Diffusion Limited Aggregation Inside 3D Models

August 19, 2017

Masters Project

Jesin Roy

s4923830

MSc Computer Animation and Visual Effects

Bournemouth University

Acknowledgment

I would like to thank Jon Macey, Phil Spicer, Michail Agoulas and Yannis Ioannidis for their advice and support especially through the difficult states of the project.
Also I would like to express my special thanks to my course mates in MSc CAVE, my family and friends without whose constant support and guidance this project would not have been possible.

Contents

List of Figures

List of Algorithms

Abstract

The primary objective of this work is to explore the phenomenon of Diffusion Limited Aggregation inside 3D models using Houdini. A brief discussion on DLA and the approach used for the implementation of same in Houdini will be examined. Issues intrinsic to the process will be addressed along with the solutions employed. As a final wrap up, few of the many possible application of the designed system will be touched upon.

1 Introduction

Randomness plays an important role in Computer Graphics especially when it comes to creating natural artifacts. Adhering to the fact that nature is governed by random and bizarre order, for creating anything natural in graphics means to be able to resemble that order. This could at its closest be achieved using fractal patterns. Fractal patterns offer computer artists a way of producing controlled randomness which can be used for a variety of special effects.

One of the popular methods of creating fractal patters is L-Systems. Their ordered yet apparent randomness has been significant for modeling natural looking shapes like trees, plants, fracture and crack propagation to name a few.

Though L- Systems are a powerful way of fractal generation, they encounter limitations when it comes to growth in an environment that is liable to change. Under such conditions the best approach is Diffusion Limited Aggregation, DLA. DLA is a model that is used for various scientific processes and is capable of producing fractal patterns. It is in many ways similar to L - Systems but topped with the advantage of being able to adapt to its surroundings in addition to yielding stochastic fractal shapes.

DLA has been widely used to perform simulation on flat 2D planes as well as on 3D surfaces. The project aims at producing a tool capable of generating fractal patterns inside 3D models. The tool presented here can provide a simple (although primitive) technique for artists to implement DLA in Houdini. The system designed is fully procedural and is adaptable to any geometrical shape. This can then be used to create effects like vine creeper, injection or growth proliferation.

2 Previous Work

There are two primary growth algorithms in computer graphics especially when it comes to creating fractal like growth patterns: L - Systems and Diffusion Limited Aggregation.

2.1 L-Systems

L – Systems or Lindenmayer Systems use a mathematical language in which an initial string of characters is matched against rules which are evaluated repeatedly and the results are used to generate geometry. The result of each evaluation becomes the basis for the next iteration of geometry, giving illustration of growth.

An L System involves three main components:

- **Alphabet:** Valid characters that can be included
- **Axiom:** A sentence that describes the initial state of the system
- **Rules:** Rules are applied to the axiom and then applied recursively, generation new sentences over and over again

L-Systems get its geometrical representation from Turtle Graphics. This places each alphabet with a specific movement which when used as a sentence generates a command describing the sequence of actions to be undertaken by the turtle: moving forwards a given distance, turning through a given distance. Ergen

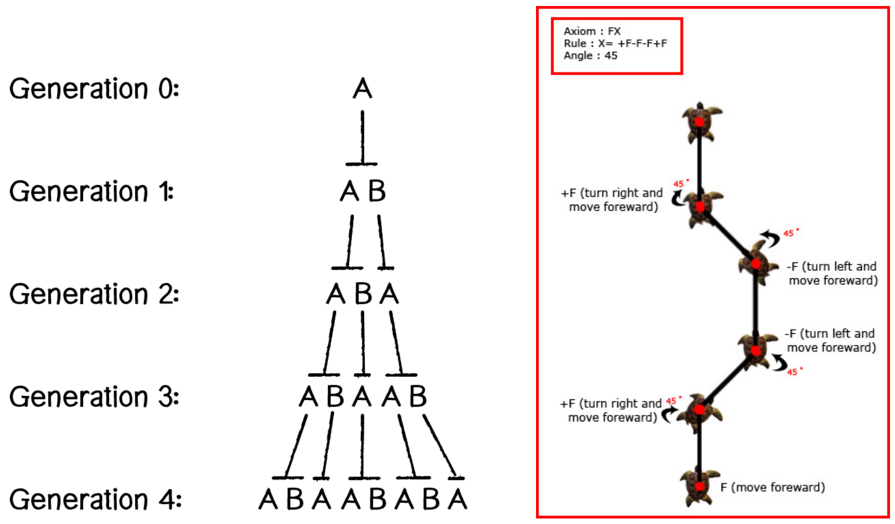

Figure 1: L-System generations and Turtle Graphics

L-System has diverse applications because of its ability to create natural fractal like forms. Though based on simple rules to begin with, the recursive nature of an L-System, leading to self similarity makes it easier to define complex plants models, algae forms, patterns like Koch curve, Cantor set, Sierpinski triangle and many more. At each iterative step, the working set is replaced with the grammar in parallel giving an adaptive way to create a variety of plants with subtle variations enabling them to have a realistic look. Apart from graphics, L-systems have useful mathematical applications. Taking an example, given by Smith (2013)using l-systems it can be shown that 2D or 3D space could be filled with a 1D object; called the space filling curves and Hilbert Curve respectively.

2.2 Diffusion Limited Aggregation

This method was introduced by Witten and Sander (1983), who first experimented with DLA as a model for irreversible colloidal aggregation. The process started at the grid center where a particle was placed. Eventually a diffusing particle was introduced in the scene which was allowed to wander freely until it was within a fixed distance of the center where it attaches to the central particle. After the first particle is attached, another particle is released and so on, all of which undergo the same process. Particles forming the structure undergo a Brownian motion (random walk) before attaching themselves (Aggregating) to the structure. The particles are considered to be in low concentration so that they don't come into contact with each other and the structure grows one particle at a time rather that by chunks of particles.

The initial aggregate need not necessarily be a particle; it could be an edge or a surface as well. The course of action for all of them will be same. With the progress of time as the aggregate particles multiply and grow, they produce complex structures with high levels of detail.

As Halsey (2000) mentions, many of the patterns we see in nature are the result of a host of processes involving an interaction between the transport and thermodynamic properties of matter and radiation involved. Usually transport is dominated by convection but in cases where instabilities in the pattern generated is high, diffusion takes over like formation of river deltas, frost on glass, mineral deposits and many other natural phenomenon. Since the process takes place at a per particle basis with high unpredictability of the attachment point owing to the Brownian motion, the resultant pattern is varied and random. Yet the association of one particle with other to form dendrite branches spread out around the initial seed portrays a masked order. This order in chaos found DLA its place in computer graphics. Cashmore (2009) mentions applications of DLA in other arenas of science like exploration of oil and crystal formation. It is also used as a catalyst for certain chemical reactions, as a way to vary the rate of reaction without altering the end product. “Catalysts need a large surface area to be most effective and often are arranged as a largely branched structure with many holes. This shape is akin to structures which are simulated by DLA” Cashmore (2009)

3 Technical Background

Though both DLA and L-Systems are based on the concept of incremental growth, what differentiates DLA from L-Systems is the versatility and quality of the results DLA offers with the potential of generating dendrite forms as seen in a number of natural objects including lichens, crystals, neurons and lightening.

There are few examples of DLA being implemented on or inside 3D surfaces. One very significant for this project is by Cashmore (2009). He develops a tool for generation of DLA on the 3D surface and uses it to create ivy like growth structure as well as applies texture map to create lava effect, rust propagation and displacement effect. He uses the concept of SDF volume in Houdini to create a thin layer over the geometry to facilitate the movement of particles and to prevent them penetrating into or escaping from the geometry. The user is given the control to choose the initial area of growth via a paint tool. The aggregate points generated are then connected using geometry and used to create vines or veins, shader and displacement effect.

Bourke (2006) tests the idea of constraining DLA growth by a surface or solid described by an STL file format which describes the surface as a collection of triangles. During simulation if the growing structure intersects a triangle, either the new particle is not added to the branch which then continues random walk or the intersection with the triangle is calculated and a branch is added that just touches the constraint surface. A sphere is used as the particle emitter. Particles are introduced into the simulation randomly on the surface of a sphere to get an even distribution at all locations especially the places with small gaps where it might be unlikely for the particles to penetrate. He also experiments with the multiple independent seeds for the not so easy to reach areas in the geometry.

One of the most prominent works in DLA is by Lomas (2005) who created the aggregate structure by simulating the paths of millions of particles flowing in a field of force thus creating structures of intricate design and immense complexity. Apart from just the Brownian motion, the particles are influenced by the flow field. "Forms are created by a process of indirect design: the shape of forms is governed by controlling initial conditions and rules for deposition rather than by designing the final form directly." Lomas (2005)

Stock (2003)experiments with DLA in 3D without any external constraint but introduces two new factors of influence; stickiness and bulk velocity. Stickiness determines the probability of a particle to attach to the aggregate structure. When a particle approaches another particle within a certain distance, it is given a chance to stick. If it fails to stick, it is moved a tiny distance away from that particle so that the diffusion distance for the next time step remains positive. It is observed that, lower the stickiness factor, denser the aggregate. For the bulk velocity, in Stock's own words, "The first task for each random-walk-accelerated step is to determine the nearest element in the frozen structure. The distance to that particle determines the radius of the sphere inside which the active particle must stay. With no bulk velocity, the set of candidate positions is the surface of that sphere. With a non-zero bulk velocity, the set of candidate positions is a smaller sphere tangent to the larger sphere. The radius of the new sphere is calculated by solving for the mean diffusion time that would be capable of convecting and diffusing the particle a distance equal to the distance to the nearest particle. Thus, one random-walk-accelerated step is composed of a bulk motion step, and a spherical diffusion step, instead of just a spherical diffusion step." Stock (2003). It is observed that, lower the bulk velocity lower is the density.

3.1 Mathematical Aspect

The measure of how complex a self similar object is given by Fractal Dimension (D). Approximately said, it measures the number of points lying in a given set. Fractal Dimension connects the number of particles with size of the cluster.

$n = r^D$

or

$\mathrm{D} = \log(\mathrm{n}) / \log(\mathrm{r})$

- n = Number of particles
- r = size of the cluster

for a self similar object:

- n = number of self similar pieces
- r = magnification factor

for fractals in two dimensions, D ≈ 1.71 and in three dimensions, D ≈ 2.5.

Halsey (2000), states the observation reached through numerical simulations, that in high numbers of spacial dimensions d, the fractal dimension of the cluster D → d-1.

"In addition, the fractal dimension of DLA appears to depend weakly on the geometry of the simulation. The result D ≈ 1.71 is obtained for radial growth from a seed. However, for growth from a surface or in a channel, one obtains a result closer to D ≈ 1.67, a small but robust difference from the radial growth ... that seems to persist to the asymptotic growth limit." Halsey (2000)
The original method of calculating fractal dimension is by computing density correlation function.

$$C_T(r) =< \rho(r')\rho(r'+r) > \tag{1}$$

Density correlation function for N particle aggregation gives information about particle distribution

$$C(r) = N^{-}\sum_{r'}\rho(r')\rho(r'+r) \tag{2}$$

- $C(r)$ = average density
- r = distance separating the two sites

$C(r)$ can be considered as a measurement of the density in a shell with mass $dM(r)$, radius r, and thickness dr.

$$C(r) = dM(r)/2\pi r dr \tag{3}$$

Amount of Mass of an object inside a circle of radius r has a power law relation:

$$M(r) \propto r^D \tag{4}$$

Substituting equation (4) in equation (3)

$$C(r) \sim dr^D/2\pi r dr \tag{5}$$

$$\Rightarrow C(r) \sim r^{D-1}dr/2\pi r dr \sim r^{D-2} \equiv r^{D-d} \tag{6}$$

For a self similar fractal, the density-density correlation function is expected to scale as:

$$C(r) \sim r^{-\alpha} \tag{7}$$

From (6) and (7), we get:
D = -α + d

- D = Fractal dimension
- d = Euclidean Dimension

4 Implementation

This section talks about the approach taken for implementing DLA in Houdini, problems encountered on way, attempt at various solutions and finally the efficiency of the program in relation to different affecting factors.

4.1 Methodology

The aim of the project is to create fractal like growth inside 3D geometry using the principle of Diffusion Limited Aggregation. The implementation has been done in two parts: Text and 3D Geometry and Intersection. The first part involves working with a single geometry, filling the geometry using growth effect. The second part, however deals with how the growth will perform when two or more objects are in contact or are penetrating into each other.

The process of Diffusion Limited Aggregation consists of the following main parts:

- The initial seed(s) called the "aggregate" which will start the growth process
- A particle called "walker" at a location inside the geometry which will undergo random walk to reach the aggregate
- Collision happens between the walker and aggregate following which the walker stops and becomes part of the aggregate
- Generating new walker which will go through the same process as the previous one
- Repeating the process until the aggregate has grown to the desired size

4.1.1 Defining Initial Aggregate

The first stage is to decide the geometry in which aggregation is to take place. The tool already provides a few default geometries but user has the option to import and apply aggregation to any geometry. Only thing to take care when using an external geometry is to ensure that the "normals" are properly oriented because if not, it will affect particle emission.

If it is Text, the user can type in a word or sentence, set the alignment, size, extrusion limit and the space between the characters. The spacing between the characters is best to be kept at minimum as it might affect the growth propagation if the distance between characters is greater than a certain limit.

Text and 3D Geometry After choosing the geometry for the aggregate formation, the initial seed has to be decided. This could be any of the static point(s) of the geometry say for example, point "0". Now, for any chosen geometry the point "0" will be the initial seed but there could be cases when the user's desired initial seed is not point 0 but some other point or a group of points or points at various different locations of the geometry. Considering all these scenarios, the user is given the flexibility to choose his/ her own initial seed region. Through the *Paint* option in Houdini, the user can paint over any region of the geometry and the points scattered over that region will act as the initial aggregate. The Paint SOP provides a very visual way with a better control to choose the growth region.

Intersection For intersection, the user is given the choice to use his/her own intersecting geometries in addition to the main geometry. Unlike the 3D Geometry and Text, the initial aggregate for intersection is defined by the area of intersection of two or more geometries. So the first step is to retrieve the area of intersection. For retrieving this area, firstly all the geometries are converted to volume using *IsoOffset*. The *Uniform Sampling Divs* is set to 100 to define the fog volume within geometry boundary. The converted volume is an opaque white fog. To view the intersecting part of fog volume, *Volume Visualization* is used which adds detail attributes to the volume to allow for visualization of volumes that require multiple volumes to be joined together. Since we would now be dealing with just the intersecting part, this can be converted back to polygons using *Volume convert*. The particles in the main geometry which are near the particles in the intersection area will act as the initial aggregate.

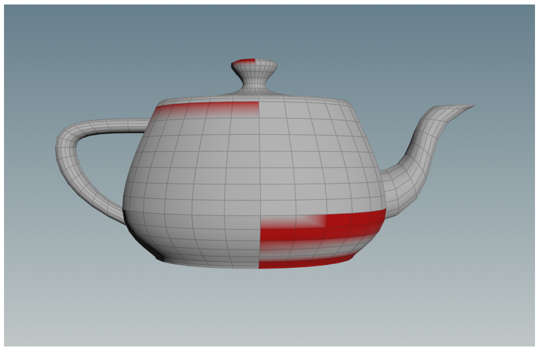

Figure 2: Painted geometry surface

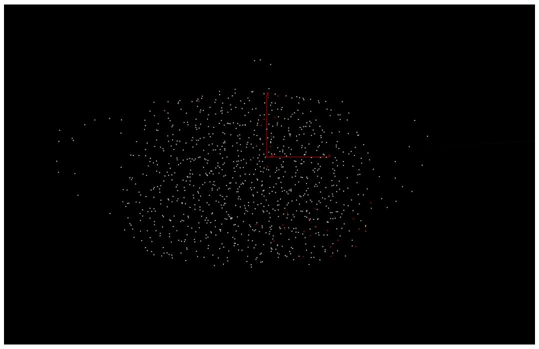

Figure 3: Particles scattered inside the geometry with colored point in the painted area

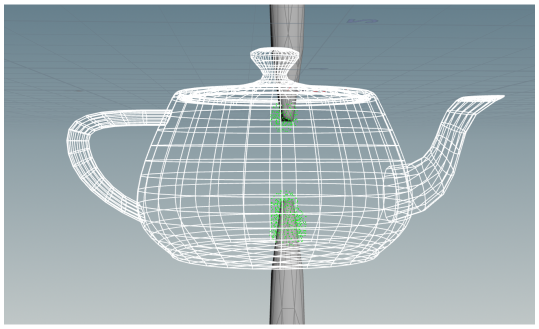

Figure 4: Aggregate points at the junction of intersection

4.1.2 Scattering Internal Particles

After initial aggregate area is defined on the main geometry, the procedure for growth has to be set up. This is done by scattering the particles within the geometry since the growth will take place inside. For this purpose, the geometry is converted into a volume using *IsoOffset* with the *Uniform Sampling Divs* set to 100. The points are then scattered inside the fog volume using *Scatter* node. Once the points are scatted, they need to know the color information from the surface. For Text and 3D Geometry, this is accomplished using *Attribute Transfer* which will transfer the color from the surface to points in the painted region thus defining the region from where the growth will start. In case of Intersection, the color is transferred from the points in the intersection area. The scattered points are then grouped separately based on color as aggregate and walkers. The user is given the control to input the number of scattered points. 3

4.1.3 Generating Random Walkers

Random walkers are particles which will via random movement collide with the aggregate points for the process of growth to begin. The generation of random walkers is done inside the pop network which handles particle simulation. The POP network uses the particles scattered inside the volume as emitters for the random walkers. They are given a starting velocity as follows:

- *fit01 (rand (@ptnum * $FF), 0.1, 1)*

Analyzing the expression, the rand function creates a random number between 0 and 1. It takes the combination of *@ptnum* which is the point number and the floating frame number $FF to ensure varied speed for every particle per frame basis. It ensures that each particle has a random number between 0 and 1. The role of fit01 is to take the range from 0 to 1 and fit it to be between a new range from 0.1 to 1. To further randomize the movement, a unique number is multiplied to the random number of each axis. If every particle has a different speed along each x, y and z axis the motion should be completely random. In this manner, no particular direction will be given preference over the others and thus following the process of creating random walks as described by the traditional DLA.

- *fit01 (rand (@ptnum * $FF * unique_number), 0.1, 1)*

A random walker can be generated at any location inside the geometry but there is a possibility that they could fly out of the geometry during the random walk and in the process moving away from the aggregate. If this happens, either collision detection can be set up to kill the particles as soon as they hit the geometry boundary or a POP Kill could be used with the input geometry as the bounding object. Bounding object ensures elimination of all the particles that step out of the bounded area. Both the options will work equally well removing the particles flowing out of the geometry and giving a random walk contained within the bounds of the geometry. Another option tested was to redirect the particles moving out back into the object. The option was ruled out because, since the emitters are continuously emitting, redirecting the particles would mean packing all the emitted particles since the starting frame inside the geometry which will create a huge cluster of undesired particles.

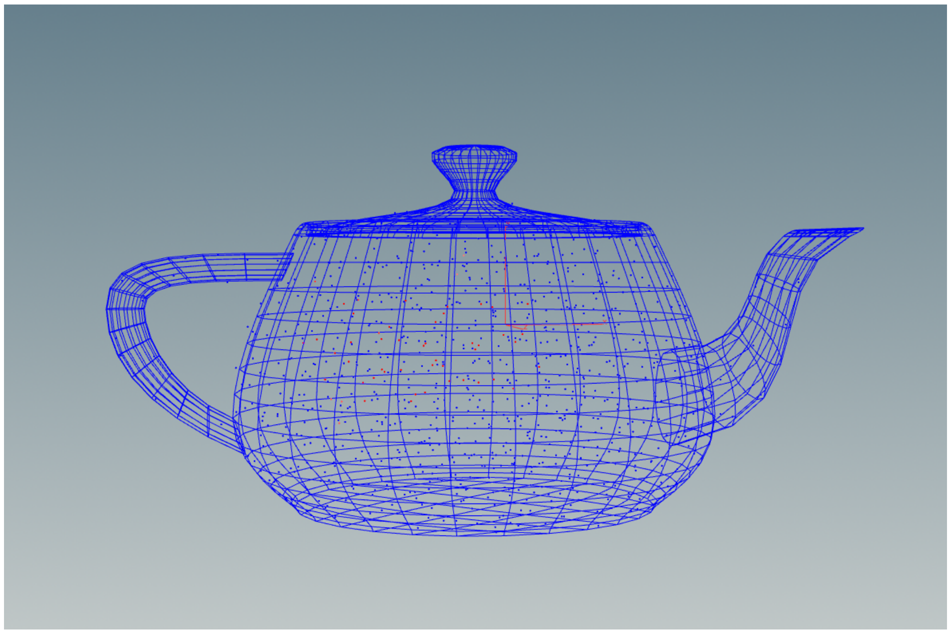

Figure 5: Emitter points scattered inside the geometry

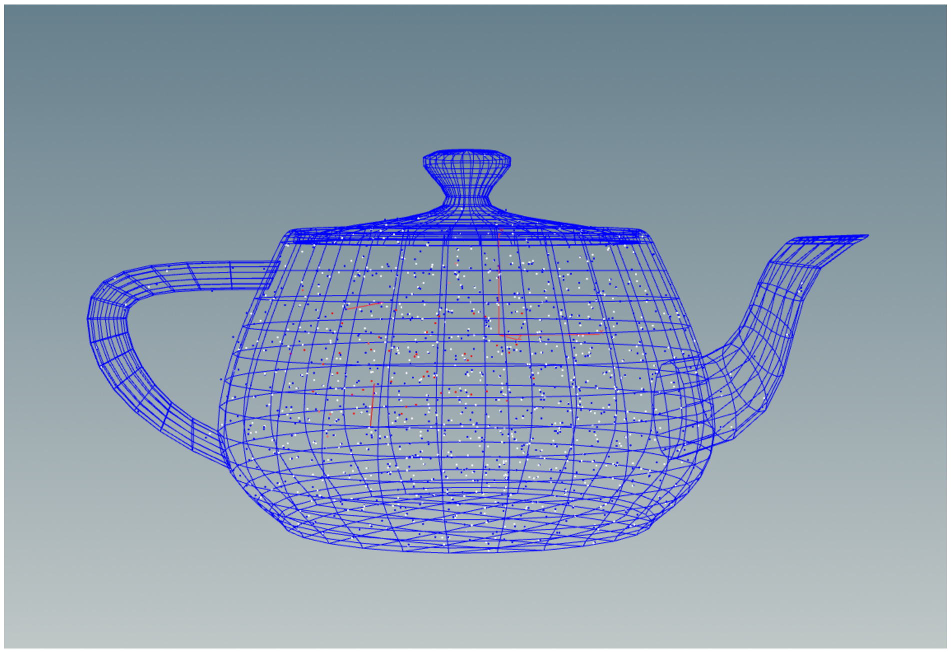

Figure 6: Starting emission

4.1.4 Collision and Growing Aggregate

Now that random walkers are generated, it is to be checked if during their random movement they collided with the initial aggregate or not. The collision can only happen when the walkers are aware of the initial aggregate. Since the initial seed points are at *SOP Level* and the walkers are at *POP*

Level, the walkers cannot interact with the seeds until they are brought down to the *POP Level*. A *sopsolver* is used to bring the *SOP Level* information into *POP Level*. Using *sopsolver* the walkers are introduced to the initial aggregate points at the first frame and those which are within the distance threshold of the aggregate are stopped, grouped separately and added to the initial aggregate to form the new aggregate.

The entire procedure is handled using vex code inside the *sopsolver*. *Pcopen()* function is used to determine the closest point, "nearpt" for each walker. *Pcopen()* opens a point cloud which is used to fetch points of the geometry within a certain distance. When the points are fetched, the *pciterate()* operation is used to iterate through all the retrieved points. Since we are working with point numbers, the point number of each point is imported using *pcimport()*.

Once the "nearpt" is found, it is checked if it is in the aggregate group. If yes, the walker is asked to stop, change color and is grouped under a separate group, "stopped". After the first frame, the walkers are made to interact with the new aggregate, i.e. the initial aggregate point and the newly stopped points and the same process of stopping and adding to the new group continues till all the walkers become part of the aggregate.

Algorithm 1 Walker point collision with the growing aggregate

```
select search distance
find two neighbor points within search distance
loop through neighbor points
    fetch point number of neighbor points
    if neighbor points belongs to aggregate or stopped group
        stop current point
        regroup from walker to stopped
        change color
    end if
end loop
```

In a similar manner just as the walkers are made aware of the growing aggregate, the scattered points which are emitting the walkers need to be alerted of the new aggregate so that when they are within the said threshold, the emission could stop. A feedback system was set up for the same using File sop in Houdini. At the end of each frame, the points which haven't collided are deleted and the stopped points are written to the file. 6 Meanwhile another File sop is used to read in the previous frame of aggregate particles. These are projected as a set of points in space which the *solver sop* uses to check the distance between the emitters and their respective nearest aggregate points. The nearest point is calculated using the *nearpoint()* function. Once the nearest point is found, the *point()* function is used to get the position of the nearest point. We already know the position of the emitter and now with the position of the nearest point we can calculate the distance between them and if it is within the threshold, the emitter could be deleted so that emission from that emitter could stop.

Algorithm 2 Stopping emission based on new aggregate

```
select search distance
find nearest aggregate or stopped point
calculate nearest point position
calculate distance between emitter and resp. nearest point
if distance < search distance
    remove from emitter group
end if
```

4.1.5 Joining Aggregate Points

At the end of the simulation we will have a *.bgeo* file with thousands of points. These points could be used as it is to create a particle effect or they could be used for some propagation like effect if joined together using lines. The joining can be easily done since we know which point was stopped by which.

As mentioned earlier, inside the POP network the "nearpt" of each walker is found and if that point is part of the aggregate or stopped group the walker is stopped. At the time when the walker is stopped, a connection is established between the walker and its "nearpt". As the point number of both the points is known at this instance, they can be joined using a polyline by adding vertex to both using *addvertex()* function and setting connection using *addprim()*. In this manner, as each walker is stopped it connects to the point which stopped it using a polyline giving a propagation like growth effect. 4

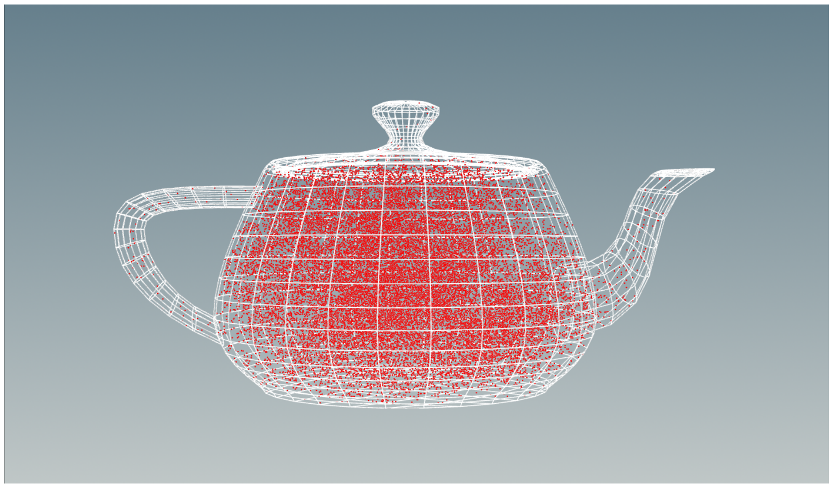

Figure 7: Geometry covered with aggregate points

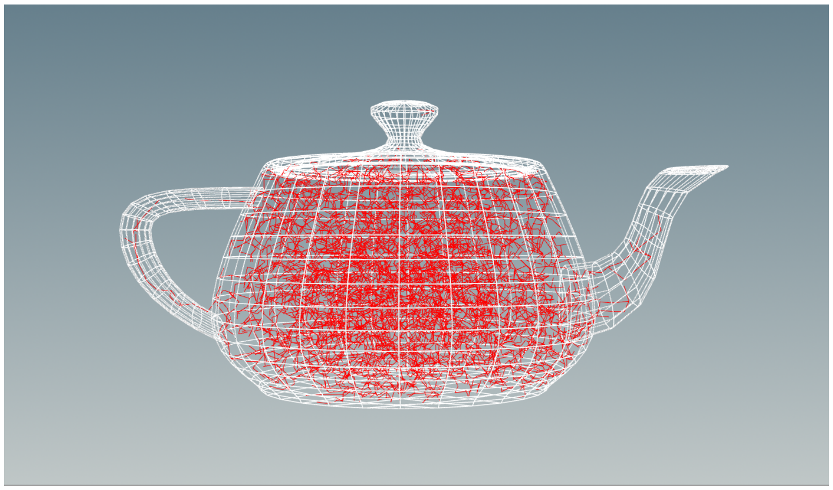

Figure 8: Connected aggregate points

4.1.6 Adding Geometry

Now that we have both points and lines, these can be used for carrying geometry which could then be used for the application of different shaders or textures or simply colors to add aesthetic appeal.

Text and 3D Geometry

Wire Fill The lines are converted to geometry using the polywire sop which constructs polygonal tubes around polylines with smooth bends and intersections and creating them into renderable geometry. The tool provides the mechanism to use polywire in two ways: one is simply replacing the lines with polywire giving user the control to change the width. The second is to have polywire with varying width. For varying width, the concept of having thicker branches at the geometry center and finer branches near the boundary is applied. The distance of each point from the geometry boundary is measured using *xyzdist()* function. This distance is then used to drive the polywire width giving a linear decrease as the points reach the geometry boundary. 7

Since this effect could also be used to grow hedge or foliage, specific points for appending additional geometries for flowers and leaves would be required. For this purpose few points are selected as seeds for growth. These seeds are selected based on two conditions: their nearness to the geometry boundary and the number of neighbors. A small distance from the geometry boundary is considered so as to keep the flowers or leaves well in sight and away from the central clustering. All the points within this distance will be considered as potential seeds but to actually be a seed they have to fulfill the second criteria as well i.e. the neighbor count. Growth can actually happen at any location in the foliage but here we account for growth only at the end points and number of neighbors of a point is necessary as it will tell us whether or not it is the last point of the branch. 10

Finally, geometry added on seed points need to look like they are part of the branch i.e their alignment should be in the direction of the branch. For the appropriate geometry arrangement, direction of the normals has to be fixed. A proper connection should ideally have the normals pointing in the direction of the branch. This could be achieved by taking vector difference between a point's position and the position of the point it will be connecting to. The difference then could be set as the direction of normal. This way, all the normals will be pointing in the direction of the edge with the normals for the extreme end points pointing directly outwards instead of being at some arbitrary angle.

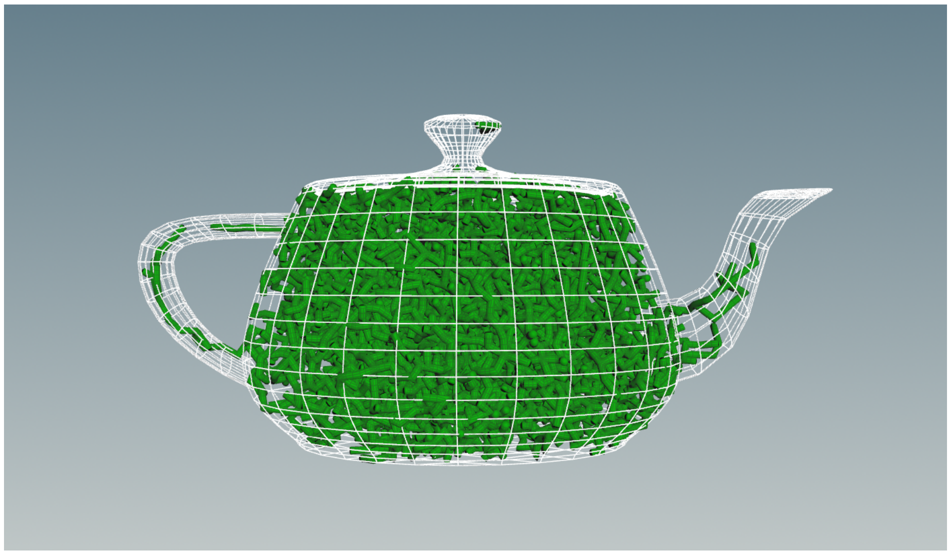

Figure 9: Wire Fill: Uniform width

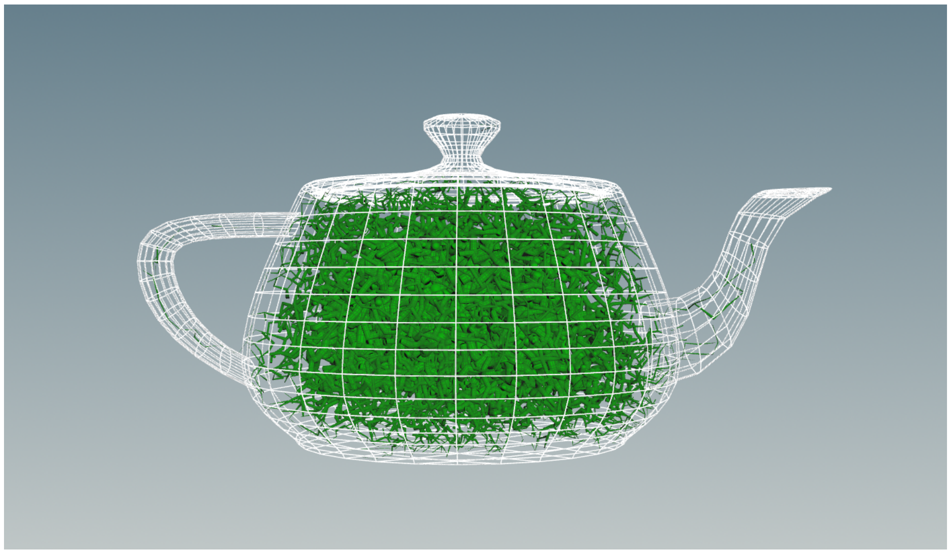

Figure 10: Wire Fill: Varying width with distance from boundary

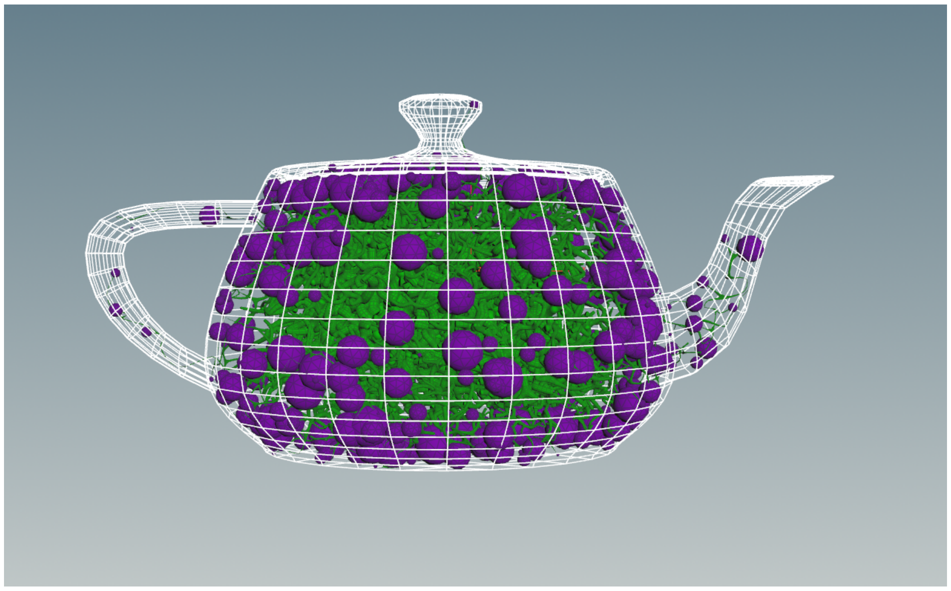

Figure 11: Wire Fill: Geometry added on seed points

Object Fill Similar to lines, the points could be replaced with geometry giving an abstract psychedelic growth effect or to have a structure created with different shapes.

Like in case of selecting main geometry, the tool provides few default shapes to select from but the user is given the option to import and use whatever object he/ she wants instead. The selected shape will replace the aggregate points. Again, to make the effect more visually interesting, the choice of scale variation is provided. The distance of each point from the boundary is calculated and *Copy Stamp* is used to allot the distance to the *Uniform Scale* attribute of the object being copied. The copied object near the boundary of the geometry will be smaller in size compared to the ones towards the center. 7

The above cases work fine as long as finer details like geometry overlap is ignored. But there could be scenarios when interpenetration of geometries might not be a desirable effect and for the same reason, an additional control for removing intersections is provided. For each point, the distance from the nearest point is calculated and the half of this distance is used as scale of the geometry being copied on that point. Only half the distance is used because, if the geometry being copied has any protrusions, there is sufficient space to accommodate it well. 9

The other aspect considered for the shape fill is color. Similar to scale, color has been given a variation based on the distance from the boundary. The change in color with distance gives a shading effect with the darker shades towards the boundary and lighter shades towards the center. This can again be randomized with a *rand()* function based on the point number *@ptnum*, frame *$F* or anything user wishes to set.

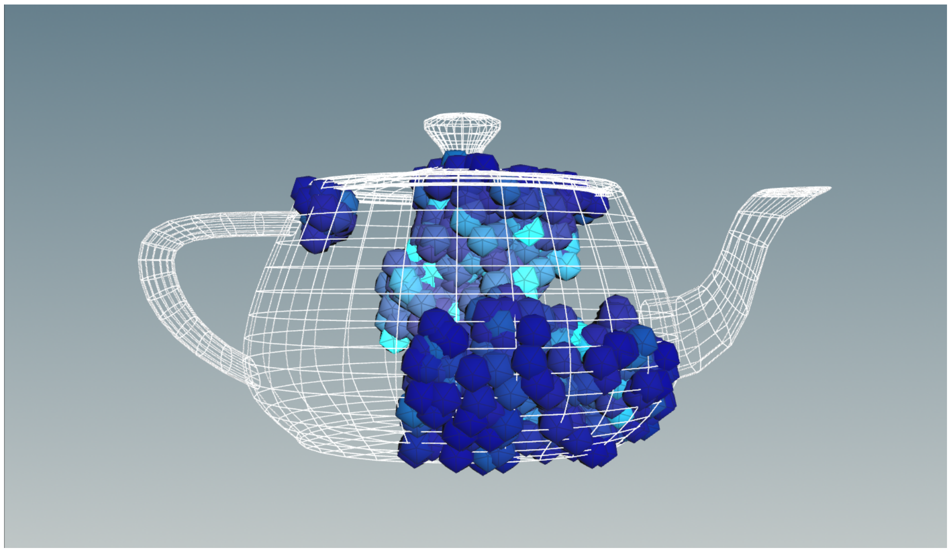

Figure 12: Object Fill: Uniform scale

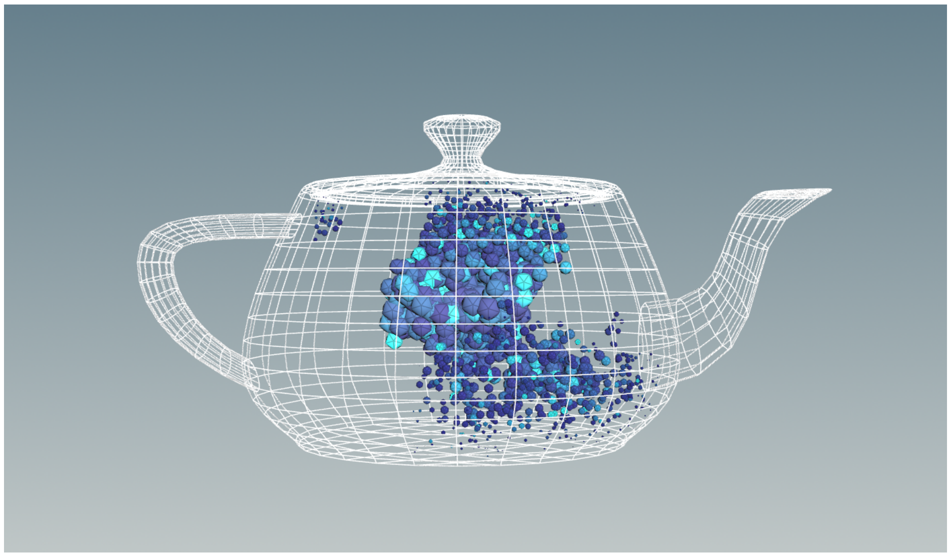

Figure 13: Object Fill: Scale varying with distance

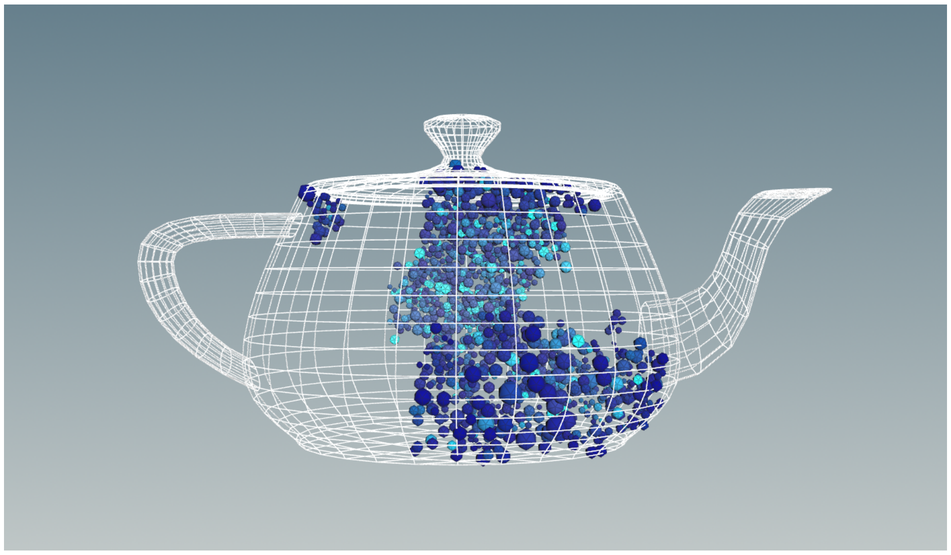

Figure 14: Object Fill: Check overlap

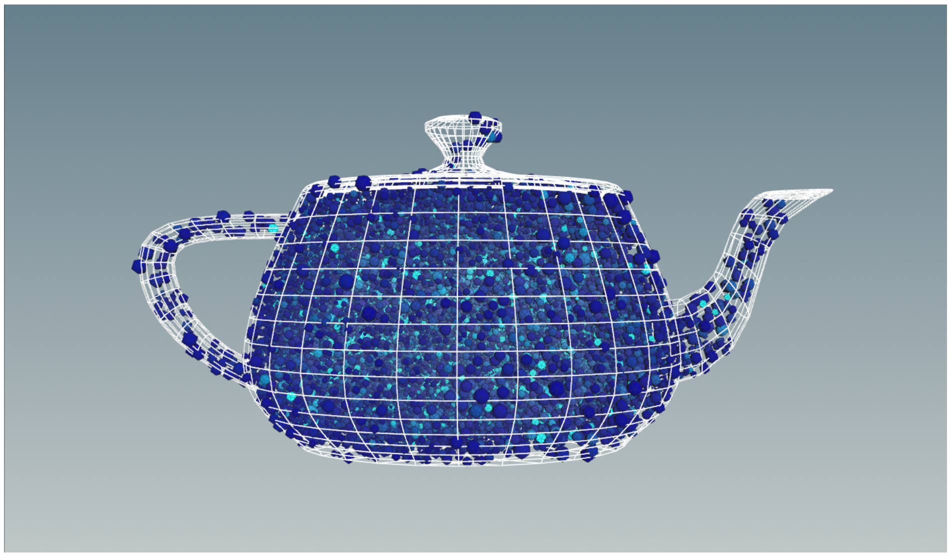

Figure 15: Object Fill: Complete Fill

Intersection

Wire Fill Polywire is added to the polylines between the points. Similar to the case for 3D Geometry and Text, wire could be either used with uniform width throughout or with applied variation. In case of 3D Geometry and Text, the scale variation was applied based on distance from the boundary. The same scale variation is applied here but with regards to distance from the points in the intersection area (called as intersection points going forward). So, further a point is from the intersection points, smaller with be the scale. 8

Firstly, for each point, the nearest intersection point is found using nearpoint(). The distance between each point and its respective nearest intersection point is found. This distance is then fitted between 0.1 and 0.9, so that regardless of the size of the object, the scale will always be within a reasonable range between 0.1 and 0.9.

- *@fit_ distance = fit01(distance(@P, nearpoint_ position), 0.1, 0.9)*

Applying the fitted valued directly will give us increasing scale as the distance increases. Since we require the opposite, we take the complement of the value before applying it to scale the width of polywire.

- *@point_ distance = 1 - @fit_ distance*

Moving to adding additional geometry like flowers or leaves, unlike the 3D Geometry and Text where all the end points within a certain distance from the geometry boundary are considered as seeds, the seeds for growth for Intersection are selected in a slightly different manner. A distance value is provided and if the distance of the point from the intersection point is greater than the provided distance, the point will be considered as a potential seed point. The second criteria of having one neighbor remains the same. This means, all the points beyond the said distance with one neighbor will sprout flowers or leaves. 11

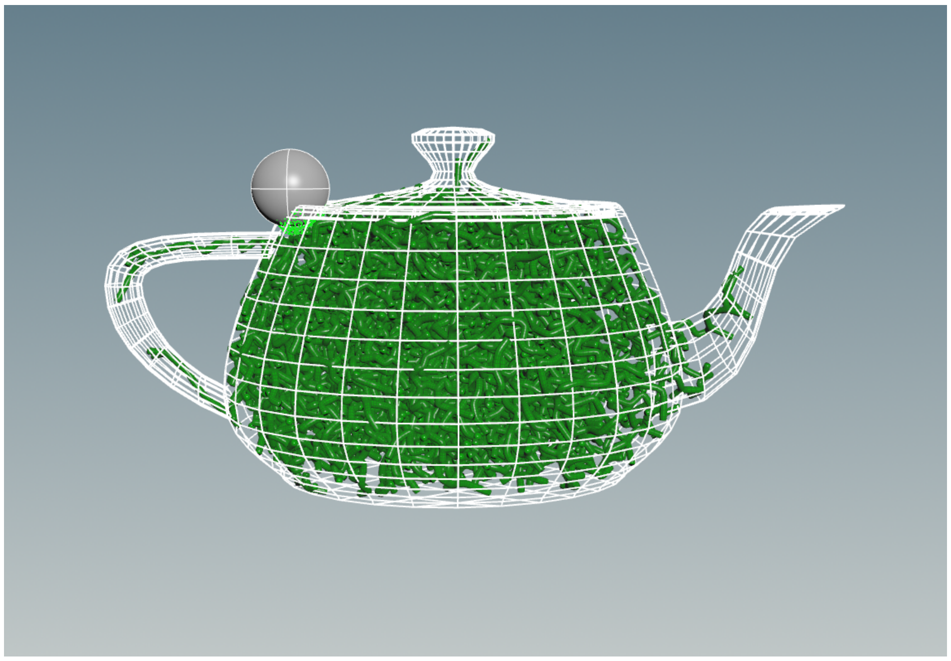

Figure 16: Wire Fill: Uniform width

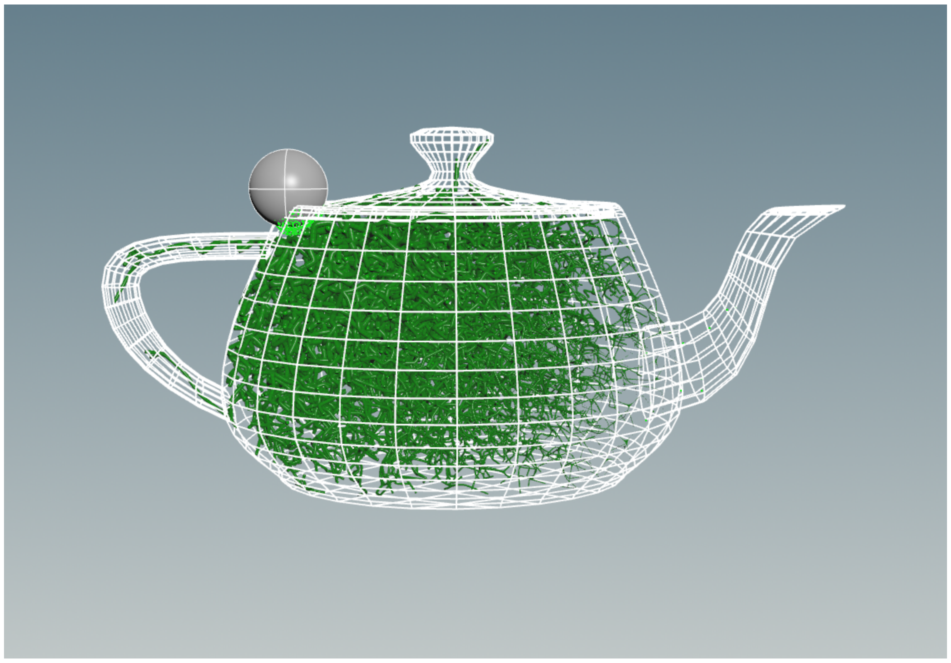

Figure 17: Wire Fill: Varying width with distance from the intersection points

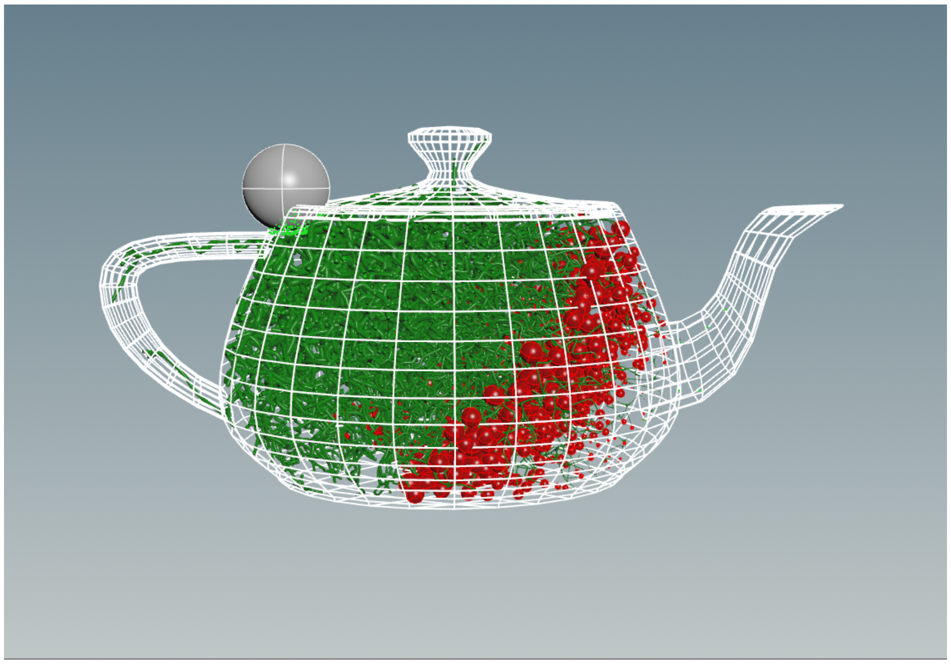

Figure 18: Wire fill on intersection with geometry on seeds

Object Fill The object fill for Intersection is similar to Text and 3D Geometry apart from scaling. Scaling in this case applies in a similar manner as the wire fill, the farther the point from the intersection points, smaller the object being copied onto it. If the overlap is to be removed, the method of scaling with respect to the nearest neighbor is invoked and each copied object will scale down accordingly to have its own independent space.

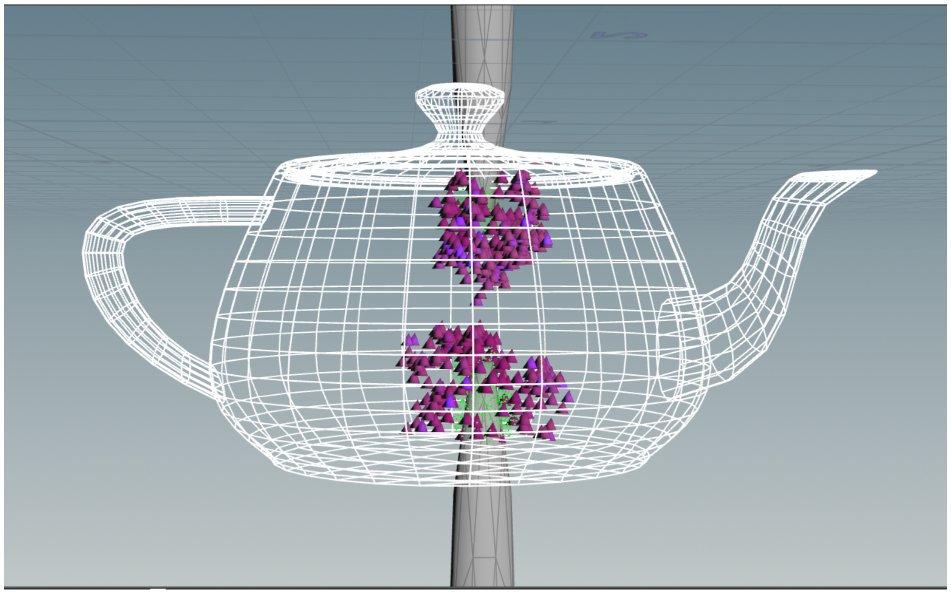

Figure 19: Object Fill: Area of interaction as the starting point

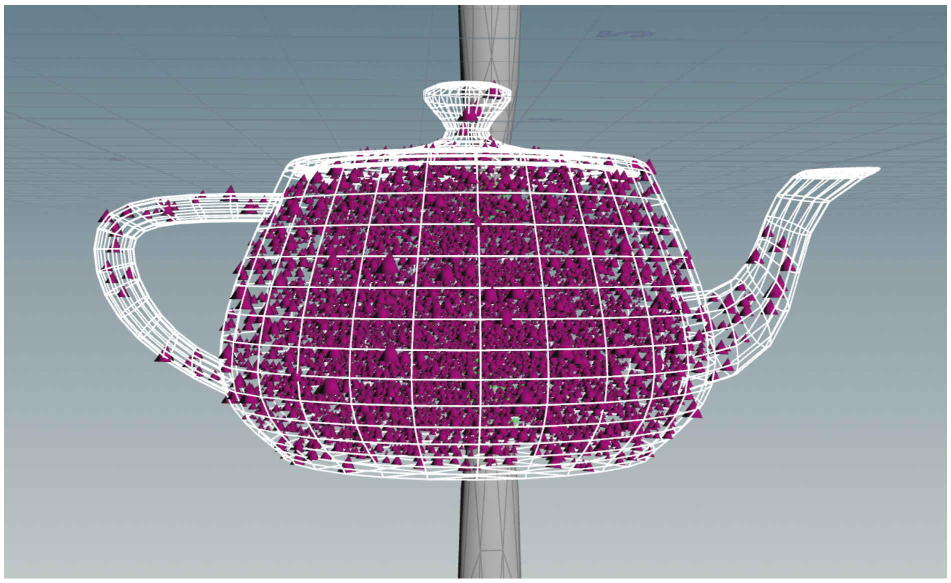

Figure 20: Object Fill: Complete fill

4.2 Problems Encountered and Attempted Solutions

A number of problems were dealt with during the development phase. Some of them were minor issues with easy fixes but others required experimenting with different probable proposals. Few of the major problems encountered and various solutions attempted are stated below.

4.2.1 Controlling Particle Emission

The emitters which are birthing random walkers can keep releasing particles throughout the course of simulation. When the growing aggregate reaches a certain distance from the emitter(s), it should stop the particle emission thus avoiding the repeated clustering of particles at the same location. After testing various approaches for controlling emission, two methods were arrived at.

First method is to nullify the birthing particles and second option is to remove the emitter itself.

In the first method, if the impulse emission is set to zero based on some condition, say frame number for instance, it will affect all the emitters and they all will stop emission simultaneously. This is not favorable since it might be possible that few of the emitters are still not in the contact region of the aggregate. This leaves us with the second option to remove the emitters as they reach the vicinity of the aggregate. While working on the method it was observed that though emitted particles and emission rate could be controlled from inside the POP network, the emitters themselves cannot be accessed from within POP network. This makes sense because, the initially scattered particles which are acting as emitter are at the SOP Level and cannot be accessed inside POP network directly.

A feedback system was created to make the emitters aware of the growing aggregate and this was done using a sover sop. Solver sop established connection between the emitters and the new aggregate enabling the emitters to perform a distance comparison and to stop when within a certain threshold.

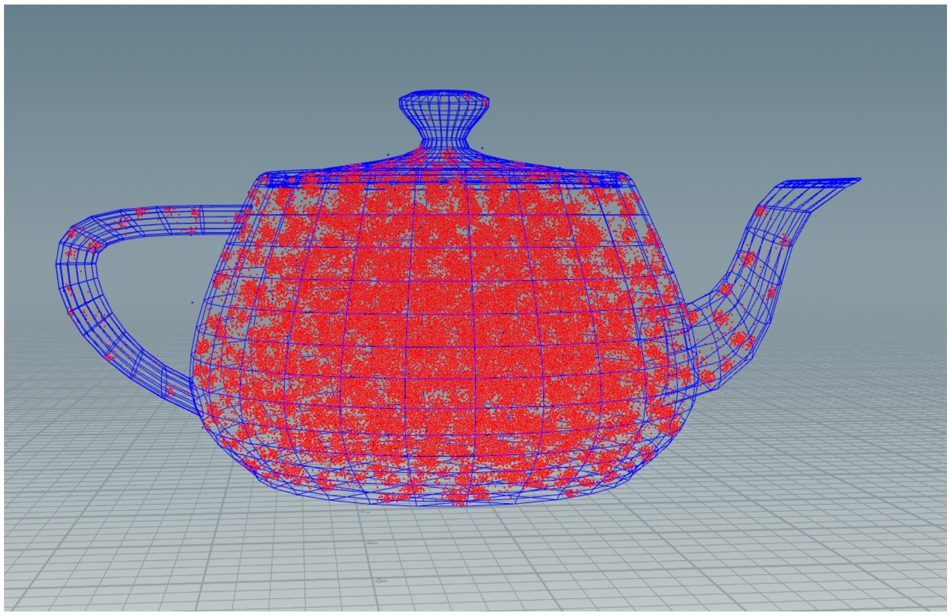

Figure 21: Clustering of particles because of continuous omission

4.2.2 Connecting Aggregate Particles

To establish a connection between the points, they were to be joined with polylines. This connection can be established either after all the points are stopped or at the instance each point is being stopped. Both the methods were tested. The first method was to check two nearest neighbors for each point in the stopped group using the *nearpoint()* function and connecting to them. This did connect the points but the joints were broken segments with pieces scattered all over the geometry. Per the understanding this was because, firstly, the *nearpoint()* function considers one of the nearest points of a point as itself and secondly, for a pair of points, the second near point for those points would be each other meaning that each point would be connecting to itself and one other point thus giving the discontinuous pieces. The neighbor point count was increased so that each point could connect to more than one point , but still the result was not as expected.

The second method was to connect the points at the time they are being stopped. Using *pcopen()* the closest point to each walker was found. Once the closest point was determined, connection was set only if it was part of the stopped group. This worked well giving a continuous spread of connected points. After investigation, this approach was found to be most commonly used when working with points also giving better results which is another reason for moving forward with this method. Tesla (2009)

The other alternative was to use the method of copy stamping as described by Cashmore (2009). He uses straight lines to be copied over stopped points. The distance between a point and the point it collided with was used as stamp to vary the length of the line that is being copied to ensure that line is of length that is equal to the distance between the points it is being placed between.

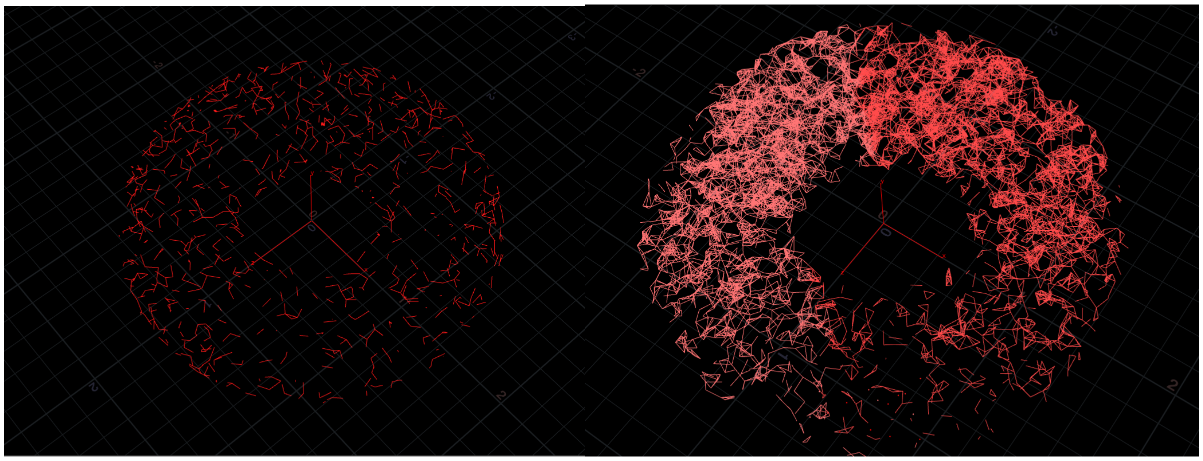

Figure 22: Connecting aggregate point with two neighbors (left) and connecting with multiple neighbors

4.2.3 Normal Inversion for External Files

Apart from using the default geometries provided by the tool, the user has the flexibility to use any desired geometry. Keeping note that this geometry could be a *.bgeo* or *alembic (.abc)* file, the tool's compatibility had to be tested with both these file formats.

A file of each format was imported and tested but it was noticed that particle emission was not being handled properly. The emission was happening only at few selected parts of the geometry and not throughout the geometry as was supposed to be. After several tests and modifications, it was found geometry normals were not properly oriented. At certain locations, normals were flipped and that is why the particle flow in those areas was interrupted.

The reverse node in houdini to change the direction of the normals could not be used since that would change the direction of even those normals which are properly oriented.

In the Figure 24, all the normals of the geometry are flipped so using a *reverse* node will make sense and will work as expected correcting the orientation to make them point outwards.

As for the Figure 25, the model was exported from Maya and the possible reason of normal inversion for just one half of the geometry could be the mirroring of the geometry while modeling.

Even after several attempts, a global fix for the problem couldn't be established and is advised to keep a check on normals during modeling phase.

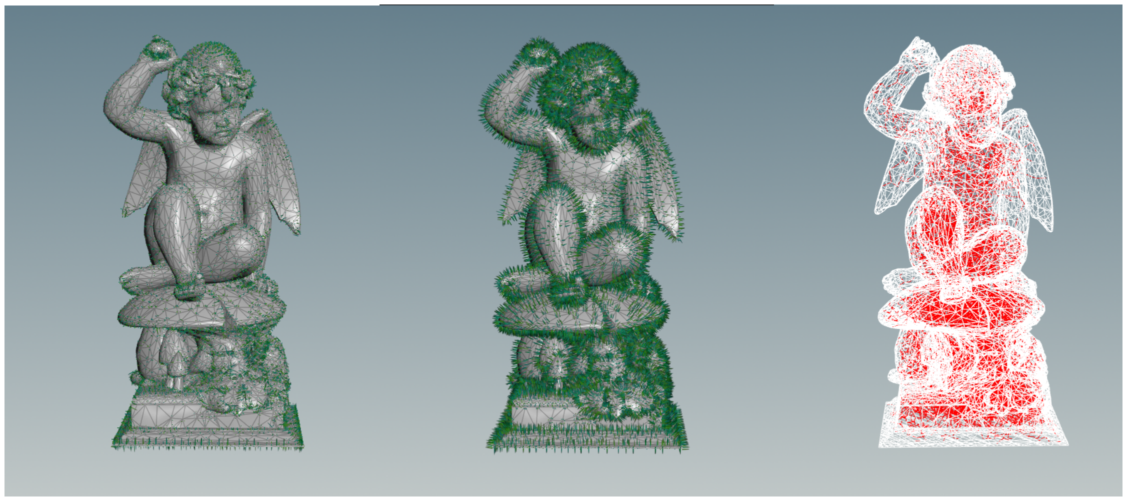

Figure 23: (From left) Normal inversion for the geometry (*.bgeo*): model with inverted normals, model with fixed normals and model with aggregation applied

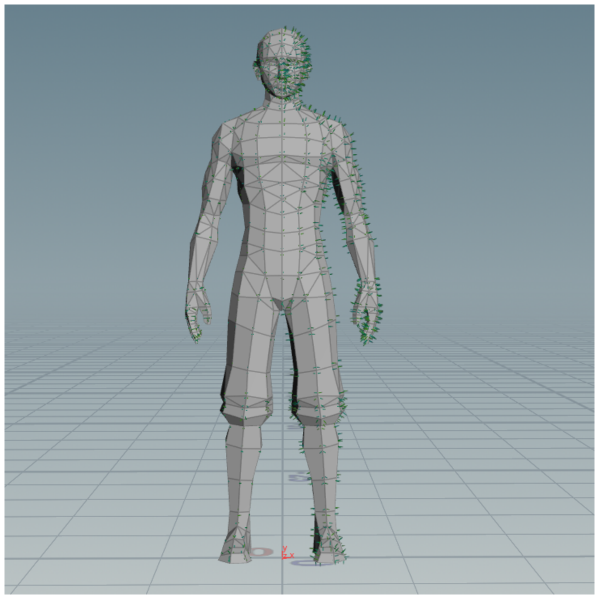

Figure 24: Normal inversion for the geometry: *.abc*

4.2.4 IsoOffset, Converting Geometry to Volume

For scattering points inside a geometry, the geometry is first converted to volume using *IsoOffset*. During this conversion, there could be possibility of encountering noise or stretch lines. The reason for such appearance is not thoroughly clear but it is observed that this is independent of whether the geometry is imported from outside or is modeled in Houdini. Also it is not necessary that all geometries when converted to volume will have the stretch lines.

One possible fix tested for this was to use Particle Fluid Surface. Once the geometry is selected, a scatter node is appended to it which will scatter points on the surface of the geometry following which the Particle Fluid Surface is used to generate a surface around the particles. The surface definition could be modified by changing Particle Separation parameter. This will generate a surface in the shape of the passed geometry. After the desired shape is achieved, *IsoOffset* could be used which will give volume properly wrapped within the geometry provided without any stretch lines.

The fix is not embedded in the tool because the surface definition is not constant and varies based on the shape and complexity of the geometry. All the default geometries provided by the tool are tested to have no stretch lines and for any geometry imported by the user, should the noise or stretch lines appear, the fix can be used to have proper particle scattering.

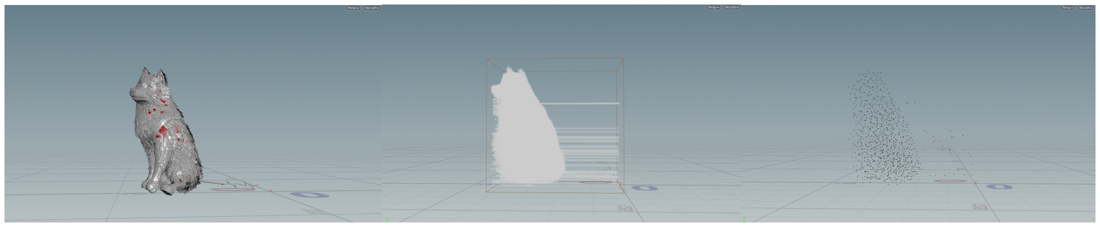

Figure 25: (From left) Original geometry, converted to volume with stretch lines and points scattered in the volume

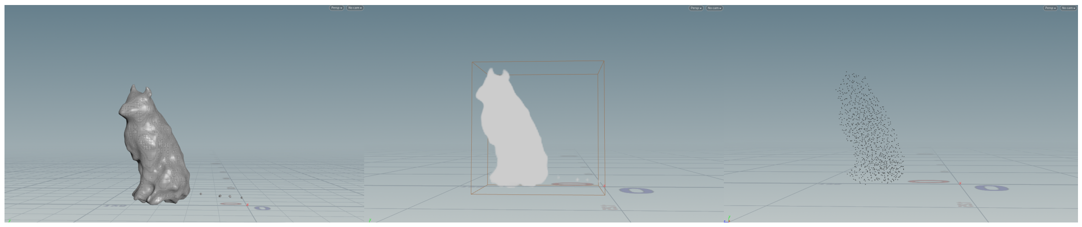

Figure 26: (From left) Particle Fluid Surface applied on original geometry, converted to volume and scattered points in volume

4.2.5 Geometry Level Transformation

Transformations could be performed at node level or at geometry level. For some reason, simulation appears to be unresponsive to the geometry level transformations. When the geometry level transformations are applied, the object is appropriately transformed leaving the particles at their original location. This will make the transformed geometry treat the particles as external particles thus killing them. If the same transformation is applied at the node level, the simulation work without an issue.

The possible fix tested for it was to change the *Geometry Source* from *First Context Geometry* (i.e. solver) to *Use Parameter Values* with *Use Object Transform* in pop source inside pop network. Using the latter option will transform all the particles with the main object. The simulation works but since it is not specified that particle emission should happen only from walker group, both walker and aggregate points will start emission. To enable emission only from walkers, the *Source Group* is set to walkers with *Group Type* as points. When the simulation starts, it fails to recognize the aggregate points and so the growth propagation doesn't happen. Also if both the groups are considered as emitters, the simulation works and the aggregation happens, but the pop network doubles up the transformation and shifts the simulation outside the original object.

The tested fix didn't work as expected and was abandoned. A permanent fix for the issue is one of the future works to be looked into. Till then, if any transformations are required to be performed, node level transformations can be relied upon.

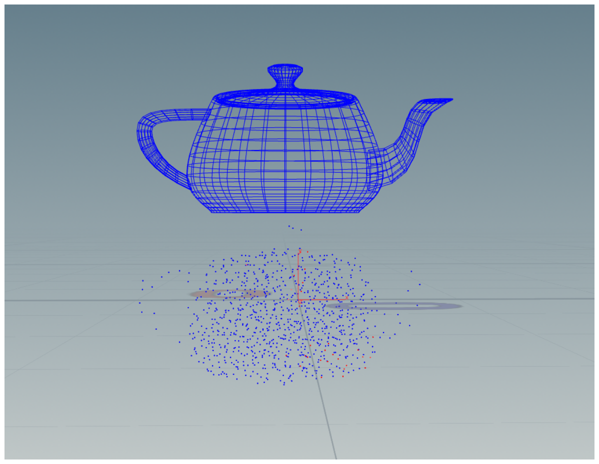
Figure 27: Geometry level transformation transforming the geometry without particles

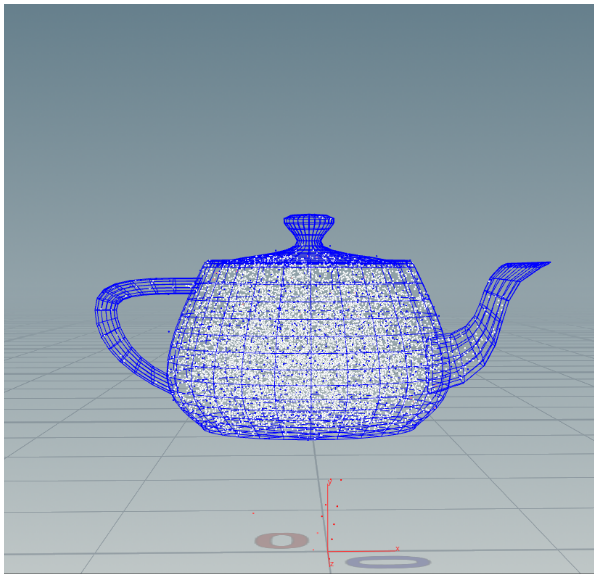
Figure 28: Using Object Transform to set points back into the geometry and starting emission

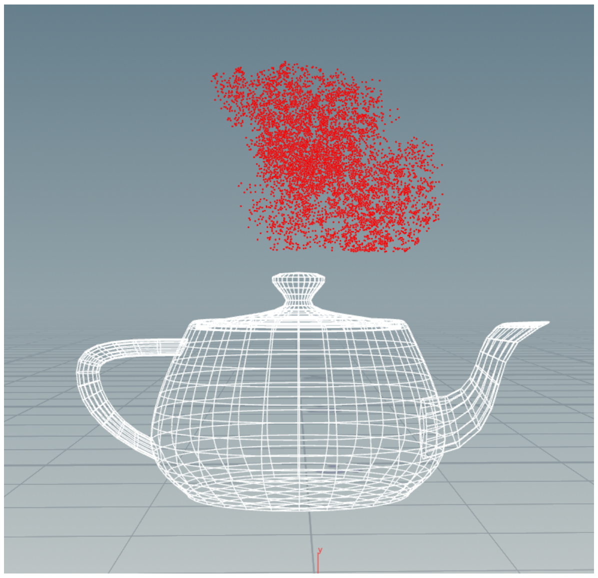

Figure 29: Simulation outside the object because of transformation being doubled inside pop network

4.3 Efficiency

The tool requires simulating particles inside the 3D object. Time taken for the process mainly depends on the search distance. The search distance is directly proportional to the size of the object. Smaller the search distance would mean processing larger number of particles per frame and might take 1:30-2 minutes to fully cover the object. With an optimal search distance (which is decided relative to the object size), the time could be reduced down to 45-60 seconds. An even higher distance would take about 15-30 seconds but the downside would be, the entire object will not be fully covered.

It is definitely slower than Houdini's inbuilt fractal generator i.e. L-Systems which generates simple patterns almost instantly and the more complex ones in less than 30 seconds. But compared to attempting to build a similar structure manually which might take several hours, the tool is a better option. Also within 3-4 attempts, the user can establish the optimal search distance which will optimize both the number of particles processed per frame and the time taken for processing.

5 Applications

The section deals with three of the many possible ways the tool could be used for. Two different 3D models: teapot and rubbertoy and a 3D text has been used to show the adaptive ability of the tool.

5.1 Vines

The connecting lines when applied with polywire could be used for vine like growth effect. Polywire with the option of decreasing width with distance enabled has been used. This would, as stated before have finer branches near the boundary of the geometry and thicker ones towards the center.

Default flowers are copied onto the seeds of growth, again with a variation effect with respect to distance to have a better visual appeal. Instead of simply copying, the growth of the flower can also be animated by scaling them with time but that aspect has not been considered here.

Apart from just the addition of the flowers (or any user preferred geometry), the user is given the control to remove flowers from any location of the geometry. This is accomplished using Paint tool. The removal area is painted upon and the seeds of growth which inherit the painted color using attribute transfer will no longer act as seeds of growth and any geometry placed on these points will be removed. The purpose of the removal option is to give user further control over where to add the additional geometry.

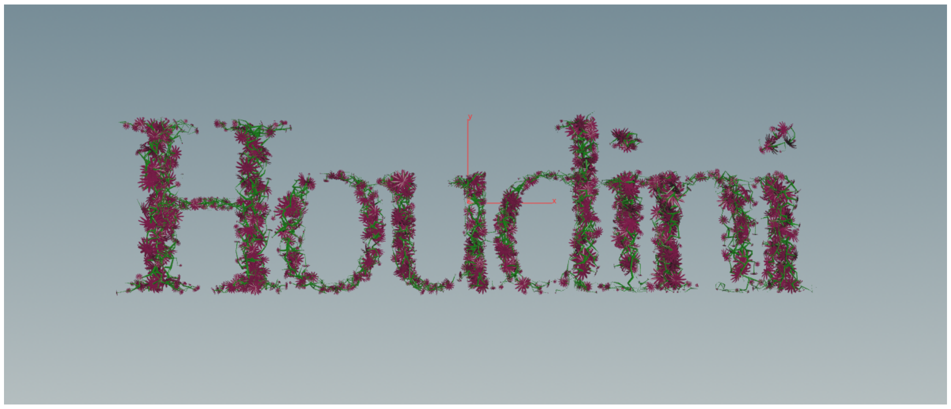

Figure 30: Vine growth on text

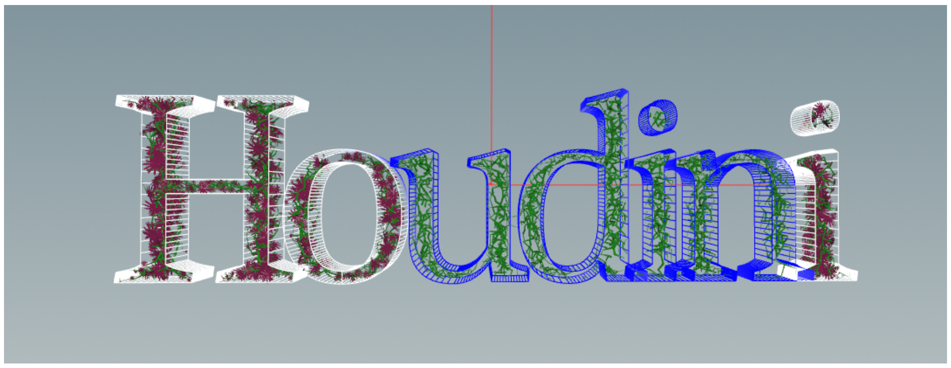

Figure 31: Painted area for removal of geometry

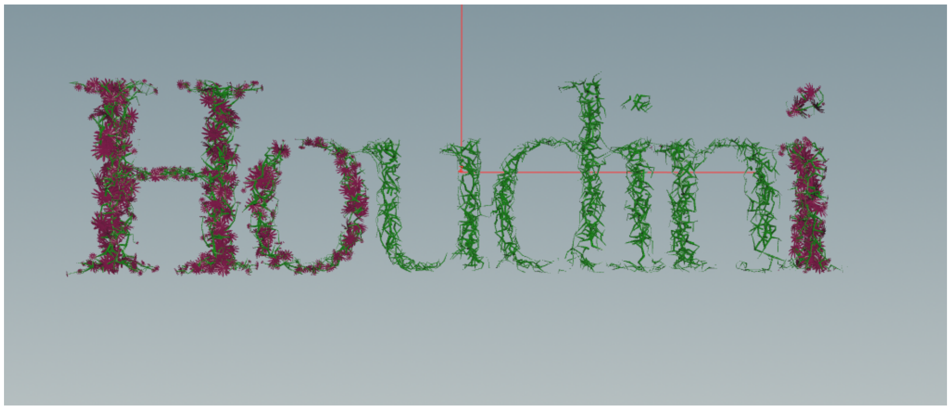

Figure 32: Vine post geometry removal

5.2 Uncanny Growth

The propagation effect of DLA can be used to show spreading of disease, virus or any such uncanny growth effect. The application shows the growth of an organic like matter which spreads inside the geometry and subsequently engulfs it. For this effect, metaball with the copy to points option was chosen. The reason of choosing metaball is because it provides a simple method for creating organic looking surface. The effect has the feel of a blobby substance covering the geometry.

It was tested with both uniform color and color variation with distance, each giving interesting results in its own stand.

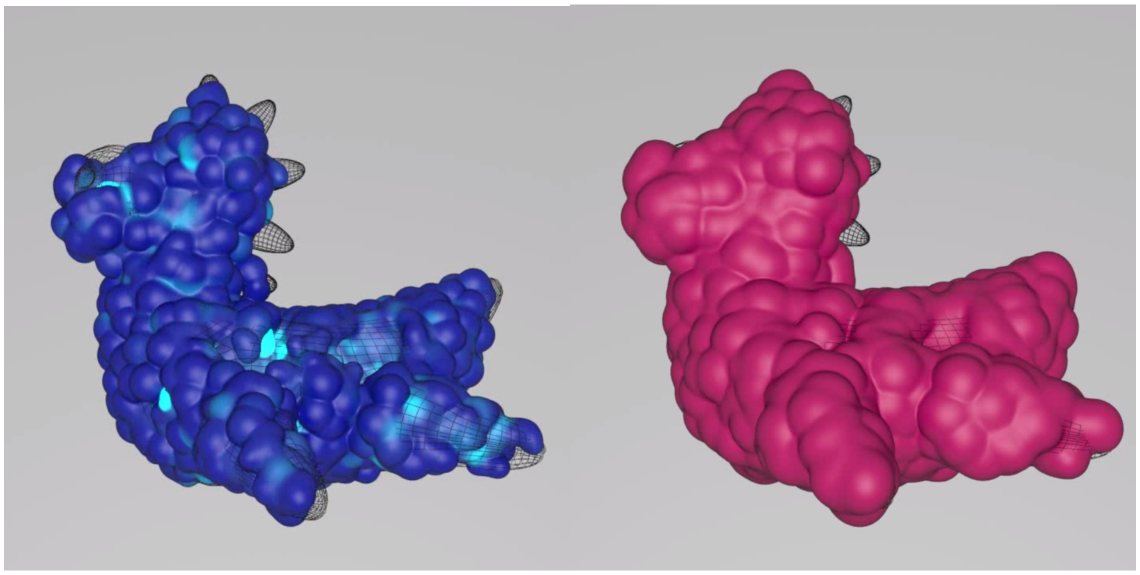

Figure 33: Growth propagation using metaball in rubbertoy: Color variation with distance (left), Uniform color

5.3 Injection

The injection effect is similar to the uncanny growth effect with the spread of blobby substance in the geometry. It is called the injection effect because it appears as if the two rods are injecting or releasing the blobby material into the geometry. The result is similar to the spreading effect shown in the wolverine adamantium scene, X-Men Origins Wolverine. X-Men Origins Wolverine (2013)

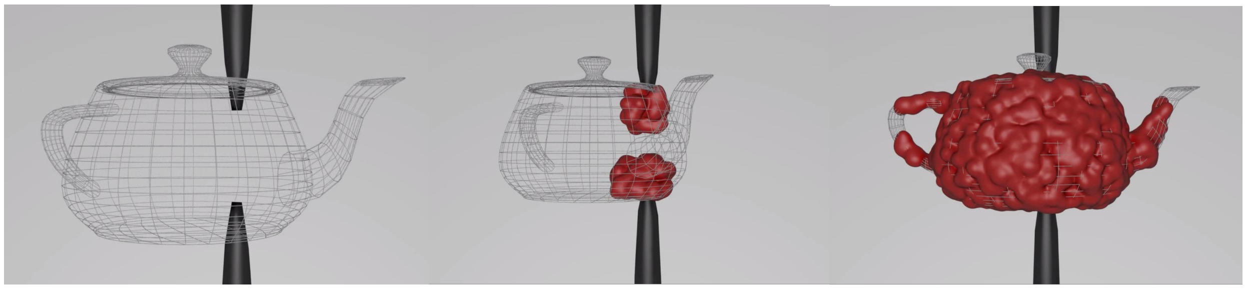

Figure 34: (From left) Injection objects interacting with the geometry, growth starting at intersection points, completely filled main geometry

6 Conclusion

The tool developed allows user to input any geometry and fill it with points or connected lines. These points and lines could be manipulated or topped with geometry to create realistic growth effect over the chosen geometry. The tool is highly adaptable as the user has the choice to update the object the geometry is being filled with or add his/ her own flowers and leaves for the vines. All these objects can be used with uniform or multiple colors or used with different shaders to create the look based on the production purpose it is being used for. Also the system is extremely flexible should there arise a need to extend it further or as a prospective research option.

It is especially handy as it is time and cost efficient when it comes to creating textures that would grow and animate inside an object.

Although the tool is capable of successfully achieving many effects, there still remains a plethora of possibilities in terms of better controls, animation aspects and speed checks which is discussed in the following section as the potential future works.

7 Future Work

The system designed could be further advanced by having control over the simulation start and finish frames or providing the ability to limit the growth to a particular section of the geometry.

Another aspect that requires special attention is the number of particles. Larger the geometry, greater will be the particles needed to completely fill the geometry which will slow down the process incredibly. In such a case, the system needs to have the mechanism to reduce the point density through minimum distance constraint i.e. removing the points within a particular distance of each point and thus bringing down the number of points significantly but leaving the overall shape unaffected.

8 References

Aker, E., 1997. *The Density-Density Correlation Function.*
Available from: http://folk.uio.no/eaker/thesis/node56.html [Accessed 10 July 2017].

Bourke, P., 1991. *DLA - Diffusion Limited Aggregation.*
Available from: http://paulbourke.net/fractals/dla/ [Accessed 24 July 2017].

Bourke, P., 2006. *Computers and Graphics. Constrained Diffusion Limited Aggregation in 3D*, 30(4), 646-649.
Available from: http://paulbourke.net/fractals/dla3d/ [Accessed 26 July 2017].

Bourke, 2004. *Diffusion Limited Aggregation (DLA) in 3D.*
Available from: http://paulbourke.net/fractals/dla3d/ [Accessed 4 July 2017].

Cashmore, M., 2009. *Applying Aggregation over 3D Surfaces.*
Available from: https://nccastaff.bournemouth.ac.uk/jmacey/MastersProjects/MSc09/index.html [Accessed 8 August 2017].

Devaney, R., 1995. *Fractal Dimension.*
Available from: http://math.bu.edu/DYSYS/chaos-game/node6.html [Accessed 16 July 2017].

Ergen, S., *L Systems in computer graphics.*
Available from: http://www.selcukergen.net/ncca_lsystems_research/lsystems.html [Accessed 21 May 2017].

Feriani, M., 2016. *Ice Growth Simulation.* Video. Vimeo. Available from: https://vimeo.com/201539220 [Accessed 2 August 2017].

Ferrysienanda, E., 2014. *A Study of Procedural Growth Algorithms and its Implementation in Houdini.* Available from: http://edwardfx.weebly.com/uploads/9/0/7/5/9075896/innovationsreport.pdf [Accessed 15 July 2017].

Halsey, T., 2000. *Diffusion Limited Aggregation-A Model for Pattern Formation.* Physics Today, 36-41. Available from: https://web.njit.edu/~kondic/capstone/2015/phys_today_DLA.pdf [Accessed 8 August 2017].

Lomas, A., 2005. *Aggregation, Complexity out of Simplicity.*
Available from: http://www.andylomas.com/aggregationImages.html [Accessed 18 July 2017].

Prusinkiewicz, P. and Lindenmayer, A., 1990. *The Algorithmic Beauty of Plants.* New York: Springer. Available from: http://algorithmicbotany.org/papers/abop/abop.pdf [Accessed 11 August 2017].

Rampe, J., 2016. *Diffusion Limited Aggregation.* Wordpress.
Available from: https://softologyblog.wordpress.com/2016/08/29/diffusion-limited-aggregation-2/ [Accessed 7 August 2017].

Rampe, J., 2017. *Pushing 3D Diffusion Limited Aggregation even further.* Wordpress.
Availabl from: https://softologyblog.wordpress.com/2017/05/22/pushing-3d-diffusion-limited-aggregation-even-further/ [Accessed 7 August 2017].

Sander, L., 1986. *Fractal Growth Process.* Nature Publishing Group,
Available from: https://www.nature.com/nature/journal/v322/n6082/abs/322789a0.html [Accessed 1 May 2017].

Shiffman, D., 2012. *The Nature of Code.* Image.
Available from: http://natureofcode.com/book/chapter-8-fractals/ [Accessed 14 August 2017].

Smith, R., 2013. *Fractals: What are some practical applications of L-systems?.* Quora.
Available from: https://www.quora.com/Fractals-What-are-some-practical-applications-of-L-systems [Accessed 14 August 2017].

Stock, M., 2003. *Snow, Chaos and Fractals.*
Available from: http://markjstock.org/dla3d/ [Accessed 7 August 2017].

Tesla, 2016. *dla.* Video. Vimeo.
Available from: https://vimeo.com/181075748 [Accessed 9 August 2017].

Tesla, 2016. *Diffusion Limited Aggregation.* ODForce.
Available from: http://forums.odforce.net/topic/27179-diffusion-limited-aggregation/ [Accessed 12 July 2017].

Witten, T. and Sander L., 1983. *Diffusion Limited Aggregation.*
Available from: https://pmc.polytechnique.fr/pagesperso/dg/cours/biblio/PRB%2027,%205686%20(1983)%20Witten,%20
limited%20aggregation%5d.pdf [Accessed 8 May 2017].

X-Men Origins Wolverine, 2013. *LOGAN GETS INJECTED WITH ADAMANTIUM PART 1 (X MEN ORIGINS WOLVERINE).* Video. YouTube.
Available from: https://www.youtube.com/watch?v=4M8LoVgPxuQ [Accessed 9 August 2017].

9 Appendix

Algorithm 3 Separating walker and aggregate points

```
/*
Points which are almost red are grouped as aggregate and rest as \
    walkers. This is because while painting the geometry surface, \
    color applied may not be {1,0,0} but close to it.
*/

if(@Cd.r>0.8 && @Cd.g<0.2 && @Cd.b<0.2)
    {
        @group_aggregate = 1;
    }
else
    @group_walker = 1;
```

Algorithm 4 Connecting the aggregate points

```
/*
The walker points are stopped and grouped separately when they are within a \
    particular distance of the aggregate points and each stopped point is \
    connected using polyline to the point which stopped it.
*/

//Setting search_dist as a user input
float search_dist = ch("Distance");

//Max points is 2 because one of the nearest point it will consider is itself
int maxpt = 2;

v@direction;
vector pos;

//Searching the nearest point within search_dist
int nearpt = pcopen(0, "P", @P, search_dist, maxpt);

//Iterating through the nearest points
while(pciterate(nearpt))
{
int ptn = -1;

//Importing the point number of the nearest point
pcimport(nearpt, "ptn", ptn);
    //Checking if the point is part of "stopped" or "aggregate" group
    if(inpointgroup(0, "stopped", ptn) == 1 || inpointgroup(0, "aggregate", \
        ptn) == 1)
    {
        //Adding a polyline between this point and the previously stopped/ \
            aggregate point
        int prim = addprim(geoself(), "polyline");
        addvertex(geoself(), prim, @ptnum);
        addvertex(geoself(), prim, ptn);

        //Adding this point to the stopped group and removing it from the \
            stream and walker groups
        setpointgroup(geoself(), "stopped", @ptnum, 1);
        setpointgroup(geoself(), "stream_seed", @ptnum, 0);
        setpointgroup(geoself(), "walker", @ptnum, 0);

        //Stopping the point by setting velocity to 0
        @v = 0;

        //Changing color for identification
        @Cd = {1,0,0};

        //Position of the nearst point
        pos = point(0, "P", ptn);

        /*
        Calculating difference between this point and the point which stopped\
            it.
        This difference will be set as direction for the normal so that when \
            a geometry is placed, it will be positioned correctly.
        */
        @direction = @P - pos;
    }
}
```

Algorithm 5 Controlling emitters

```
/*
Checking if the emitter is within the "search_dist" of the growing aggregate.
If yes, remove the emitter from the group.
*/

int nearpt;
float @dist;
vector pos;
float search_dist = ch("Distance");

//Find the nearest point to this point on the aggregate
nearpt = nearpoint(1, @P);

//Calculate the position of the nearest point
pos = point(1, "P", nearpt);

//Calculate the distance between this point and the nearpt
@dist = distance(pos, @P);

//If this point is within the search distance, remove from walker(emitter)
if(@dist <= search_dist)
{
    @group_walker = 0;
}
```

Algorithm 6 Removing extra points

```
/*
Check for points not in stopped or aggregate group and remove them.
*/

//Select points not in stopped group
if(inpointgroup(0,"stopped",@ptnum)!=1)
{
    //Check if they are in aggregate group
    if(inpointgroup(0,"aggregate",@ptnum)==1)
    {
        //if the points are in the aggregate group but are not connected, \
            remove them
        if(neighbourcount(0, @ptnum) == 0)
            removepoint(0, @ptnum);
    }

    //if the points are neither in stopped nor in aggregate remove them
    removepoint(0,@ptnum);
}
```

Algorithm 7 Distance from object boundary

```
/*
Calculates distance from the object boundary and uses this distance to drive \
    the polywire width or object size for 3D Geometry and Text.
*/

f@mydist;

//Chekcing the distance of this point from the boundary of the input geometry
@mydist = xyzdist(1,@P);
```

Algorithm 8 Distance from intersection points

```
/*
This part is used for setting the varying size or width of the  object fill \
    and wire fill for Intersection.
As the distance from the visualization points increases, the size or width  \
    should decrease.
*/

int near;
vector near_position;
float @fit_dist;

f@point_distance;

//Find the nearest visualization point to this point
near = nearpoint(1, @P);

//Position of the "near" point
near_position = point(1, "P", near);

//Distance between "near" point and this point and fitting it between 0 and 1
@fit_dist = fit01(distance(@P, near_position), 0, 1);

//Calculating complement since increasing the distance should decrease the \
    width/ size
@point_distance = 1 - @fit_dist;
```

Algorithm 9 Checking geometry overlap

```
/*
In case of Object Fill, making sure the copied geometries do not \
    interpenetrate.
*/

f@dist;

//Taking search_dist as the user input
float search_dist = ch("Distance");
vector pos;

//Find the nearest point to this point within search_dist
int handle = pcopen(0, "P", @P, search_dist, 2);

//Iterating through the retrieved points
while(pciterate(handle))
{
    //Importing the position
    pcimport(handle, "P", pos);

    //If the point is not itself, calculate the distance between this point \
        and its neighbour
    if(@P != pos)
    {
        @dist = distance(pos, @P);
    }
}
```

Algorithm 10 Setting seed points for additional geometry: 3D Geometry and text

```
/*
Setting seed points for the placement of additional geometry. These seed \
    points are decided based on two criterias: 1. Their nearness to the \
    object boundary and 2. The count of neighbours to check if its an end \
    point.
*/

//Checking the distance from the boundary. Setting a very small distance to \
    keep the additional geometry closer to the boundary and away from central\
     clustering.
if(@mydist < 0.05)
{
    //Checking for points with only one neighbour
    if(neighbourcount(0, @ptnum) == 1)
    {
        //Grouping the points under a new group
        @group_seeds = 1;

        //Setting a new color for identification
        @Cd = {0,1,0};
    }
}
```

Algorithm 11 Setting seed points for additional geometry: Intersection

```
/*
This part defines from where the placement of addtional geometry will start.
The distance specified by the user is used as the minimum distance from  the \
    visualization points.
If any points lie beyond that distance,  they would be considered as seeds \
    for growth.
*/

float choose_distance = ch("Distance");

// Checking if the distance of this point from the visualization points is \
    greater than the user input distance. Using @fit_dist from Algorithm 8.
if(@fit_dist >= choose_distance)
{
    //Checking for points with only one neighbour
    if(neighbourcount(0, @ptnum) == 1)
    {
        //Grouping the points under a new group
        @group_seeds = 1;

        //Setting a new color for identification
        @Cd = {0,1,0};
    }
}
```