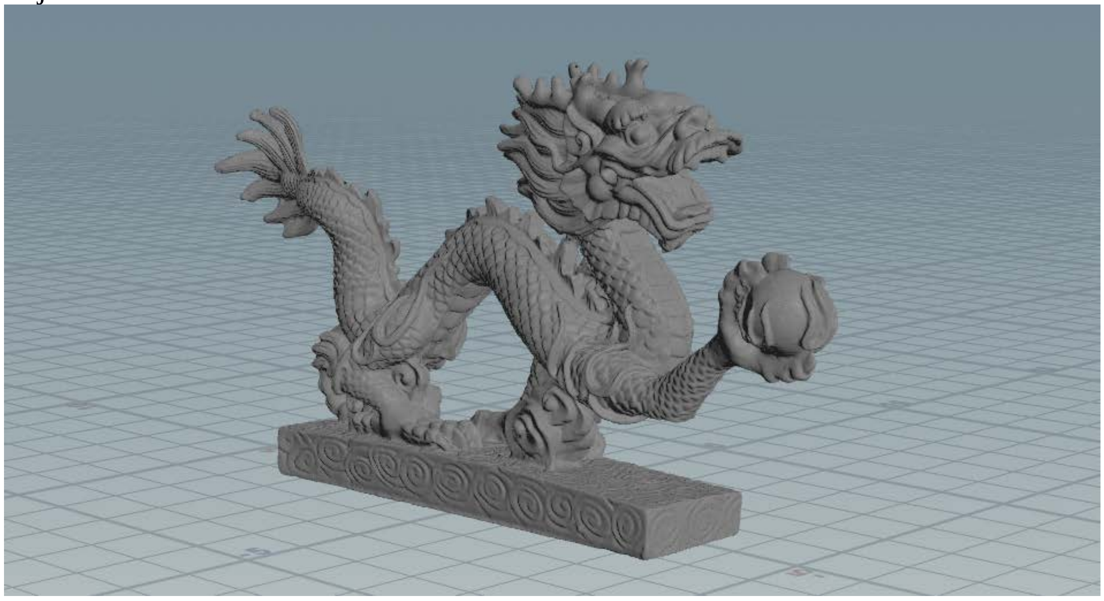
In the Initial stages of development, it became evident that a better understanding of the different type of possible simulations is essential. In the end after reviewing the possible outcomes from either system and comparing it to the work of previous master's students it was established that it would be very hard to establish a dynamics system as complete and thorough tested as the system currently in use in Houdini mainly due to project strict time constraints. It was decided that for the 2 pipelines the systems to be used were a modified Rigid Body Dynamics grain solver system for high detailed snow simulations and a particle system for snowfall simulations.

5. Pipeline Development

An artist should have some object on which to test the pipeline. Whether that is a terrain or an object makes no difference the only possible problems are the magnitude of scale. The pipeline allows for up to 4 objects to be worked on simultaneously due to restrictions placed on the subfolder nodes' number of inputs.

The chosen object for testing is a 3d scan of a dragon statue. If the pipeline can work on a high definition scan of a real-world item then it should be able to handle any other type of object.

Object to be tested:

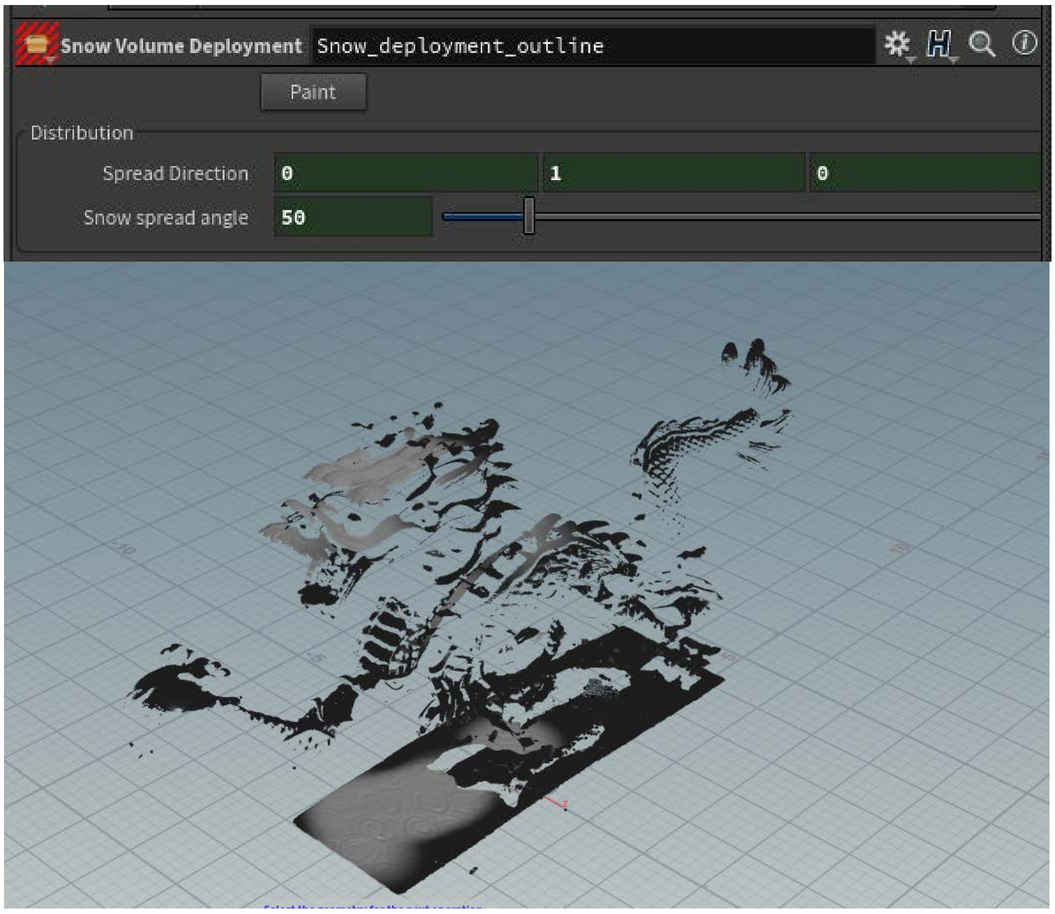


5.1. Snow Deployment Algorithm

The first part of the package of tools created for FX artists who wish to create winter scenes was the snow deployment algorithm. This is a tool designed to take up to 4 objects as input and generate the surface to be covered with snow. The tool is designed to take the objects and group all the surfaces which satisfy direction instructions.

The basic instructions are a spread direction vector which defines the general direction the snow will clump at. It allows for more abstract spreading of snow based on the objects input for example for Inception like bending effects where the snow has to spread at an angle. The second instruction is the spread angle which defines the covering percentage of the object for objects which need to be only partially covered or completely hidden from the viewer would have different spread angles.

The two variables provide general direction and control, however for improved flexibility of the tool a painter tool was implemented. Based on a painter tool from our course materials the artist can paint the areas of the objects they do not want to have any snow the painting tool has one basic rule all black areas will be fed into the next steps as input data any other colored area



Snow_deployment_outline

Distribution

Spread Direction 0 1 0

Snow spread angle 50

5.2. Snow Volume Estimation

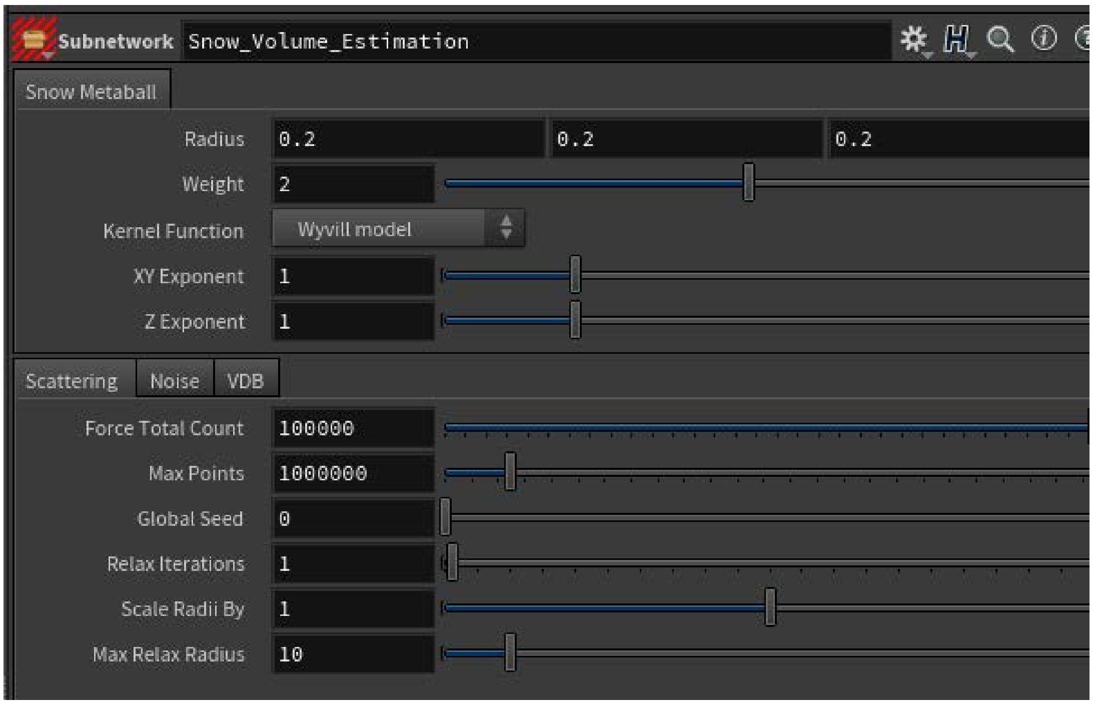
This is a tool used for creating a polygon mold of the snow. The tool is used after the snow deployment tool and used its output surface as input. It scatters points onto the surface to be covered with snow and generates metaballs over each and every point. The metaballs are then glued together, converted into a polygon and have noise applied to them in order to create a look similar to piled up snow.
In this tool once again, the artist has quite a few options to tailor the shape of the snow depending on their requirements.

The sections with tools are split into 4 categories.
The first is the metaballs details which would help adjust the granularity and the scale of the individual components of the snow allowing the artist to adjust the scale of the snow particles depending on the scale and requirements of the scene.

The second one is the scattering algorithm which handles how many points to utilize for spreading out the metaballs and can be tweaked to increase or decrease the level of detail depending on the magnitude of the scene it is applied to.

The third is the noise set of instructions where the user can adjust the type of noise algorithm as well as the variables related to the noise used for creating the snow shape

The last set of tools are the VDB tools which allow the artist to edit the level of detail and the type of VDB setting s either level of detail or divisions per span to adjust the VDB to their needs.

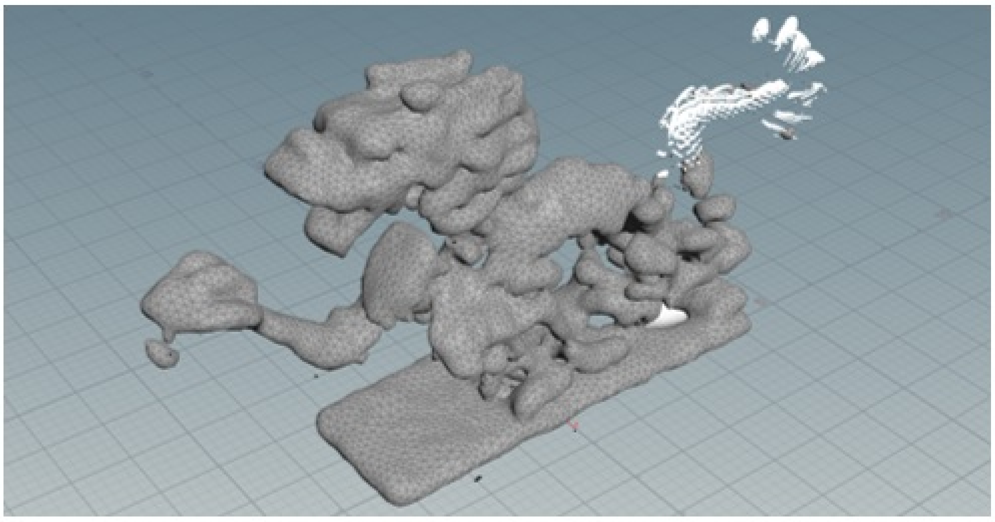

Subnetwork Snow_Volume_Estimation

Snow Metaball

Radius	0.2	0.2	0.2
Weight	2		
Kernel Function	Wyvill model		
XY Exponent	1		
Z Exponent	1		

Scattering | Noise | VDB

Force Total Count	100000
Max Points	1000000
Global Seed	0
Relax Iterations	1
Scale Radii By	1
Max Relax Radius	10



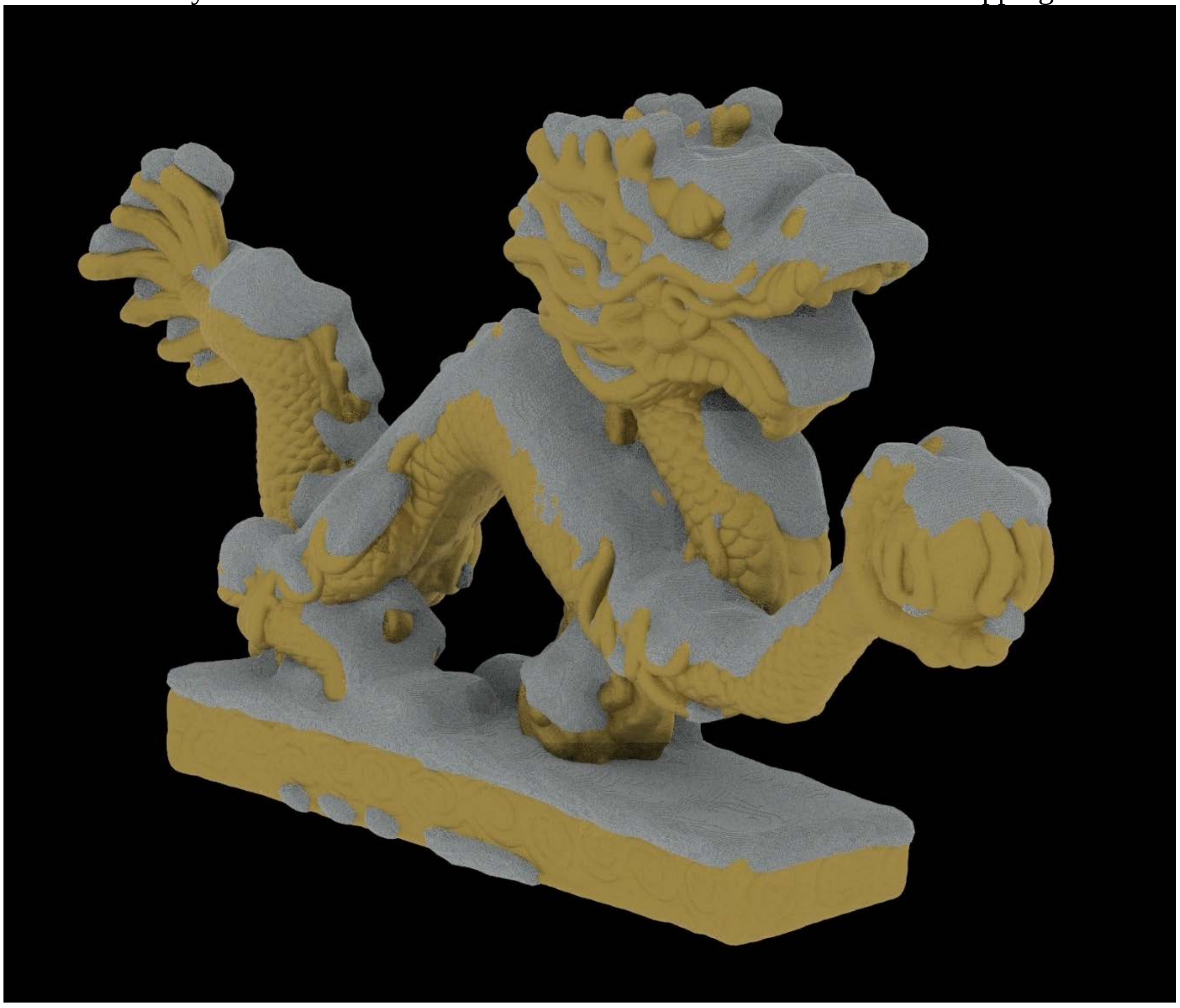
In the above example the Artist has decided to remove all snow from the Dragons tail.

5.3. Snow Grain Solver

Once the Polygonal shape is created the Snow solver can be called to convert the polygon into a grain-based shape adding the RDB physics with a new variation of the basic shader used in the original grain solver. Originally the grain solver was edited with the intention of replacing the VOP modules which have an extra layer for artist development. Removing the layer was believed to allow for a more optimized solver and faster execution of the code. Initially progress was satisfactory however the internal DOP nodes had internal constraints which were not possible to reproduce and the modules were unable to connect to the inbuild it modules. After several tests it was discovered that the VOPs are altered by Houdini to operate directly though VEX thus making the optimization unnecessary as it had been already implemented.

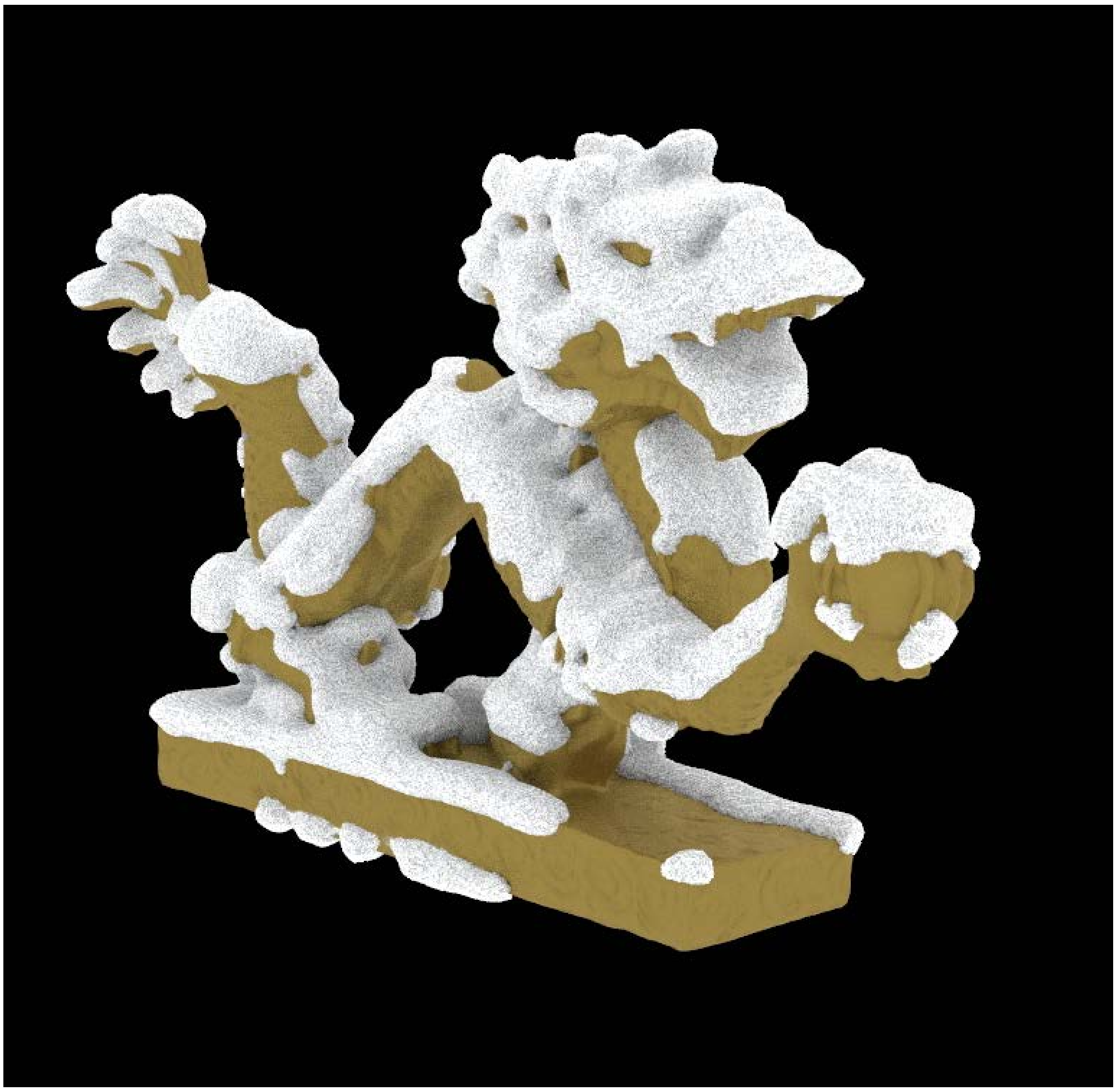
Further changings were attempted including a change from SHOP sprites for the grains to MAT shader of the individual grains however there was a requirement that nothing but SHOPs can work with it.

Finally, after several attempts to write a new shader the grain color and sprite were tweaked in order to give the particles a snow like look. Initially it was intended for this shader to show all of the required sub surface scattering and transparency characteristic for snow however in the end the only case in which the solution worked was with 2 shaders overlapping.



5.4. Snow Shader

The snow Shader is a combination between a surface shader and a volume shader. This technique was chosen in order to recreate the characteristic slightly transparent snow shader and allow for subsurface scattering seen in clumped snow.



5.5. Procedural Snowflake

Snow crystals show intriguing six-fold symmetry with a seemingly endless possible variation. The diversity of natural crystals creates a daunting task of how to recreate them in computer graphics. The goal of the algorithm is to create a simple local model in which from a point of origin a snowflake can grow and exhibit a variety of two dimensional forms similar to the ones seen in snow crystals.

However, while we use general physical notions to design our algorithm, we are not trying to fit equations of the physics. The model we give is two-dimensional,
so, we cannot hope to model the forms of snow crystals that have substantial three-dimensional components (Reiter 2004).
This allows for the simulation of snowflakes exhibiting growth of crystals with sector stellar, dendrite and plate forms.

The initial method implemented is characterized as a Boolean model for snowflake growth that evolves in time through the use of hexagonal lattice by the defined set of rules.
The algorithm will determine at each time step for each and every sell whether it is ice or not. On the next step cells that were ice will remain ice while new cells that were not ice may become ice only if and only if exactly one of the neighboring cells is ice.

While that model provides
examples of abstract plates and sectors, it does not provide global dendrite or stellar growth (Reiter 2004).

Thus, in order to improve on the drawbacks, the model was redefined basing the growth behavior on 2 fundamental parameters which opens a wide range of possible growth patterns similar to the ones seen for natural snow crystals.

The method derived utilizes a hexagonal arrangement of cells and each cell contains a value. The value is used to represent the measured amount of water present in the cellular location. Values over the coefficient of 1 are considered to be ice whereas lower values are taken to be water in a form that may move to neighboring cells.

The Algorithm used can be explained in two stages.
Each stage may be computed by using a nearest neighbour hexagonal real-valued cellular automaton, and hence the algorithm we use could be viewed as a one-stage automaton on neighbourhoods containing two levels (Reiter 2004).
The values of the cells are given by the values at the receptive sites plus a constant ad added to it the value of a diffusion term.

The diffusion term is a local average of a modified cellular field obtained by setting the receptive sites to zero (Reiter 2004).

The average is discussed further in the paper.

Cells with the value 0 in the array of unreceptive sites are used as values in the averaging while the zeros in the array of the receptive sites are place holders. The reason for using the above described method is that receptive sites are seen as permanently storing any mass that is derived at that point.

In the unreceptive cells the mass is free to move and thus moves toward an average value. In the end the defined constant added to the receptive sites is used to not only represent a simple generalisation of the model but also allows for the idea that water may be available from outside the hexagonal plane of growth.

The constant parameter to be used for the receptive sites is one of the parameters that can be and will be varied. The second parameter that can be varied and used is the background level of the hexagonal plane. The snowflake will begin to grow form a point of origin which will be a single hexagonal cell with a defined value equal to one in a seat of background where the snowflake can grow.

We will usually denote the added constant by c and the background level by b.
The boundary conditions are taken to be fixed at the background level at a fixed (Euclidean) distance from the initial
cell. (Reiter 2004).
The boundary conditions are aimed to attempt to be as isotropic as possible. One problem is that the different boundary points have a different number of hexagonal steps from the origin so the data flow would take a different amount of time to flow from the centre to the different boundary points. Number of average values may be utilised on the unreceptive cells. The one utilised in this case is the weight for the centre cell is ½ and for the neighbouring cells is 1/12.
The diffusion of the vapor field of the unreceptive sites at a position P and time t, denoted u(t,P), is expected to
satisfy the diffusion equation. (Reiter 2004).

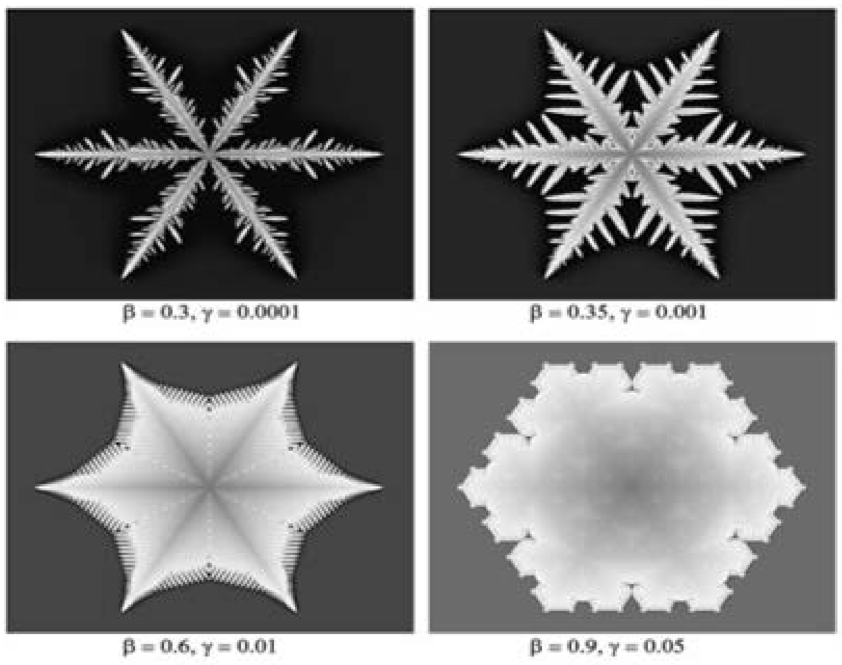
$$\nabla^2 u \approx \frac{2}{3}\left(-6u(t,P) + \sum_{N\in nn(P)} u(t,N)\right)$$

where nn(P) denotes the set of nearest neighbours of P. Thus, solutions may be approximated on a hexagonal lattice by. (Reiter 2004).

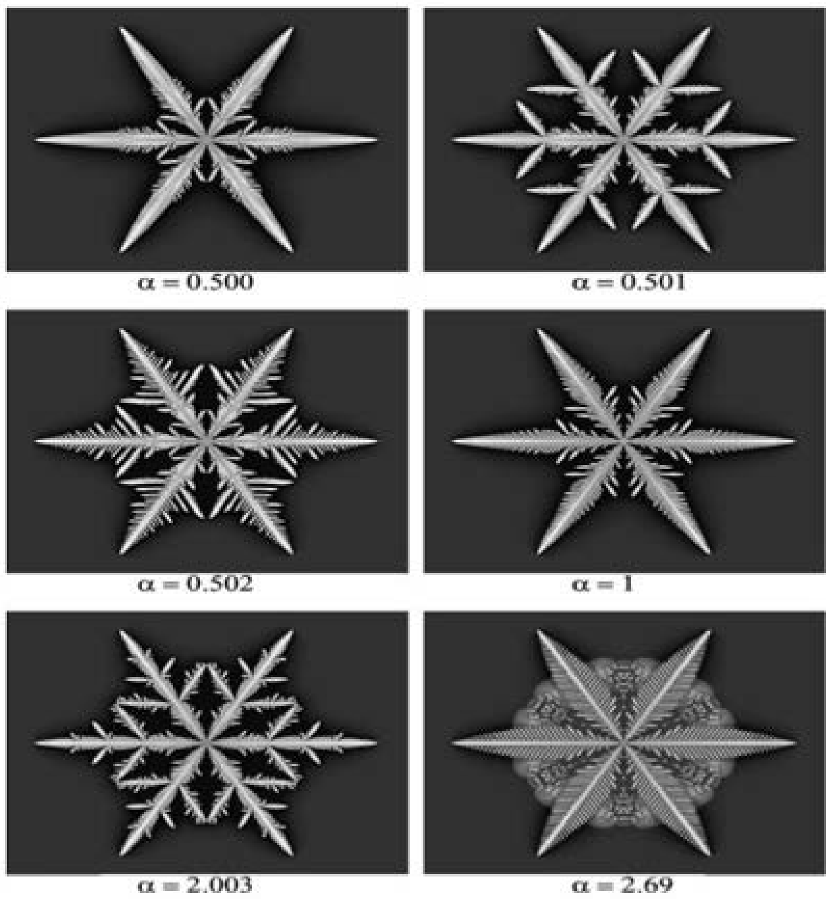
$$u(t+1,P) \approx u(t,P) + \frac{\alpha}{12}\left(-6u(t,P) + \sum_{N\in nn(P)} u(t,N)\right)$$

where we have reorganized constants via a = 8a so that our model with centre weight 1/2 and weight 1/12 for neighbours corresponds to taking a = 1. (Reiter 2004).

The real growth of the crystal structure is highly dependent on the value of Alpha. The variables each in turn can create variations in the snowflakes however drastic changes are seen when Alpha is changed or the Randomness variable is changed significantly.



$\beta = 0.3, \gamma = 0.0001$
$\beta = 0.35, \gamma = 0.001$
$\beta = 0.6, \gamma = 0.01$
$\beta = 0.9, \gamma = 0.05$



The variables of the diffusion constant for Beta =0.4 and Gamma =0.0001 It demonstrates that for Alpha = 0.5,0.501,0.502 give very different crystallised structures. One thing they all share is that they all exhibit dendrite growth.

The examples display the standard model where Alpha = 1 which appears to be very similar to the Alpha =0.5. An intriguing construction occurs when Alpha =2.003

When developing the procedural snowflake with the aide of 2 sources the paper of Reiter from 2004 and a masterclass which works on this system it was emphasised that randomisation is crucial so as to not get the perfect crystal structures like in the mathematical paper.

5.6. Snowflake Generator

Once the snowflake algorithm was proven to be successful an algorithm for generating variations of particles was needed in order to automate the generation process of the particles.

The overall result was not as intended as the gradual increase in randomness proved problematic. Small changes in randomness showed exponential increase in both render time and memory usage. The final system is useful for creating variations of pre-defined snowflakes variables.

Overall the procedural snowflake is perfect for creating snowflakes with significant changes and experimentations with designs whereas the generator can be used to afterwards generate snowflakes with small variations of the result from the procedural snowflake algorithm.

It utilizes 3 algorithms in order to change the variables dynamically and create variations of the initial snowflake. The simulation is designed to generate 10 variations of a snowflake however the equation might need to be edited for each and every case. The equations listed were utilized as they proved to be relatively stable.

$$\alpha = abs * (rand(0.8 + \frac{(\$FF * (8 * 10^{-4})))}{2}$$

$$\beta = 0.2 * abs((rand \frac{((0.8 + (\$FF * (8 * 10^{-4})))}{8}$$

$$\Omega = 0.2 * abs(\left(rand \frac{((\$FF * (4 * 10^{-5}))}{16} \right))$$

Ω = Background Randomness

5.7. Snowfall Single Snowflake System

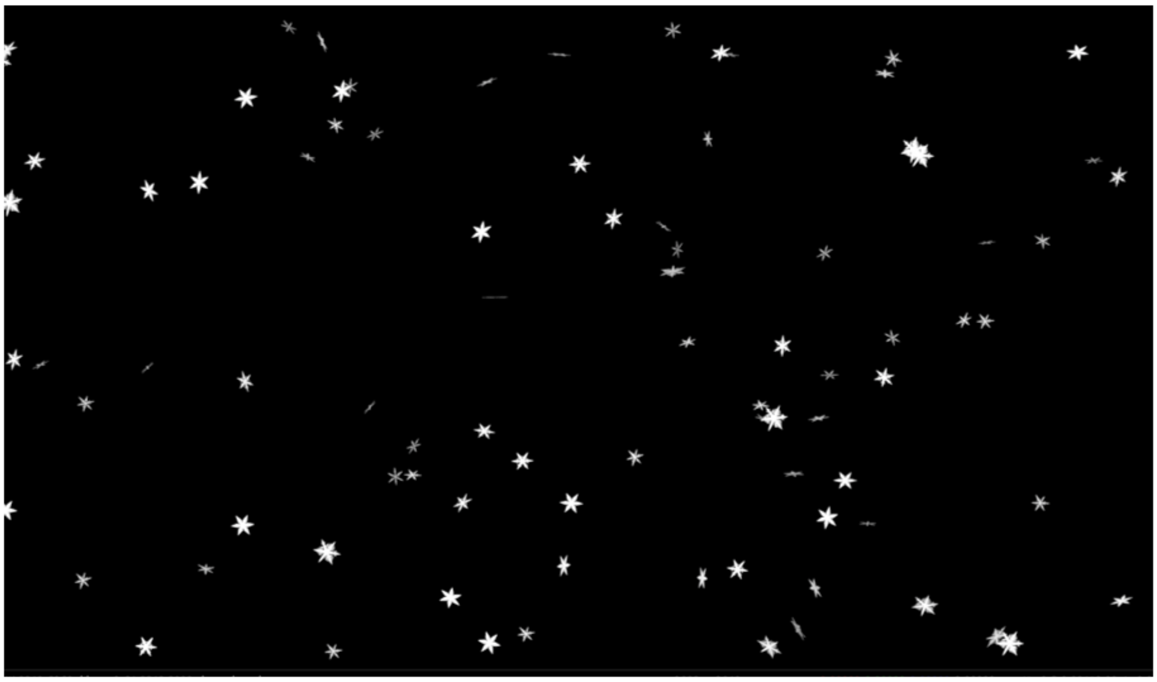
In order to apply the created snowflakes in a realistic scenario the snowfall tool was created. In this tool a particle system which simulates the movement of snowflakes has a procedural snowflake instanced on each and every point creating a more realistic feel and a higher level of detail in the final simulation. The simulation utilizes a Snow Behavior DOP Digital Asset created to encapsulate all of the required settings for Transforming a regular particle simulation into a falling snow particle simulation.

The tool is useful for far way shots which would still give a great amount of detail the inclusion of the snowflakes creates a more realistic feel to the falling snow and depending on the snowflake used can enhance far away shots.

5.8. Snowfall Multiple Snowflake System

A heavier version of the previous system instances multiple snowflakes onto the points at random. It utilizes a slightly modified version of the Snow Behavior Digital Asset called the Snowflake Selector Digital Asset which includes an extra module which allows the user to use multiple snowflakes. This makes the tool ideal for close up shots where the snowflakes would be visible falling near the characters.

It currently uses 6 different snowflakes however the variable can be altered depending on the users needs. One point of caution is to either utilize multiple light snowflakes or 2 or three heavier more detailed snowflakes otherwise the simulation might run out of memory.



6. Conclusion

In conclusion all of the tools work as predicted:

The snow deployment algorithm- was a fairly straightforward implementation up to the painting toolset however was ultimately solved.

Snow Volume Estimation- Works fairly effective although in higher resolution meshes it can experience a slowing down. It can be improved by experimenting with caching data in order to optimize the calculation.

Snow Shader- is unfortunately the most ineffective part of the pipeline as in order to create realistic subsurface scattering and volume the shaders need to overlap and superimpose creating the effective of volume which can prove problematic with simulation using more than 5-6 million particles as they would grow by a magnitude of 2n.

procedural snowflake- was perhaps one of the trickiest parts to implement simply due to the difficulty of translating mathematics into custom solvers however with good references from the masterclass the development was successful after roughly a week and was modified continuously in order to fit the new purpose to be fed into the particle system.

snowflake generator-originally the snowflake generator was intended to streamline the snowflake production process however the variable growth proved highly problematic as small jumps in the variables caused exponential growth both in render time and memory used causing the simulation to crash repeatedly. The equations listed above proved to be much more stable however may still cause crashes' due to the growth in files can quickly become exponential.

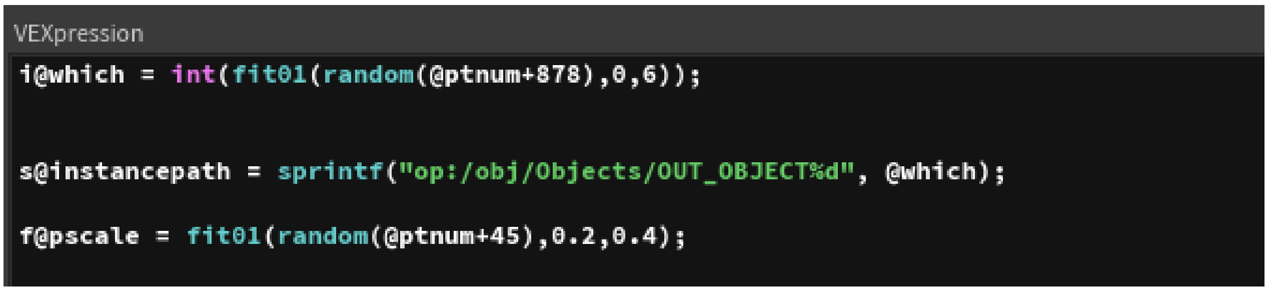
snowfall single snowflake system-The development proved to be very straightforward as the snowflakes were simply mapped onto the particle behavior at the point of generation through the use of copy to points nodes. A basic white marble shader was used to give a shiny white look to the snowflakes without taking up too much resources with a heavy shader.

snowfall multiple snowflake system- This proved to be a very problematic system to implement. Originally the system was attempting to map different snowflakes onto the particles at the point of generation however the nodes instead changed the snowflake mapped onto all particles and the effect turned into transforming snowflakes.

The second attempt was utilizing a switch node which send a different snowflake every frame to a copy stamp node and created a custom variable which linked the snowflakes to points through the inbuild stamp function. This function caused every snowflake instead of being instanced to be loaded onto the memory quickly running the computer out of memory. This method was initially tested with basic geometry however proved ineffective with heavier geometry and was scrapped as memory inefficient.

From analyzing the memory usage, it can be determined that using a combination of the switch node with a stamp copy node can be used for lightweight sprites but not the heavy snowflake alembic caches.

The final solution was to modify and implement a script.



VEXpression

```
i@which = int(fit01(random(@ptnum+878),0,6));


s@instancepath = sprintf("op:/obj/Objects/OUT_OBJECT%d", @which);

f@pscale = fit01(random(@ptnum+45),0.2,0.4);
```

In conclusion all the tools implemented appear to be working adequately and would be able to aid an artist in the development of snow FX.

7. References

Reiter, C. (2004). *A local cellular model for snow crystal growth*. 1st ed. [ebook] Department of Mathematics, Lafayette College, Easton, PA 18042, USA. Available at: http://www.patarnott.com/pdf/SnowCrystalGrowth.pdf [Accessed 15 Aug. 2018].

Bartels, P. (2018). *Position Based Dynamics*. [online] Nccastaff.bournemouth.ac.uk. Available at: https://nccastaff.bournemouth.ac.uk/jmacey/MastersProjects/MSc15/03Pieterjan/thesis.pdf [Accessed 16 Aug. 2018].

Chu, C. (2018). *very beginner houdini dry/wet sand tutorial*. [online] YouTube. Available at: https://www.youtube.com/watch?v=2gF0jNp1INA [Accessed 16 Aug. 2018].

Jong, E. (2018). *Simulating Snow with the Material Point Method*. [online] Nccastaff.bmth.ac.uk. Available at: https://nccastaff.bmth.ac.uk/jmacey/MastersProjects/MSc15/05Esther/thesisEMdeJong.pdf [Accessed 16 Aug. 2018].

Orell, A. (2018). *Position Based Dynamics for Character Effects*. [online] Nccastaff.bournemouth.ac.uk. Available at: https://nccastaff.bournemouth.ac.uk/jmacey/MastersProjects/MSc16/12/thesis.pdf [Accessed 16 Aug. 2018].

Sørensen, I. (2018). *Simulating Melting with the Material Point Method*. [online] Nccastaff.bournemouth.ac.uk. Available at: https://nccastaff.bournemouth.ac.uk/jmacey/MastersProjects/MSc16/14/Thesis.pdf [Accessed 16 Aug. 2018].

Stomakhin, A. and Schroeder, C. (2018). [online] Math.ucla.edu. Available at: https://www.math.ucla.edu/~jteran/papers/SSCTS13.pdf [Accessed 16 Aug. 2018].

Vimeo. (2018). *A material point method for snow simulation*. [online] Available at: https://vimeo.com/160322962 [Accessed 16 Aug. 2018].

Vimeo. (2018). *Creating A Procedural Snowflake*. [online] Available at: https://vimeo.com/245585252 [Accessed 16 Aug. 2018].

Vimeo. (2018). *Instant Snow with Houdini*. [online] Available at: https://vimeo.com/151809783 [Accessed 16 Aug. 2018].

YouTube. (2018). *Disney's Frozen A Material Point Method For Snow Simulation*. [online] Available at: https://www.youtube.com/watch?v=O0kyDKu8K-k [Accessed 16 Aug. 2018].

YouTube. (2018). *Disney's Frozen A Material Point Method For Snow Simulation*. [online] Available at: https://www.youtube.com/watch?v=O0kyDKu8K-k [Accessed 16 Aug. 2018].

YouTube. (2018). *Houdini | Wet Sand*. [online] Available at: https://www.youtube.com/watch?v=nXI_ew-mtP4 [Accessed 16 Aug. 2018].

YouTube. (2018). *Houdini Snow and Ice 1*. [online] Available at: https://www.youtube.com/watch?v=yELy9PkO5uI [Accessed 16 Aug. 2018].

YouTube. (2018). *Houdini Snow and Ice 2*. [online] Available at: https://www.youtube.com/watch?v=aypGKn_gupI&t=1s [Accessed 16 Aug. 2018].

Dalvi, R. (2018). *Introduction to POP grains*. [online] Vimeo. Available at: https://vimeo.com/132847114 [Accessed 16 Aug. 2018].

Lait, J. (2018). *H15 Masterclass | Grains*. [online] Vimeo. Available at: https://vimeo.com/142534638 [Accessed 16 Aug. 2018].

Sidefx.com. (2018). *Forums | SideFX*. [online] Available at: https://www.sidefx.com/forum/ [Accessed 16 Aug. 2018].

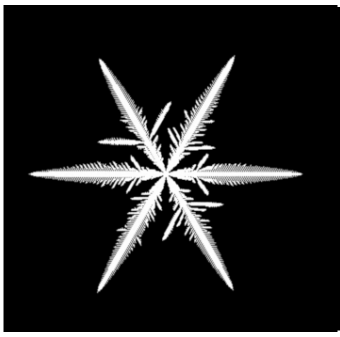
Vimeo. (2018). *DIY Scatter Tool (Solving Problems in Houdini)*. [online] Available at: https://vimeo.com/198658562 [Accessed 16 Aug. 2018].

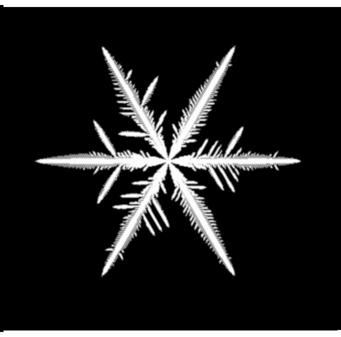
Müller, M. and Heidelberger, B. (2007). *Position Based Dynamics*. [online] Matthias-mueller-fischer.ch. Available at: http://matthias-mueller-fischer.ch/publications/posBasedDyn.pdf [Accessed 16 Aug. 2018].

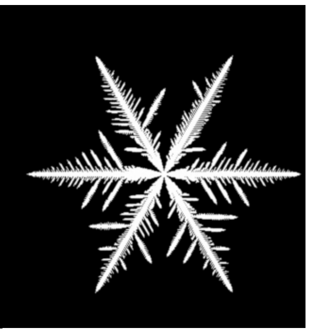
Papadimitriou, A. (2012). *Sand Simulation*. [online] Nccastaff.bournemouth.ac.uk. Available at: https://nccastaff.bournemouth.ac.uk/jmacey/ [Accessed 18 Aug. 2018].

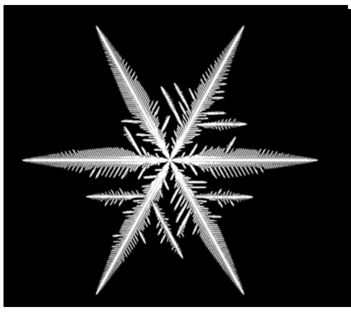
8. Appendices

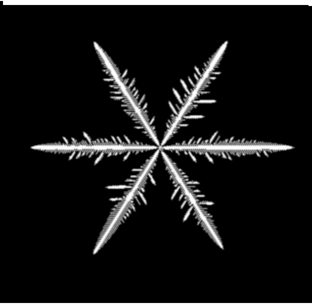
From the Procedural Snowflake generator here are a few examples of how the snowflakes crystalized there are differences in the branching thanks to a higher randomization value than in the original paper where the idea was to produce nearly perfect procedural snowflakes. Here the randomization variables allow for more variation at the price of more calculations.

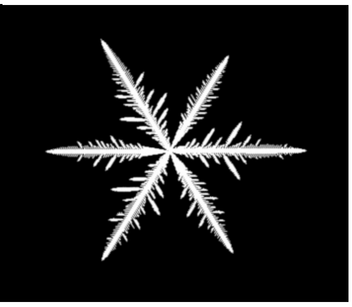

$\alpha=0.5$ $\beta=0.4$

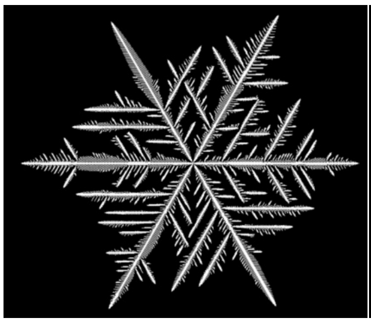

$\alpha=0.501$ $\beta=0.4$

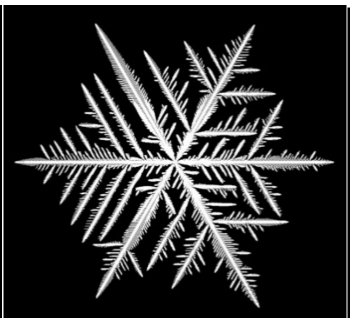

$\alpha=0.502$ $\beta=0.4$

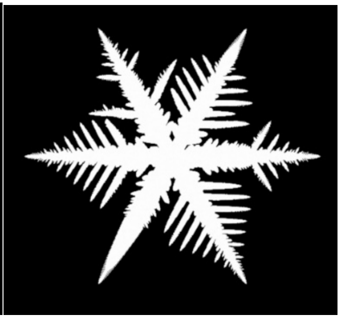

$\alpha= 1$ $\beta= 0.4$


$\alpha= 1$ $\beta= 0.3$ $\mu=0.0$


$\alpha= 1$ $\beta= 0.3$ $\mu=0.0001$


$\alpha= 1$ $\beta= 0.4$ $\mu=0.0$


$\alpha= 1$ $\beta= 0.3$ $\mu=0.0001$


$\alpha= 1$ $\beta= 0.35$ $\mu=0.001$