Applying aggregation over 3D surfaces

Creating geometry and controlling shaders

Masters Thesis

Michael Cashmore

NCCA Bournemouth University

August 21, 2009

Contents

1.0 Introduction

Creating structured randomness in computer graphics is an extremely important and useful part of virtually any production. Lines that are too straight and clean will suffer from looking man made. Fractal patterns offer computer artists a way of having controlled randomness that can be used in a variety of circumstances.

L-systems have long been a popular method of creating fractals, their apparent randomness that still has structure provides the user with natural looking shapes which translates into many areas. Uses of L-systems range from modelling trees or plants, to DNA chains or even replicating crack propagation.

Diffusion-limited aggregation, DLA, is a model used by scientists that is also capable of producing fractals. In simple circumstances the outcome of DLA simulations can be very similar to that of a l-system, but in more complicated scenarios they can be far more powerful. L-systems will start with an axiom and grow based on a set of rules but they have no way of adapting to their surroundings unlike DLA which has the advantage that it provides a unique fractal geometry which will. Changing the environment a DLA simulation takes place in will alter the fractal it produces whereas l-systems are unable to respond to this.

Side Effects software Houdini comes with a powerful built in l-system generator capable of producing a variety of shapes and patterns. The disadvantage of this is that there is no way of making them grow over a piece of complex geometry, adapting to their surroundings and completely covering the geometry in fractal growth. Typically DLA is simulated on a flat 2D plane but it was the aim of this project to produce a DLA system that was capable of generating fractal patterns over 3D models. The system designed is completely procedural and is adaptable to virtually any shape of geometry. Once the fractal geometry is created it can be used to replicate structures such as vines or veins over an object, or to create a texture or displacement map. This can then be used at shader level to create a variety of effects such as lava covering a character or rust propagation.

2.0 Previous Work

2.1 L-systems

Lindenmayer developed the first formal language for describing l-systems in 1968 in his paper The Algorithmic Beauty of Plants [1]. His paper describes an extremely powerful language that is still used today and can create complex structures from very simple rules. Nowadays the use of l-systems is widespread, they are routinely used for modelling trees and plants or even DNA chains for example.

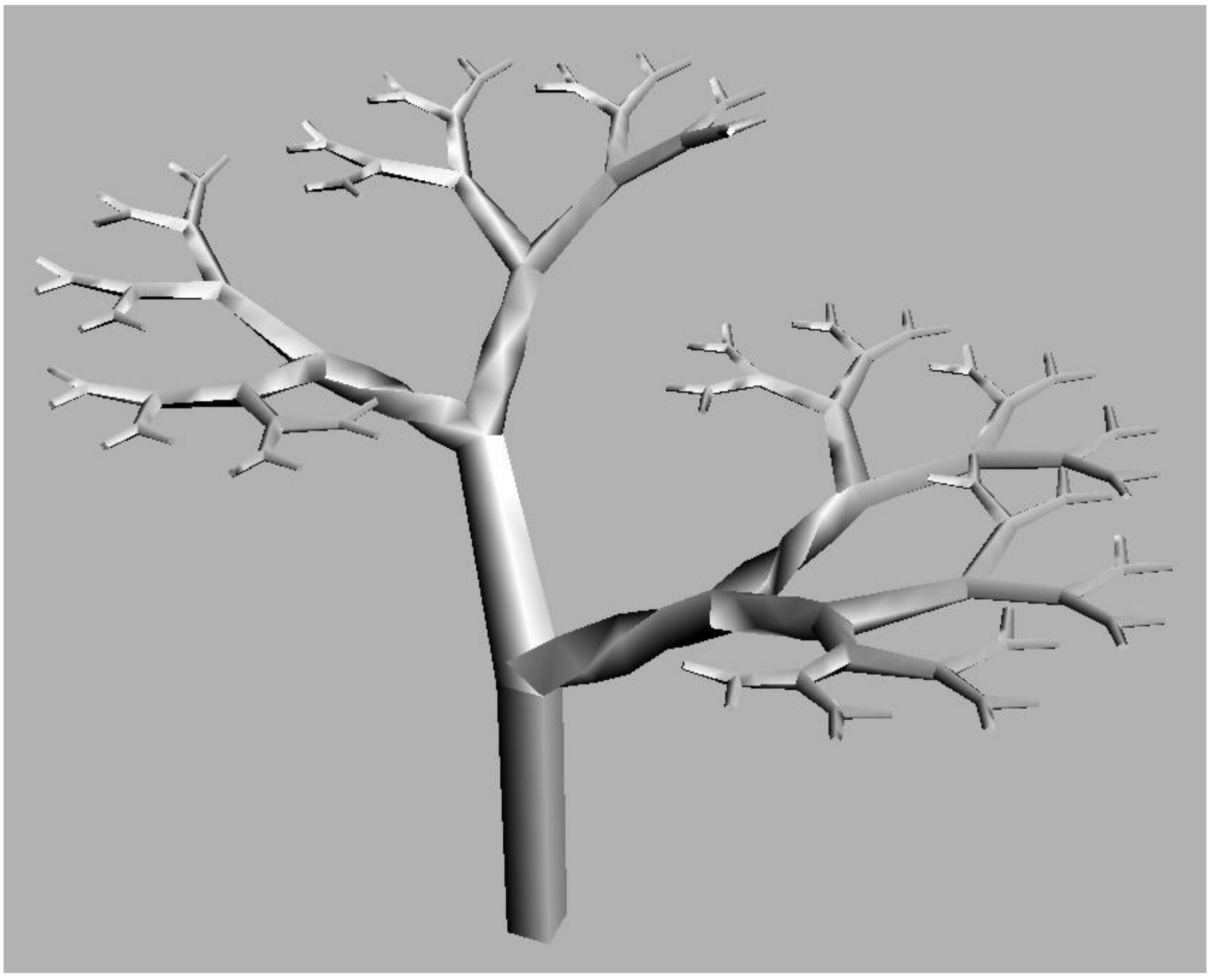

Figure 1: An example of an l-system tree generated using Houdini's L-system SOP. [2]

Houdini comes built with its own powerful l-system generator the Lsystem SOP. This lets the user use pre-made structures as shown in figure 1, or they can use a language similar to that developed by Lindenmayer to generate their own structures.

2.2 Diffusion-limited aggregation

Diffusion-limited aggregation is a model first proposed by Witten and Sander in 1981[3]. The model is an idealisation of the process in which matter randomly combines to form structures, such as crystals formation, bacteria growth and soot dendrites.

Scientists have used DLA in the exploration of oil resources, where oil is embedded in the sand by water pressure into the sand. The surface between the water and oil exhibits shapes which can be simulated by DLA. Other uses are is the field of catalyst formation. In processes for chemical engineering catalysts are used as a way to increase or decrease the rate of a chemical reaction. They do not alter the end product of a reaction but they do change the speed at which it occurs. Catalysts need a large surface area to be most effective and often are arranged with a largely branched structure with many holes. This shape is akin to structures which are simulated by DLA [4].

Perhaps the area which is most relevant to this project is the work by Andy Lomas [5]. Lomas grew sculpted shapes by simulating the path of millions of particles which flowed randomly in a field of forces. Over time they aggregated over a surface to produce structures like those shown in figure 2 of great complexity.

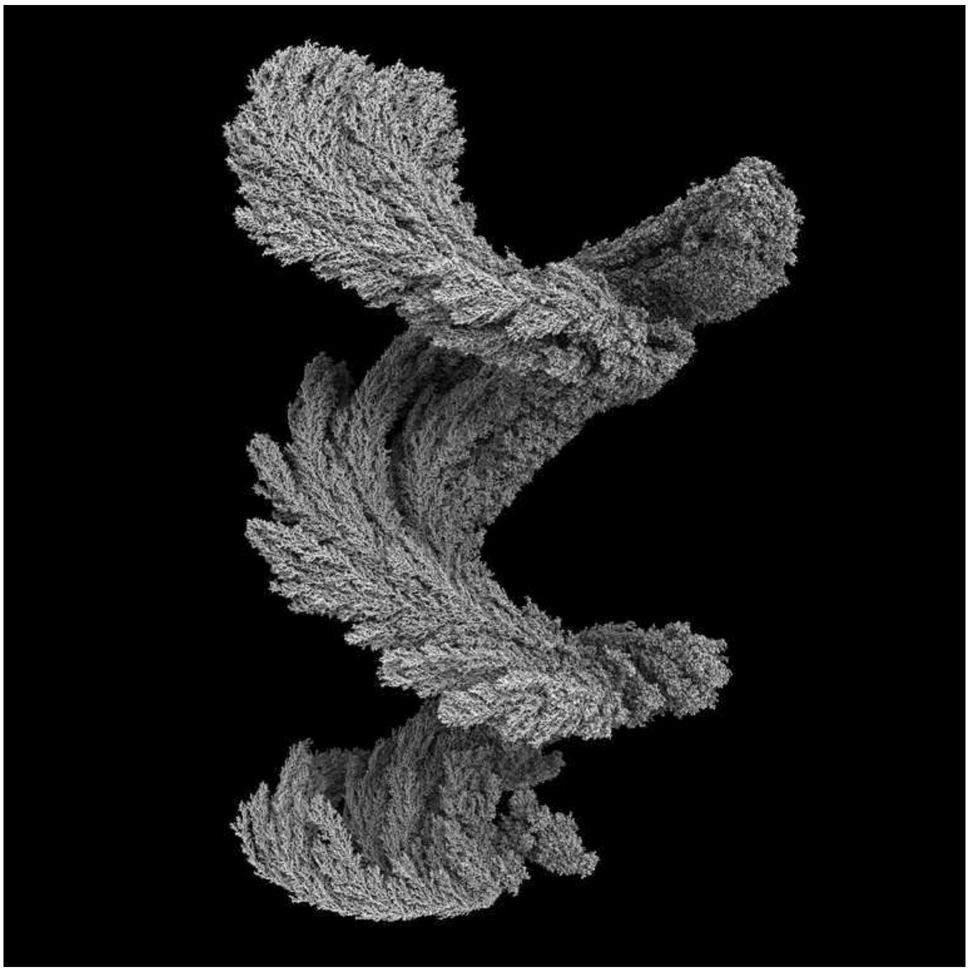

Figure 2: An example of a structure sculpted by Andy Lomas using the aggregation process. [5]

3.0 Technical Background

3.1 Diffusion-limited aggregation

A simple diffusion-limited aggregation simulation takes place with a seed particle normally placed at the origin of a lattice, this known as the aggregate. This is the start of the aggregation structure**.**

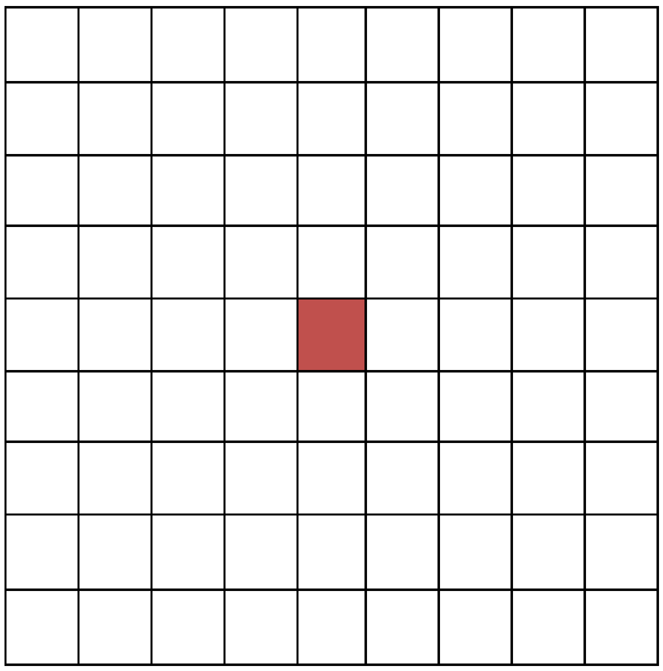

Figure 3: Seed particle, this is the beginnings of the aggregate.

Another particle is birthed at a random position at the edge of the lattice. This particle can now undergo a random walk on the lattice. The random walk is used to replicate the random motion described by the Brownian motion model.[6] Brownian motion is the model proposed by Robert Brown in 1827 which describes the apparent random motion of particles that are suspended in a fluid. This is where the “diffusion” term comes from.

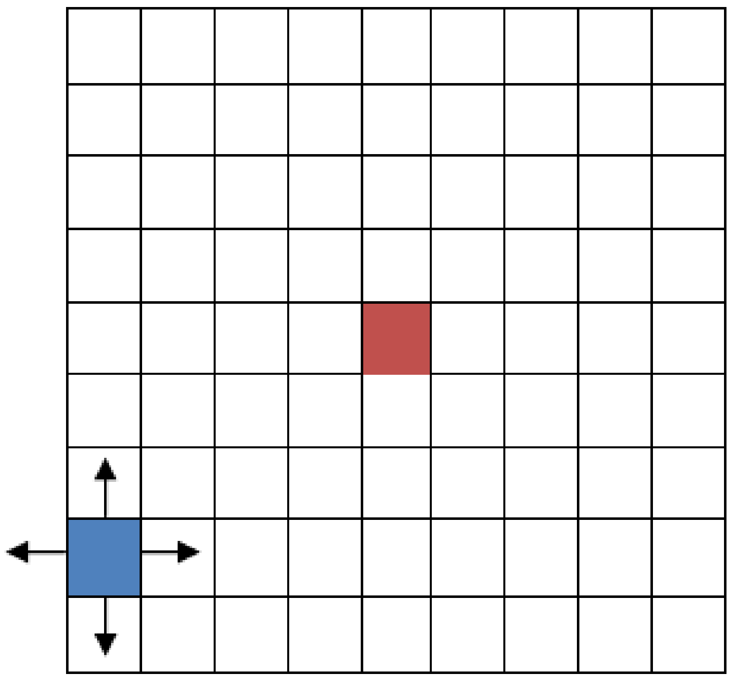

Figure 4: Another particle is birthed at a random position on the edge of the lattice. This particle will now undertake a random walk

As the particle moves around in this random motion it will eventually come into contact with the seed particle. When this happens it will stick to the seed particle and be added to the aggregate.

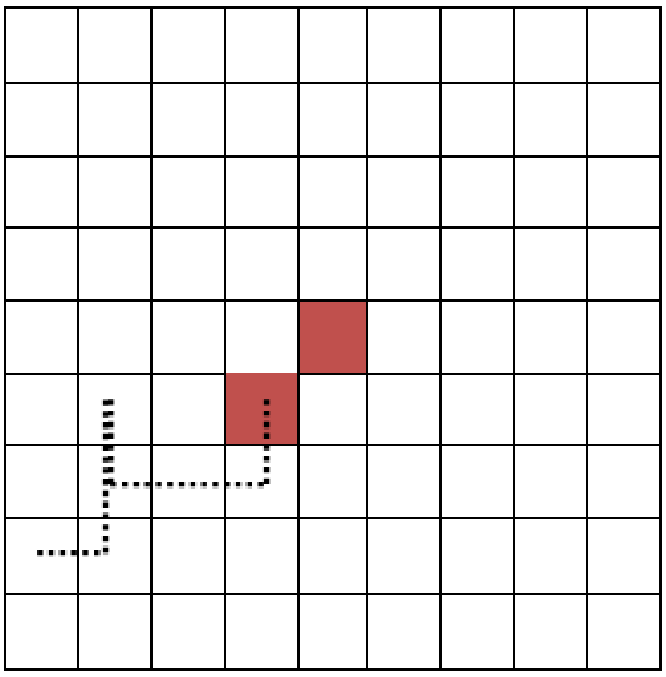

Figure 5: The particle will undergo a random walk. If the particle comes into contact with the seed particle it will stick to it and become part of the now growing structure.

It is considered “diffusion limited” because the particles are considered to be in a low concentration, therefore they are considered not to come into contact with one another but instead only with the structure. This means the structure grows one particle at a time rather than in large sections [7].

When left to accumulate the aggregated particles can produce very complex structures that exhibit extremely high levels of detail. The structures created are examples of fractal geometry. Figure 6 is an example of just a single seed point that was initialised in the centre of a grid with aggregation left to accumulate onto it.

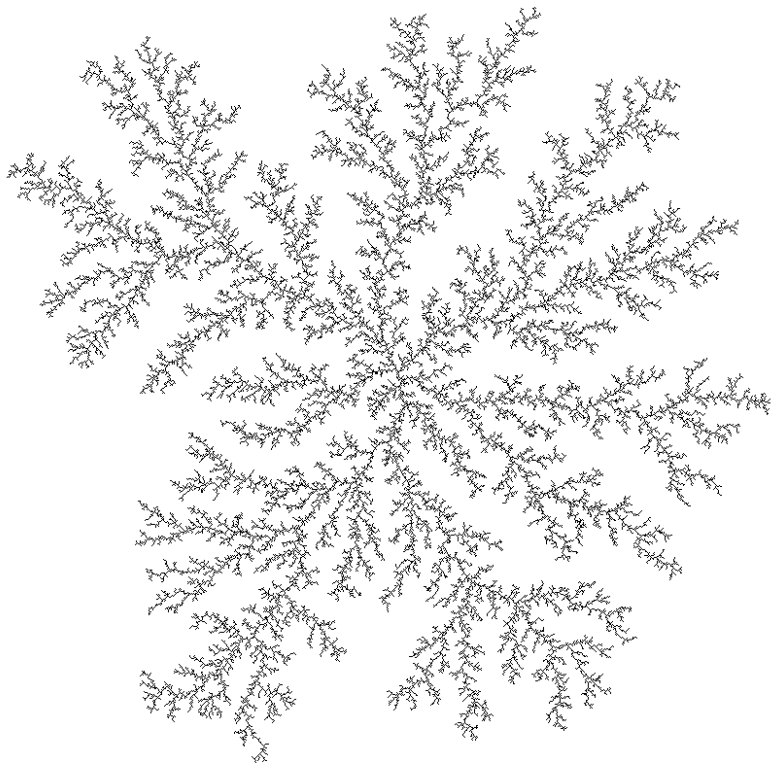

Figure 6: Aggregation that has accumulated on a point attractor at the centre. [7]

4.0 Implementation

4.1 Simulation Process

As the aim of the project is to generate fractal patterns over a 3D surface, using an algorithm based on the principles of diffusion-limited aggregation is a logical choice. Although DLA is traditionally simulated on a flat plane it should be possible to translate the general algorithm to a 3D surface.

A traditional DLA algorithm can be broken down into the following steps:

- An initial particle or set of particles are created from which any further growth will start. This is known as the ‘aggregate’.

- A particle is then placed at a random location on the surface of the plane and undergoes a random walk. These are known as ‘walkers’
- If a walker collides with the aggregate it will stick and become a part of the aggregate.
- A new walker is born and undergoes a random walk.
- This process is repeated until the aggregate has grown to the desired size.

Replicating this algorithm on a 3D surface using Houdini would allow the user to create particles arranged in a fractal pattern over a 3D surface. Using these points to generate geometry or affect shaders on the object would then give an interesting and visual outcome to the algorithm.

4.2 Creating the aggregate

It is important to make the tool as user friendly and interactive as possible, with a simple and intuitive way to create the aggregate. Placing particles onto the surfaces as the initial starting aggregate would have been the most obvious method if following the traditional DLA algorithm verbatim. However this would have been less intuitive and slower for a user wishing to set up a simulation. Instead it was decided that Houdini’s Paint SOP would be implemented for this purpose. If you consider that Houdini describes geometry with the notion of points, edges and primitives, two points are joined together by an edge and three or more edges combine to create a polygon shape known as a primitive. The Paint SOP allows you to manipulate, or paint on, colour attributes to the points on your geometry with a brush tool. This allows the user of the tool to simply paint on a region he or she wishes to be the aggregate. This provides a simple and very visual way to choose where the fractal growth will start from.

4.3 Simulating the random walk

A random walk simulation on the surface of a flat 2-dimensional plane is a simple process, at each step the walker is given an equal probability of walking either forward, back, left or right. If the walker reaches the boundaries of the plane it can be terminated and a new walk started.

Attempting to simulate a random walk over a 3-dimensional surface is a much more challenging problem. Keeping a walker on the surface of a complex piece of geometry while at the same time allowing it to take a random step in either of the x, y or z axes is difficult. At any position on the surface of the geometry there is the potential that a step in a particular direction could take the walker away from the surface. If this was allowed to continue then very quickly the particles would leave the surface and be scattered randomly in space around the piece of geometry. So the challenge is to allow the particles to freely move in any direction but at the same time keep them on the geometry’s surface.

Houdini has a powerful and highly adaptable set of particle operators, POPs, built into the software. One of the operators, the Collision POP, allows particles to be set to collide with a surface level operator, SOP. This means a set of particles can be set to collide with any piece of geometry within the network and gives a number of options known as behaviours as how to react upon collision with the allocated SOP. One of these behaviours is called slide, this option means the particle will slide along the surface of the collision object sticking to its surface before eventually falling off.

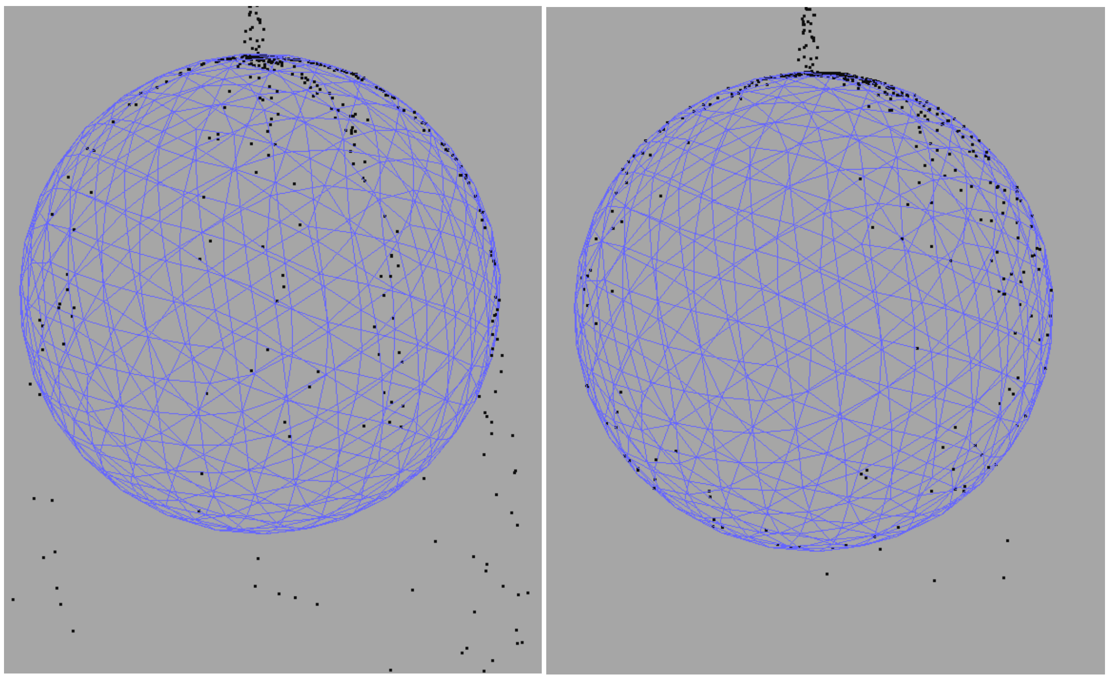

Figure 7: One the left without the use of the Property POP particles fall from the surface. On the right with use of the Property POP particles stick to the surface more but still some fall off.

One of the problems with the slide option from the Collision POP is that the particles only have a small resistance to leaving the surface they are sliding along. As the particle travels along the changing topology of the surface any sharp changes in the direction will cause it to slide off the surface and continue its path away from the geometry. This problem can be reduced by using the Property POP, which allows the Cling attribute of the particle to be manipulated and therefore increased. Increasing this attribute makes the particle more resistant to leaving the surface of the object it is sliding along. However this only reduces the likelihood that the particle will leave the surface, if the particle were to experience a greater a force at any moment then it would no longer stay on the surface.

Since Houdini contains no in built operators that will allow particles to stay on the surface of an object in a satisfactory manner then two custom VOP POP networks were created instead. Firstly in order for the networks to work a signed distance field based on the geometry is created. This is done using the Isooffset SOP, this operator allows volumes of different types to be created, for this purpose an SDF Volume is chosen. A signed distance field is a technique in mathematics for determining how far away a point is from the boundary of a set. Points inside the set have positive values, whereas points outside the set have negative values. If we consider the set to be our geometry and the point to be our particles, then the SDF will give us not only the distance from the surface of the geometry to the particle but also whether the particle is inside or outside of the geometry [8].

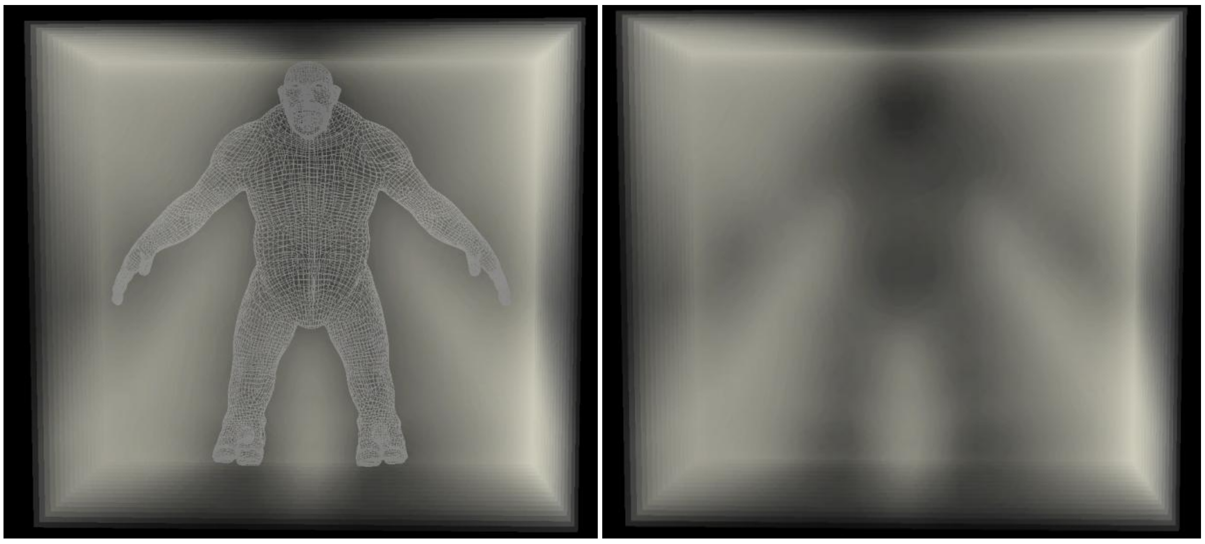
Figure 8: An example of an SDF volume that has been generated for a troll model.

Two VOP POP networks are used to keep the particles on the surface of the geometry, each using a slightly different SDF volume. One network reacts to particles that are inside the geometry and uses an SDF volume that considers the geometry to be its real size without altering it. The other network is for particles that are outside of the geometry, and uses an SDF which creates a slight offset around it. The role of this offset is to create a thin layer surrounding the geometry which the particles are to be confined to.

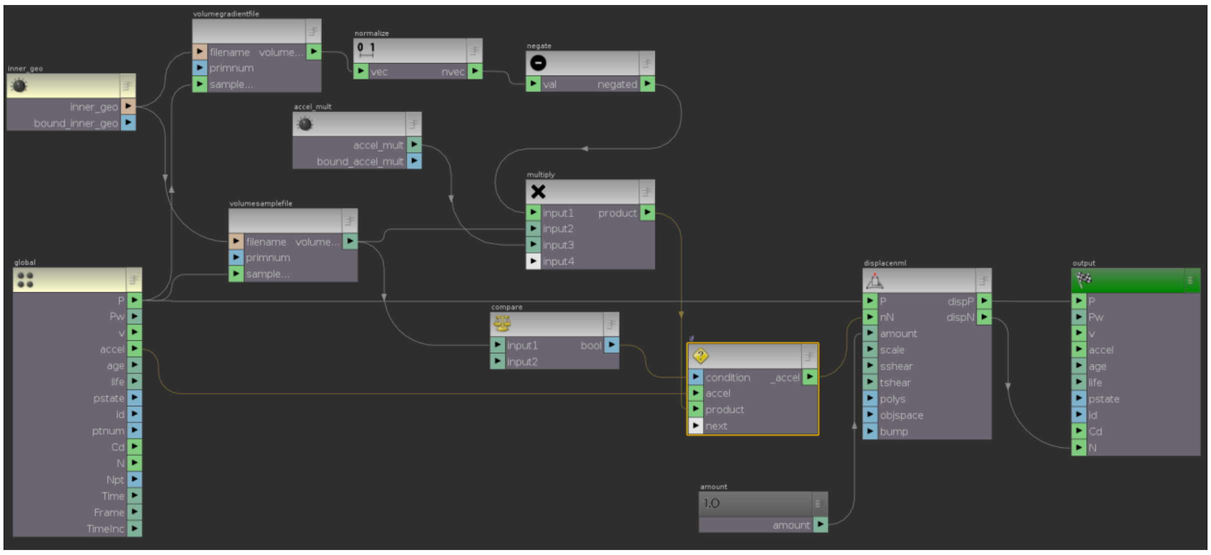

Figure 9: The VOP POP network that is used to stop the particles from entering the geometry

If we consider the network(figure 9) which deals with particles that enter the geometry, reading the network from left to right the 'Volume gradient from file' node takes in the particles position, P and calculates the direction to the nearest point on the surface of the geometry. The 'Volume sample from file' node samples the SDF for a positive or negative value letting it know whether the particle is inside or outside of the volume. Using a Compare node to compare the value returned from the Volume sample from file node against 0. This can then return a boolean true or false value as to whether the value is greater than 0, and hence if the particle is inside or outside of the volume. With this information the If node can act, if the particle is inside the volume then it can multiply the direction to the nearest point on the surface by the distance to this point. This vector can then be used to move the particle with the 'Displace along normal' node. The result is that the particle has now had its position changed from inside the geometry to the surface of the geometry.

With the other network moving particles from the outside of the geometry onto the surface of the slightly offset layer surrounding the geometry, then with the two networks combined the particles are confined to staying within a thin layer that surrounds the object.

With the two VOP POP networks keeping the particles on the surface of the geometry the particles can now undergo a random walk. A Force POP is appended to the network, this operator allows forces to be applied to the particles. The role of this operator is to simulate a random walk in three dimensions for the particles. This is done with the use of an expression that is used in the x, y and z components of the Force POP. The expression uses the `fit` function as follows:

$$fit01(rand(\$PT * \$FF), -1,1)$$

If this expression is broken down and we isolate the `rand` function we see that it will create a random number between 0 and 1. This uses the variables $PT and $FF as its seed. $PT is the point number of the particle, since each particle has a unique point number then this will ensure that the random number generated is unique for each particle. $FF is the floating frame number, using this ensures that the seed for every particle is different on a per frame basis. With these two variables combined it ensures that each particle generates a random number between 0 and 1 that is unique for every particle and different at every frame.

The role of the `fit01` function is to take a number that ranges between 0 and 1 and fit it to be between a new range. In this case the new range is set to be between -1 and 1. The effect of the expressions when combined together is that a unique random number that changes on a per frame basis is generated for every particle. This number will range between -1 and 1. If this number is now considered to be a force acting on the particle in one of the axis then this works as a good approximation to a traditional random walk.

To add more variation each axis has the seed $PT and $FF multiplied by a different number, this ensures that the x, y and z axis each receive a different seed and hence a unique force is generated in each axis.

If every axis uses a random number that has been fitted between -1 and 1 then the particles should be moving in a completely random motion. No direction is given a preference and so the motion of the particles should be following a random walk as described by the traditional DLA algorithm. However giving the particles a greater probability of moving in one direction might actually be desirable sometimes. For example if a user wishes some growth to start at the feet of a character and grow upwards then the growth might look better if it's growing more vertically rather than truly randomly. To do this the tool is set up to allow the user to give preference to a particular direction. If we consider the example of growth starting at the feet of a character we would like the particles to favour moving downwards in the negative y direction rather than the positive. The expression therefore for the force acting on the particles in the y-direction might be:

$$fit01(rand(\$PT * \$FF * 157.5), -1,0.9)$$

Here the force is being fitted between -1 and 0.9, this gives the particles a higher probability of moving in the negative direction. The number 157.5 is an example of multiplying the seed by a unique number to ensure each axis generates a unique force.

4.4 Limitations of SDF Volumes

For the majority of cases generating an SDF volume as a way to keep the particles on the surface of the geometry works very well, however there are some limitations to this method. Geometry with smaller, more intricate details makes it difficult to generate a good SDF volume. With the example of the troll model in figure 8 areas such as fingers and the mouth have finer detail. Trying to generate a satisfactory volume from these parts becomes problematic. The Isooffset SOP which creates the SDF volume allows the resolution of the volume to be increased; this is an option which the tool passes onto the user. Increasing the resolution helps to decrease the problem and for the examples generated for this project produce more than satisfactory results. However if geometry that was more complex was used no matter how high the resolution was set the outcome would never reach a satisfactory level. Without being able to generate a good enough SDF volume the particles would be allowed to enter geometry creating growth that would go inside the geometry.

4.5 Collision Detection

If collisions are broken up into two areas, collisions which take place between walkers and the initial (painted on) aggregate are considered to be primary collisions. Then collisions which take place between walkers and particles which have become a part of the aggregate are secondary collisions.

4.6 Primary Collisions

Areas that have been painted with the Paint SOP to be defined as the initial aggregate area are different from the rest of the surface. Points which make up the aggregate carry a colour attribute with them and this can be used to detect whether a walker has collided with the initial aggregate.

In Houdini the initial aggregate area is considered to be at surface operator level , SOPs, whereas the particles which act as our walkers are confined to particle operator level, POPs. In order for the particles to be aware of the information at SOP level this needs to be imported into the POP network. This can be done with an `Attribute transfer` node. This node allows attributes like colour to be transferred from a SOP level node to a particle if the particle is within a certain distance. Setting this distance to be very large means that the particles can be far away from the aggregate and they would still have the colour attribute transferred to them. If the distance is very small then the particles need to get very close to the aggregate to receive the attribute. For the purposes of this project a small distance value is needed so the particles collide accurately with the aggregate.

Once the particles have had the colour attribute transferred to them then they can be grouped as a separate group of particles. These can then be stopped and become a part of the aggregate.

Initially it was hoped that the system would be able to use textures to create the initial aggregate instead of the paint tool. For this to work the particles would need to know the UV coordinate of their current position. This would enable them to look up the colour of the texture they are on to see if they have collided. Particles were given UV coordinates by using the Attribute Transfer POP, this transferred a weighted average of the UV coordinate from the four closest points on the geometry to the particle. However using this method creates a problem because geometry often has UV coordinates applied in sections, the arms on a human are often given UVs separately to the body and legs for example. This means that where the two sections meet a point on the arm could have a vastly different UV coordinate to that of a point on the chest. If a particle is on the boundary between the chest and arm of the model and it takes a weighted average from its four closest points to generate its UV value then this will generate a result that is not a true representation of its current position. The effect of this is particles could be far away from where the initial aggregate has been painted on the texture map and they would still think they have collided with the aggregate. Without being able to research a satisfactory way of giving particles a UV coordinate based on their world position it meant using textures to create the aggregate was not possible. However the Paint SOP approach works well and is very fast and easy for the user to update and so is probably a better alternative than the texture method.

4.7 Secondary Collisions

As the simulation evolves over time the aggregate will grow as more walkers collide with the existing aggregate. Secondary collisions are considered to be when a walker collides with a particle that has joined the aggregate. In order for this to happen a feedback system needs to be created. Particles which were originally walkers and have collided with the initial aggregate need to become a part of it so that new walkers can come and collide with them. These in turn need to become a part of the aggregate and so on while the simulation takes place.

Houdini offers a couple of solutions to creating a feedback system. One is to use the SOP solver which is contained inside the DOP network. This provides a way of creating a feedback system, but experience has shown this method to often be unstable and to frequently return errors. For this reason a more stable feedback system was created by making a custom system using two File SOPs. At the end of the network which simulates the particle, all particles which haven't collided through primary or secondary collisions are deleted. This leaves just the particles which will be a part of the new aggregate. These are then written to file on a per frame basis using the first File SOP. Meanwhile the second File SOP is reading in the previous frames set of aggregate particles.

With the second File SOP reading in the aggregate particles this creates a target for the walker particles to perform secondary collisions with. The second File SOP has created a surface level operator that is nothing more than a series on points in space. Houdini provides no in built nodes that allow particles to collide with just points so instead a few attributes were created for each particle. Firstly each walker particle finds which point from the File SOP is closest to itself. To do this the hscript function `nearpoint` is used, this function returns the nearest point in a surface node's geometry to a given point. If the surface node is the File SOP and the point in space is the current position of the walker, then this function will return the value required. This point number is then stored as an attribute called $NEARPT.

Now the walker needs to calculate the distance between itself and the nearest point in the aggregate. This is done using the hscript function `distance`, which returns the distance between two 3D points. The walker already knows its current position but it does not know the position of the closest point in the aggregate that is stored in the File SOP. The walker does however know the particle number of the nearest point as this is stored as the attribute $NEARPT and can use this to query the position of it. This is done using the `point` function, which can be used to return any attribute a point may contain including its position. With the position of the nearest particle now known the `distance` function can be used to calculate the distance between these two points.

The distance is stored as an attribute called $PTDIST, which can be checked against a threshold distance set by the user. If $PTDIST is less than the distance specified then the particle can be considered to have collided with the point. The particle is then stopped and grouped ready to become a part of the aggregate.

As described previously at the end of the frame any particles that are considered to have collided either through primary or secondary collisions are written to file. This file will then be read in at the next frame as part of the new aggregate. Walker particles can then update their $NEARPT value and then $PTDIST, allowing the secondary collision process to continue in the necessary feedback loop.

4.8 Birthing walker particles

The traditional DLA algorithm births a new walker particle at a random position on the surface of the 2D plane. The position of where it's birthed can be completely random as long as it isn't on top of existing aggregate. To mimic this in a 3D geometry context using Houdini a Scatter SOP was used. Particles should not be emitted from the aggregate region since this would mean they would instantaneously collide with it, but any other region is allowed to have particles birthed on it. Since the user of the tool has painted on the initial aggregate region, when the geometry comes into the tool any surface that has been painted is disregarded, this leaves only the surface that is considered good for emission. Several hundred points are scattered randomly onto this surface with use of the Scatter SOP. These points are the points that are allowed to birth walkers.

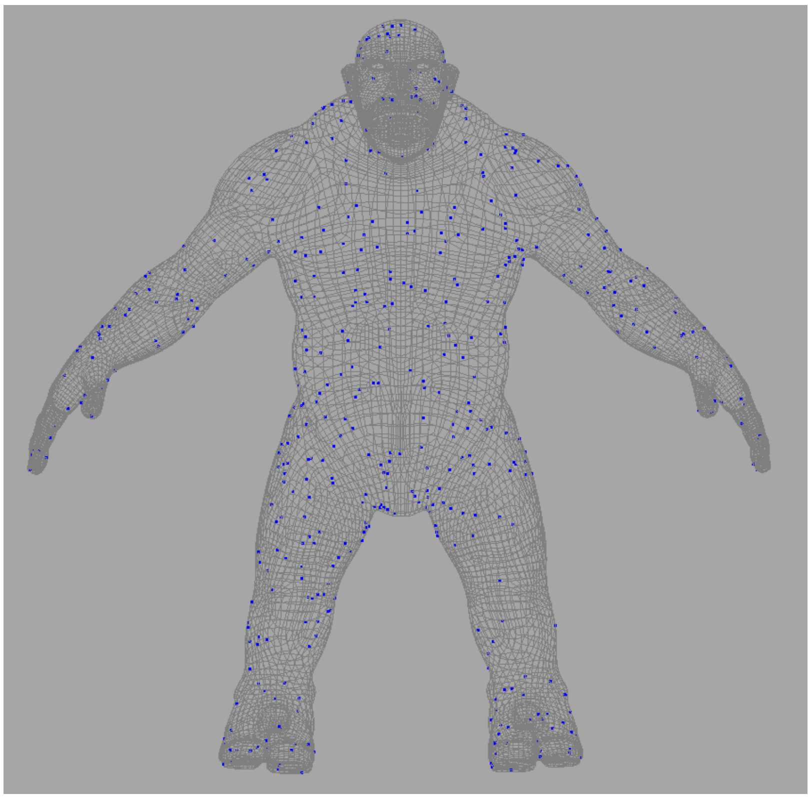

Figure 10: Using the Scatter SOP to randomly place points over the surface of the geometry. From these points walker particles can be birthed.

As the aggregate grows in size the scattered points will become covered by the aggregate. If they were to still emit particles at this stage then the walkers would again instantaneously collide with the aggregate. This would cause many problems for the growth as thousands of particles would become concentrated into small regions. To stop this from happening the scattered points are made aware of how far away they are from the aggregate.

In a similar way to how walkers detect their distance to the nearest point in the aggregate so too do the scattered points. They find the nearest point in the aggregate to themselves and calculate the distance between the two. After several tests it was found that if the distance between the scattered point and the aggregate was less than four times the distance threshold that is used for collisions, this would yield unsatisfactory results.

4.9 Joining the points together

The process of simulating the particles creates a bgeo file full of several thousand points that have been aggregated over the chosen geometry. In order to use these particles in a useful way they need to be joined together with lines so that they can then be manipulated further.

As previously mentioned during the simulation process the particles created two $NEARPT and $PTDIST. Using these two attributes it is then possible to create lines which will connect each particle to the particle it collided with. Houdini has a node called the Copy SOP which can copy geometry on to a point or set of points. One useful feature of the Copy SOP is that when the geometry is copied onto the point it will orientate itself in the direction point's normal. The consequence of this is that if the normal of the point was the vector that described the direction between itself and the point it collided with then the geometry copied onto that point would be aligned to be pointing towards the collided point. The points can have their normals altered using an Attribute Create SOP. Calculating the vector between the point and the particle it collided with is possible since it has the attribute $NEARPT stored. This can be fed to the `point` function to find the position of the point it collided with. Finding the vector between it and the collided point then becomes a simple task. This vector can then be normalized and used as the point's normal.

Another useful feature of the Copy SOP is the `stamp` function which allows the geometry that is being copied onto the point(s) to be varied or altered for each copy that is made. This allows great variety to be achieved as virtually any attribute of the geometry can be altered, from size, orientation to colour.

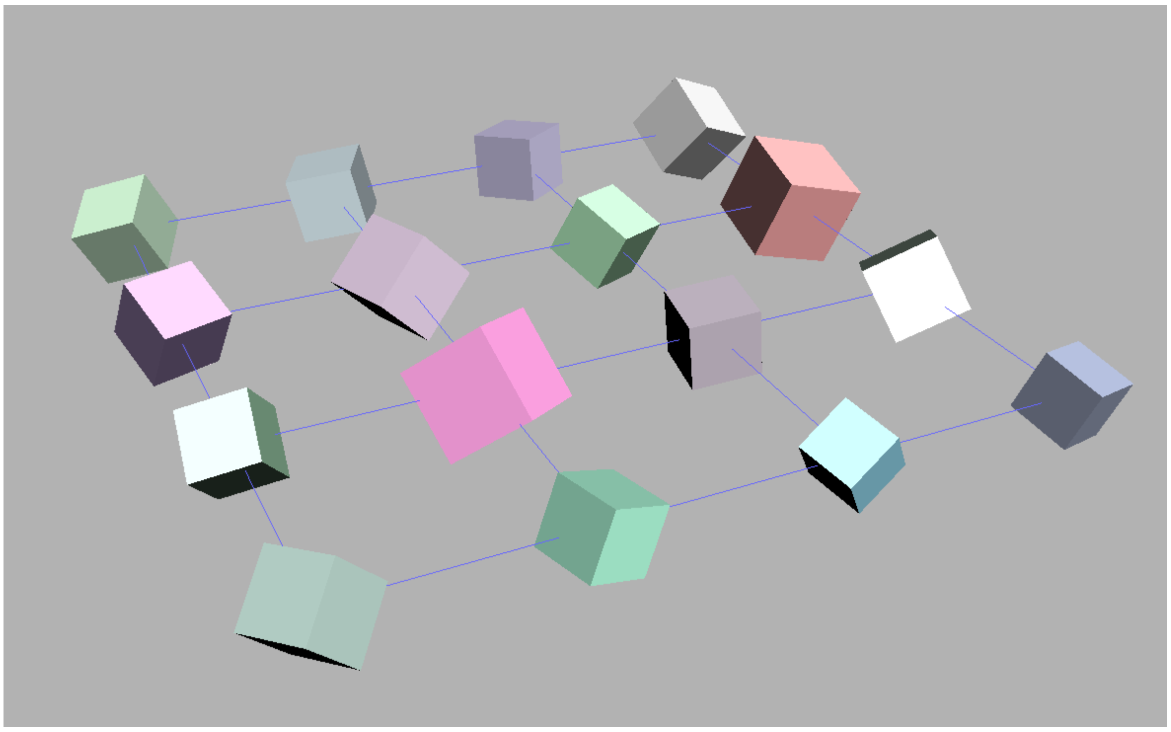

Figure 11: Using the Copy SOP boxes have been copied onto a grid of points. The `stamp` function allows variation in size, orientation and colour to easily be established.

To connect the simulated particles together straight lines were used as the geometry that would be copied by the Copy SOP. With the normals now set on the points to point in the correct direction, the lines just need to be of the correct length to connect the points together. The attribute $PTDIST stores the distance between the point and the point it collided with. Using this in the `stamp` function to vary the length of the line that is being copied ensure that the line will be the correct distance between the points.

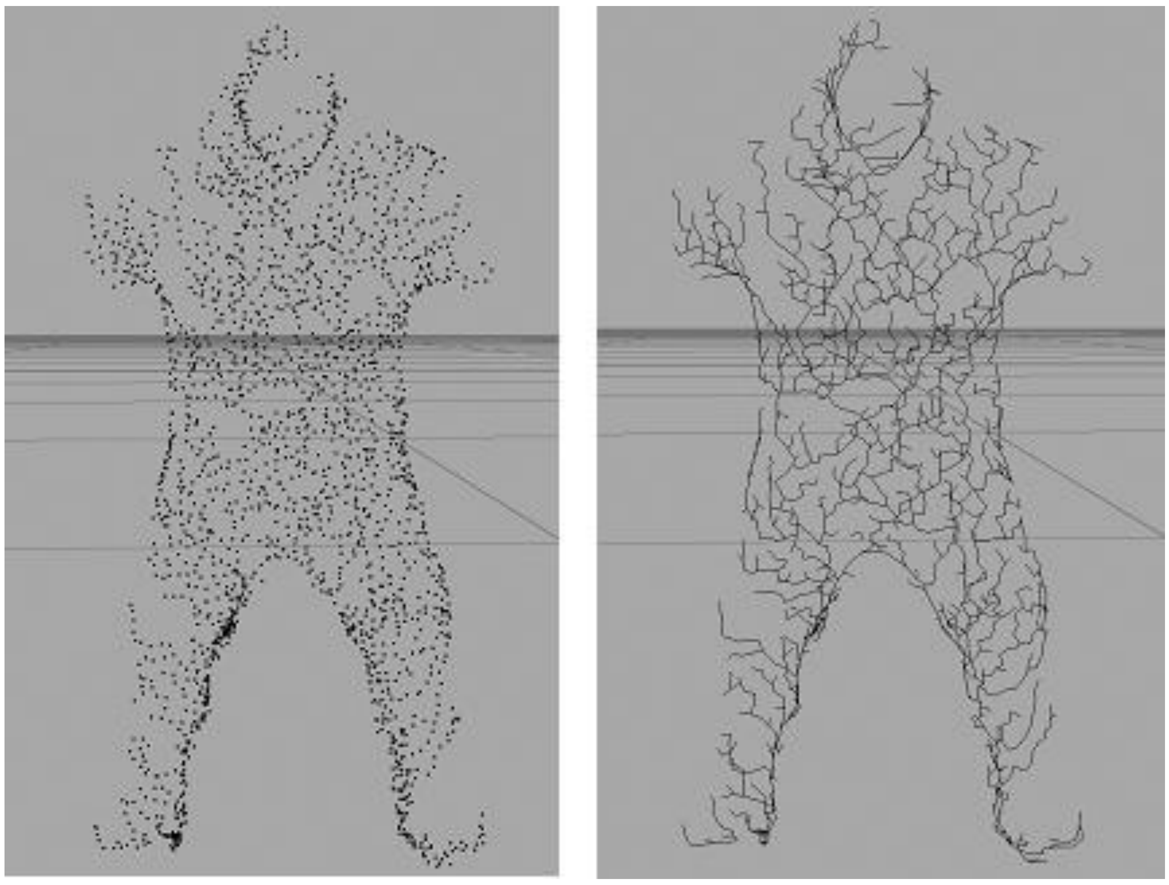

Figure 12: Simulated points are now joined together by lines by use of the Copy SOP

With the points now converted into lines these can now be used to generate interesting geometry that can be used to create useful effects. Using the Polywire SOP the lines can be converted into 3D geometry.

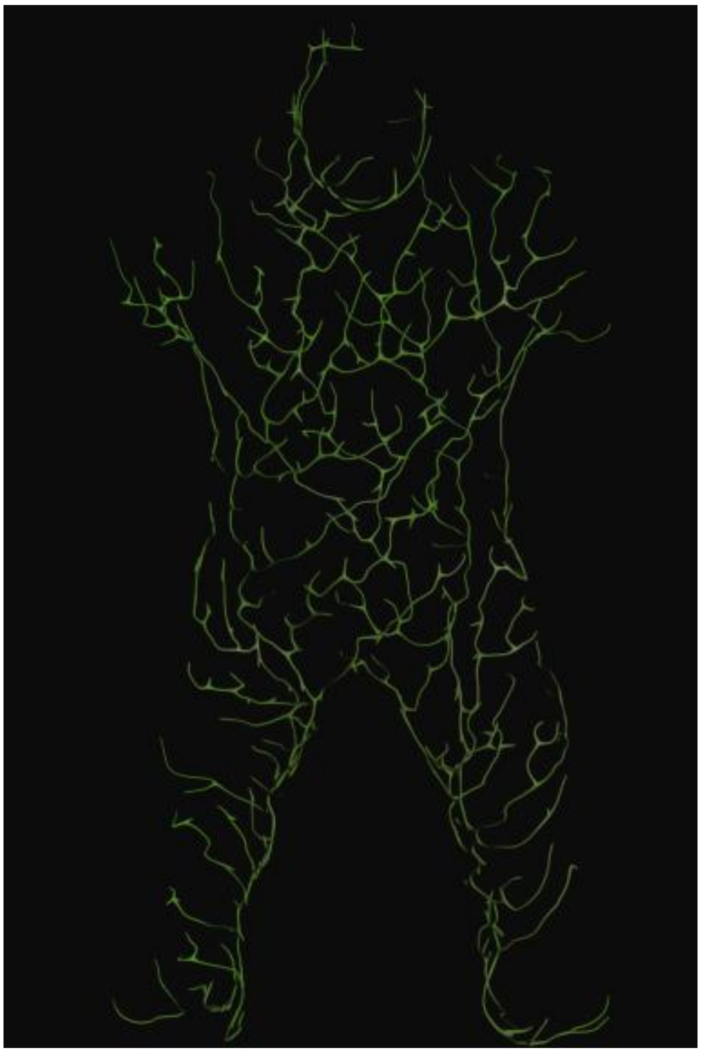

Figure 13: Using the Polywire SOP to give the line geometry

5.0 Animating the growth

Giving the user a frame range over which the growth can be animated is a useful and important option for the tool to provide. This is achieved by manipulating the $LIFE attribute of the particles. When the particles collided with either primary or secondary collisions their $LIFE attribute was set to zero. After every frame this value increases, and the effect of this is that by the end of the simulation the particles that collided early on will have a large life value whereas particles that collided later will have a lower value. When the simulation is finished these values act as a time line showing the order in which the collisions took place. The Sort SOP has the ability to sort points in order based on a chosen value. Setting this value to be $LIFE allows the network to sort the points in descending order based on life.

The lowest life value will always be 0 and with the points now sorted the largest $LIFE value can also be easily determined. The $LIFE attribute of all points can now be fitted to a new range. The `fit` function can do this, the `fit` function takes in a value which was between an old range and returns a relative value between a new range. In this case the old range is between 0 and the largest $LIFE value found from the Sort SOP. The new range can be set to range between 0 and 1, this new value can now be stored on the point as an attribute called $ANIMATE.

The current frame number can now be fitted to be between 0 and 1 also. In this `fit` function the old range for the frame number can be the frame range set by the user, the new range is 0 to 1. For example if the user set the frame range to be between 0 and 500, and the current frame was at 250 the new fitted value will 0.5, and if the current frame was 500 then the new fitted value would be 1.

Now to animate the growth the attribute $ANIMATE can be compared to the fitted frame value. If $ANIMATE is less than this value then it should be visible, if it is greater it's not ready to be seen and should be deleted. The effect of this means the user can easily choose a frame range and the growth will be fitted within that frame range.

5.1 Adding and animating additional geometry

Adding additional geometry to grow from the originally simulated points is an important way of adding interesting effects to the tool. This project concentrated on growing plant like foliage points but really anything could be used depending on the effect needed.

Any geometry added to the system needs to be birthed and grow at the correct time. Leaves or branches need to start growing after the initial vine has reached them, if they were to start growing earlier then they would appear to be growing by themselves unconnected to the plant.

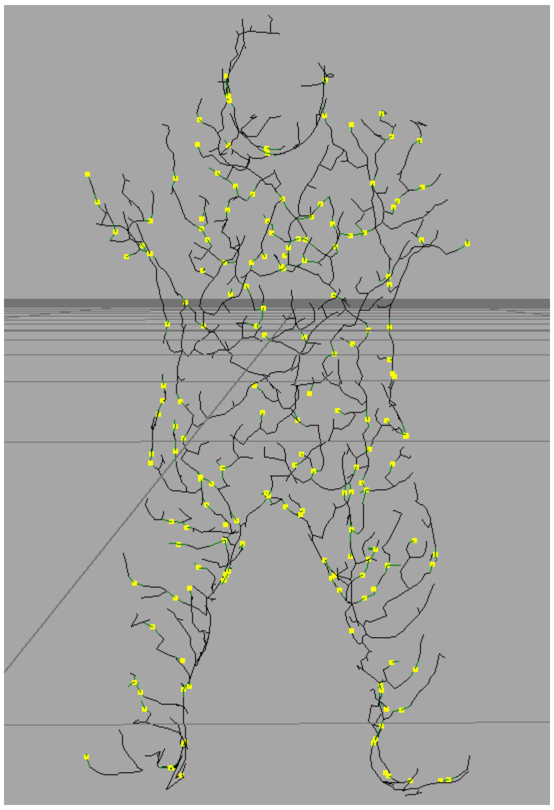

Figure 14: Points that have been selected to have branches grow from them. The number of points selected is a value controllable by the user of the tool.

To achieve this firstly a random set of points are chosen from which the geometry will grow, as shown in figure 14. The number of points is an option that is easily adjusted by the user.

Using the Copy SOP the piece of geometry is copied onto these points. The geometry used can be separated into two categories, leaves and flowers being one, and branches or spiralling tendrils the other.

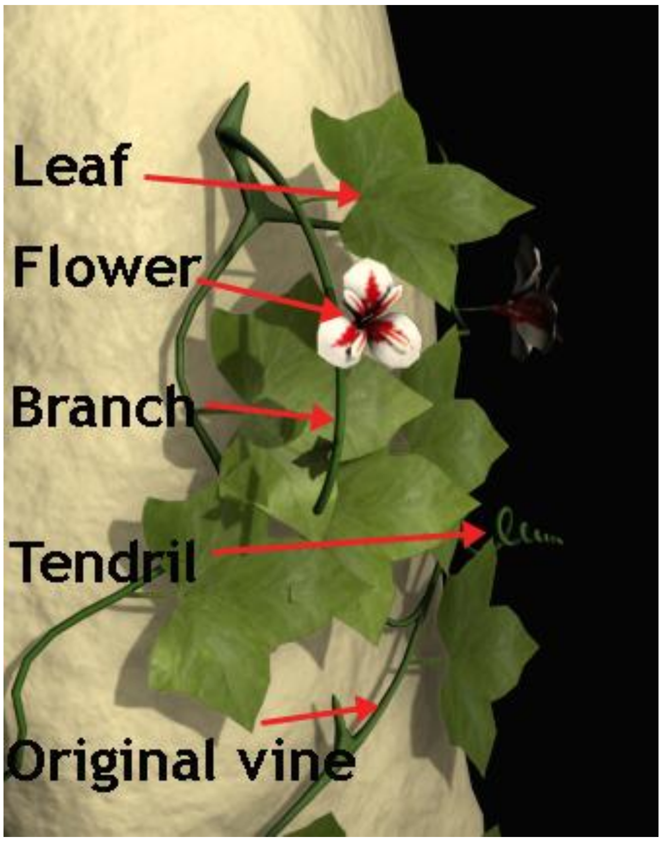

Figure 15: Showing the different types of geometry that make up the plant.

5.2 Leaves & Flowers

When a leaf or flower is copied onto the selected point, the Copy SOP uses the `stamp` function to give the geometry the $ANIMATE value from the point it is copying onto. The leaf or flower can then use this value to animate its own growth using the scale option in the Transform SOP. In the scale field of the Transform SOP a `fit` function is used, this function’s action means that the leaf or flower is scaled to 0 when the growth of the vine hasn't reached the leaf/flower but then increases to 1 once the vine reaches it. This is done by comparing the $ANIMATE value with the frame number that has been fitted between 0 and 1(as describe in section 5.0). When the $ANIMATE value is equal to the fitted frame value the leaf or flower can now start to scale its size from 0 to 1, scaling the size over a time period that can be specified by the user acts to mimic a growth effect.

5.3 Branches & Spiralling tendrils

When a branch or spiralling tendril is copied to the growth it is done in a way that means it becomes a part of the original vine that is growing on the geometry. Since the original lines will be converted to geometry with use of the Polywire SOP it is important that the branches or tendrils are converted by the Polywire SOP at the same time. This creates a seamless connection between the different pieces of geometry and gives the appearance that they really are joined together as if they were one plant.

The Copy SOP uses the `stamp` function to give the branch or tendril the $ANIMATE value from the point they are being copied to. The point of the geometry (either the branch or tendril) that is closest to the vine is given an exactly equal $ANIMATE value, each point further down the branch or tendril is given a value that is incrementally larger. By doing this, when the growth is animated using the fitted frame number (see section 5.0) the branch or tendril will become animated along with the original vine. The speed at which this extra geometry grows is controlled by the size of the incremented $ANIMATE value assigned to each point on the geometry, the size of this value is controllable by the user of the tool meaning they have full control over the speed at which their plant grows.

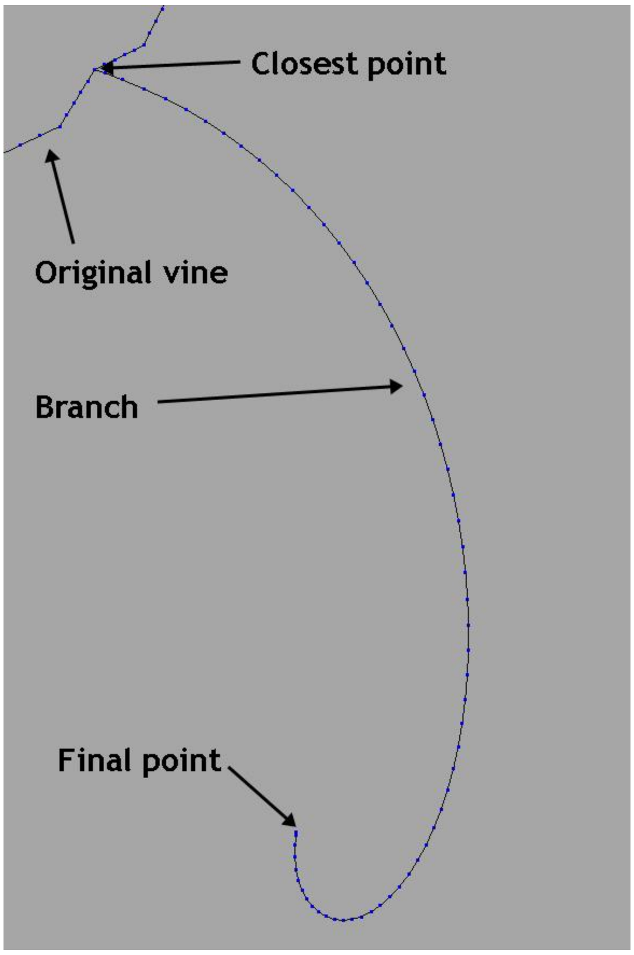

Figure 16: The closest point to the vine receives a value equal to $ANIMATE, every point afterwards receives an incrementally larger value with the final point receiving the largest.

6.0 Conversion to texture space

Instead of converting the lines generated into plant geometry they can also be converted to texture space which would give the user more options to create different effects. The aim of this part of the tool is to allow the user to use the simulated fractal lines as a mask for a texture or displacement map.

The tool requires the geometry's points to have UV co-ordinates; this is normally the case for most objects in a 3D environment so it is reasonable that the tool should work this way. With the fractal lines now covering the surface of the geometry the Attribute Transfer SOP can be used. This node allows attributes to be passed from one SOP to another. This is useful as it allows the UV coordinate information from the geometry to be transferred to the points which make up the lines. The Attribute Transfer SOP gives several options as to how this information is transferred. By default the node would pass the UV coordinate from whichever point is closest on the geometry to each point that makes up the line. However this would lead to points receiving values that weren't true to their position on the surface. Instead it would be better if the points on the line took a weighted average of the 4 closest points on the geometry it is resting on. In this way it will receive a UV coordinate that is a much more accurate reflection of its position on the surface.

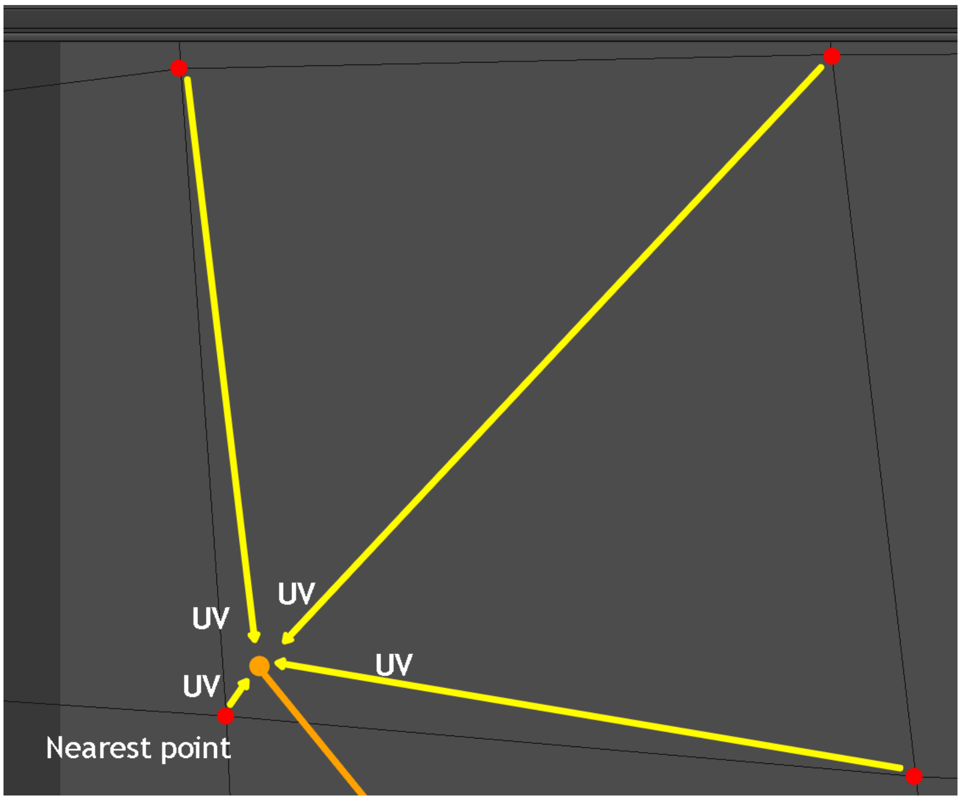

Figure 17: UV information is passed from the geometries points in red to the points that make up the fractal lines in orange. If the UV is transferred only from the nearest point then this is not a true reflection of the points position on the surface. However if the point is given a weighted average of the 4 closest points this provides a much more accurate value.

Now that each point that makes up the lines has a UV coordinate they can have their world space coordinate set to be equal to their UV coordinate. The effect of this is that the lines will be orientated in a square 2D plane with values ranging 0 to 1. However, since the UVs can be spread out in UV's space in any arrangement the creator wishes, this means that for example the UVs of a human shoulder could be separated far away from the chest. The consequence of this is that when the lines are converted to UV space some will become very stretched as shown in figure 18.

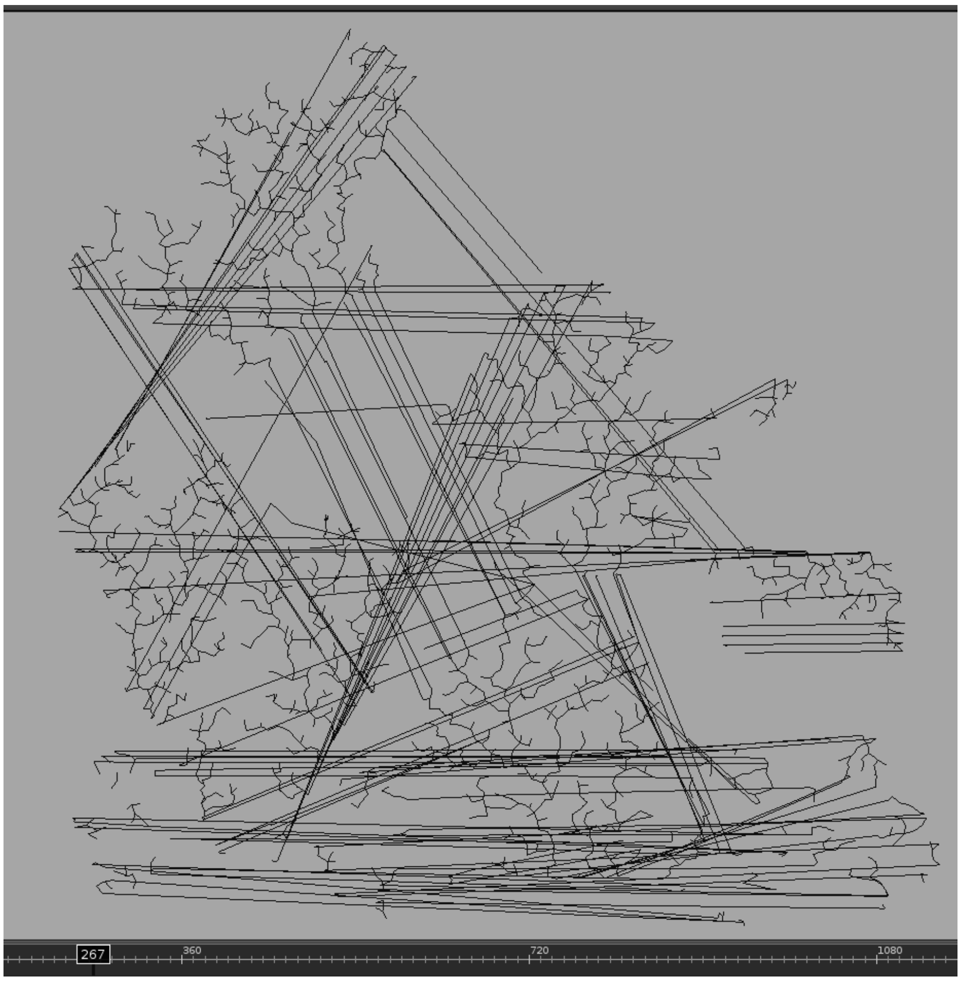

Figure 18: When initially converting into texture space lines that are placed on the border of different UV sections will become stretched. These will be much longer than the other lines and a simple filter based on the length removes them.

To overcome this problem the length of each line is checked and if it exceeds a set threshold then it can be deleted leaving only lines of the correct length. Now that the lines are positioned in a way that can be used for textures they are converted into tubes using the Wireframe SOP, which gives the lines a width.

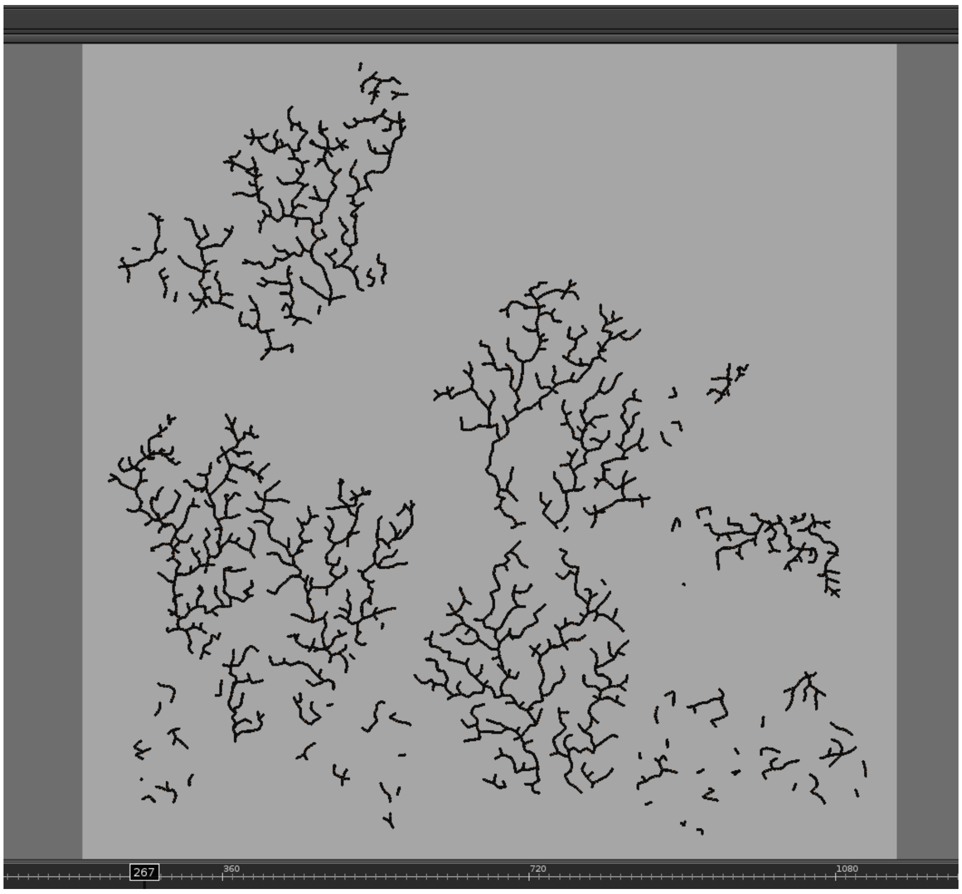

Figure 19: After removing the longer lines the remaining ones can be given width by using the Wireframe SOP. This can now rendered with an orthographic camera to create a texture or displacement map.

They can now be rendered using an orthographic camera. Rendering them out has created a texture or displacement mask. This mask can be used for a variety of effects which have been demonstrated. Due to the random and fractal shape of the lines that are produced they could be used for numerous purposes. This project shows them being used for a lava effect, rust growth and veins.

7.0 Compositing and Rendering

7.1 Plant Growth

To show the adaptable ability of the tool it was decided to render sequences of the vine plant growing over three differently shaped objects; a troll, teapot and chair. These objects all have different and interesting topologies, and are similar to typical objects that might be used in a full production.

Several passes were used and composited to create the final look of the sequence. These were diffuse, specular, ambient occlusion, shadow and a variation pass. With the emphasis of this project not strongly on the compositing level this report will not go into too much detail on the compositing details since many of them are standard procedures.

Variation was added to the flowers by randomly selecting between 6 different textures. Each texture was slightly different so to avoid the flowers all looking the same. The hue value was then also randomised on a per flower level to add additional variation. Since one texture was used for all the leaves, variation was added by using a separate pass, this was the variation pass. Each leaf generated a random colour at render time in this pass. In compositing this pass could then be used to vary the colour of each leaf.

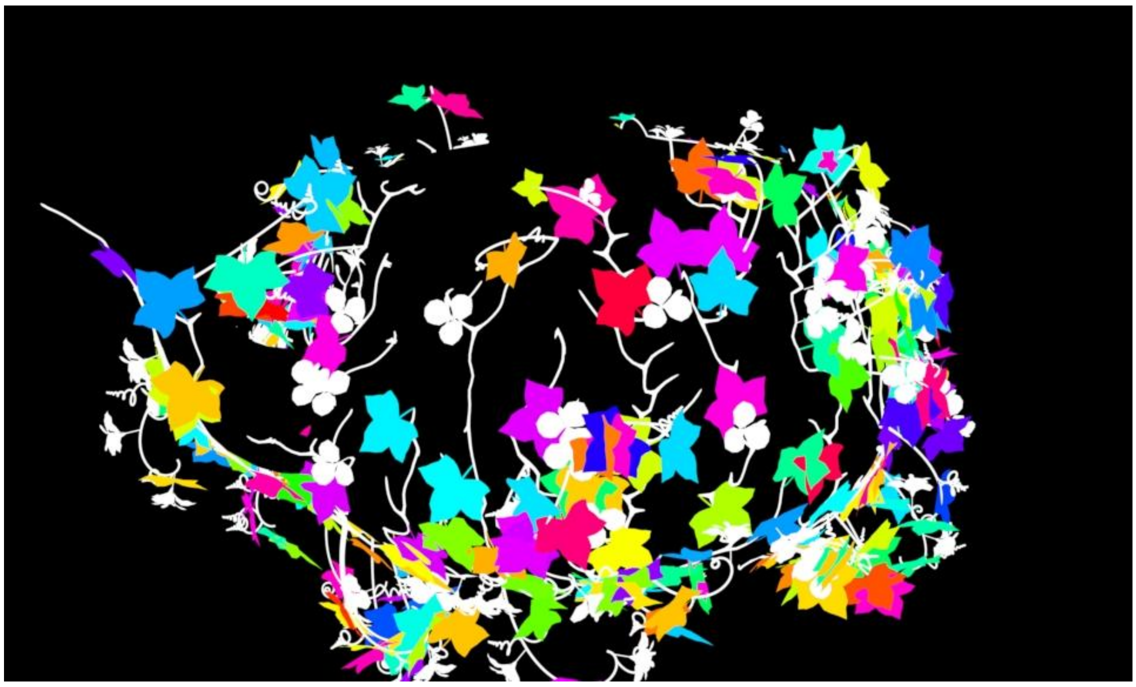

Figure 20: The variation pass, each leaf has a random number assigned to it. Using this it generates a random colour at render time which comes out as a separate pass.

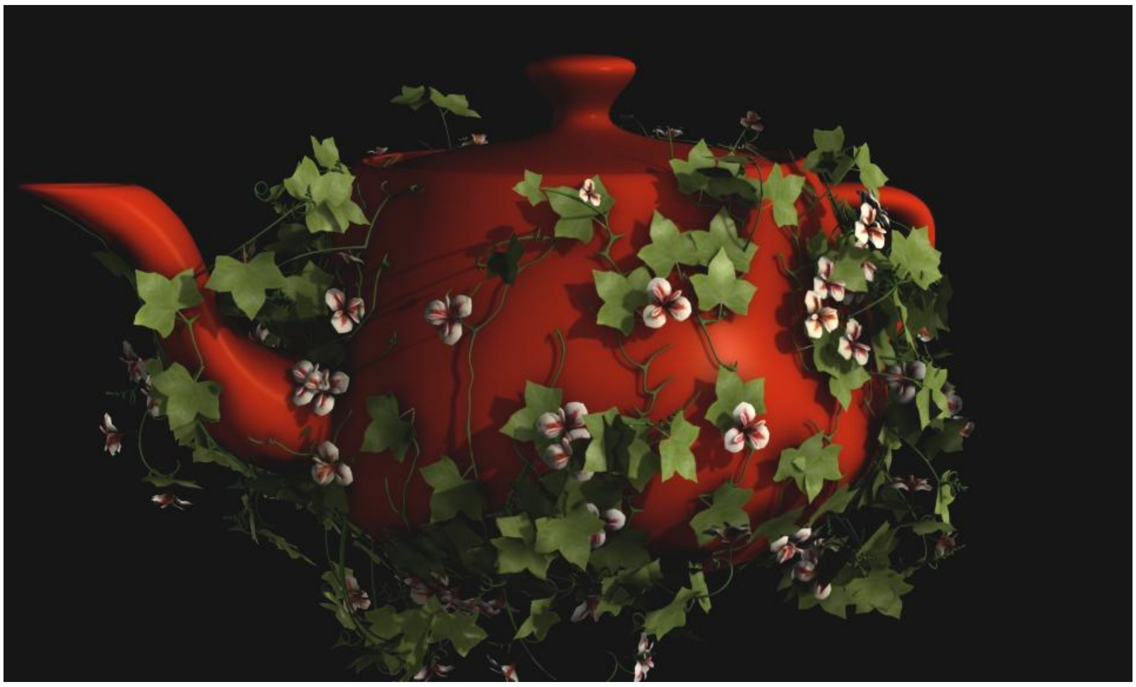

Figure 21: Final composited image without using the variation pass.

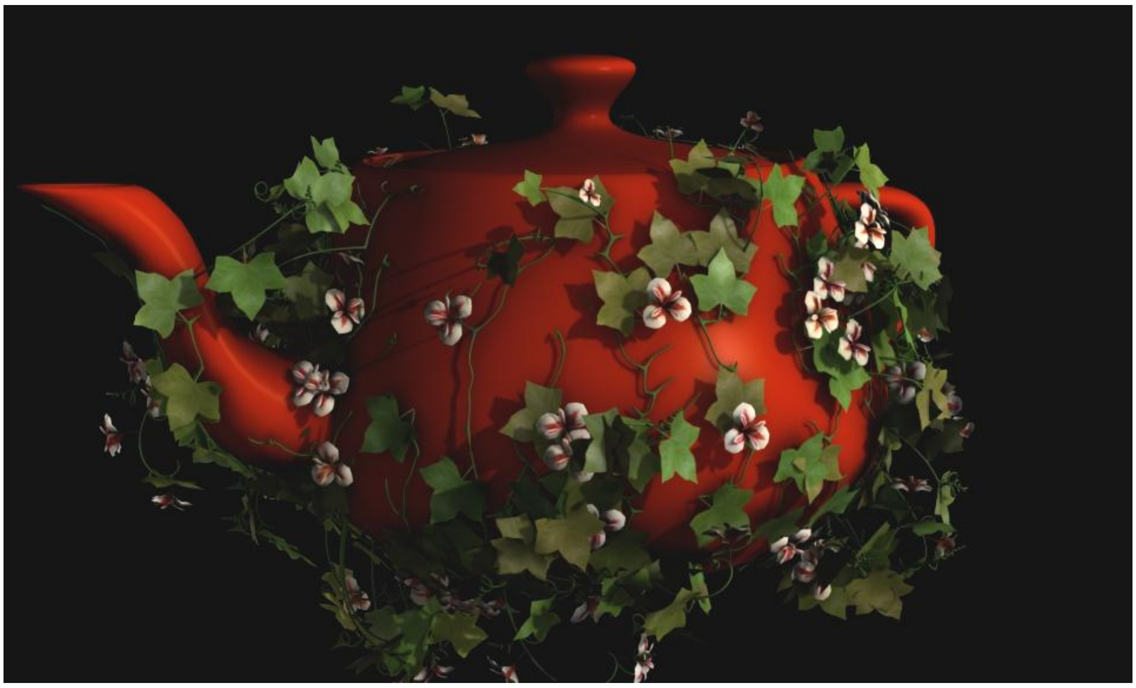

Figure 22: Final composited image using the variation pass.

7.2 Texture and Displacement Maps

Rendering the wireframe created in section 6.0 creates a texture map made up of lines in a fractal arrangement. This pattern can be animated and lends itself to many uses from veins on a creature, to rust propagation or a lava effect over an object.

For a vein effect on a creature the texture map generated can be used directly in a shader as a displacement and texture map. The effect creates bulging veins similar to that of the Incredible Hulk.

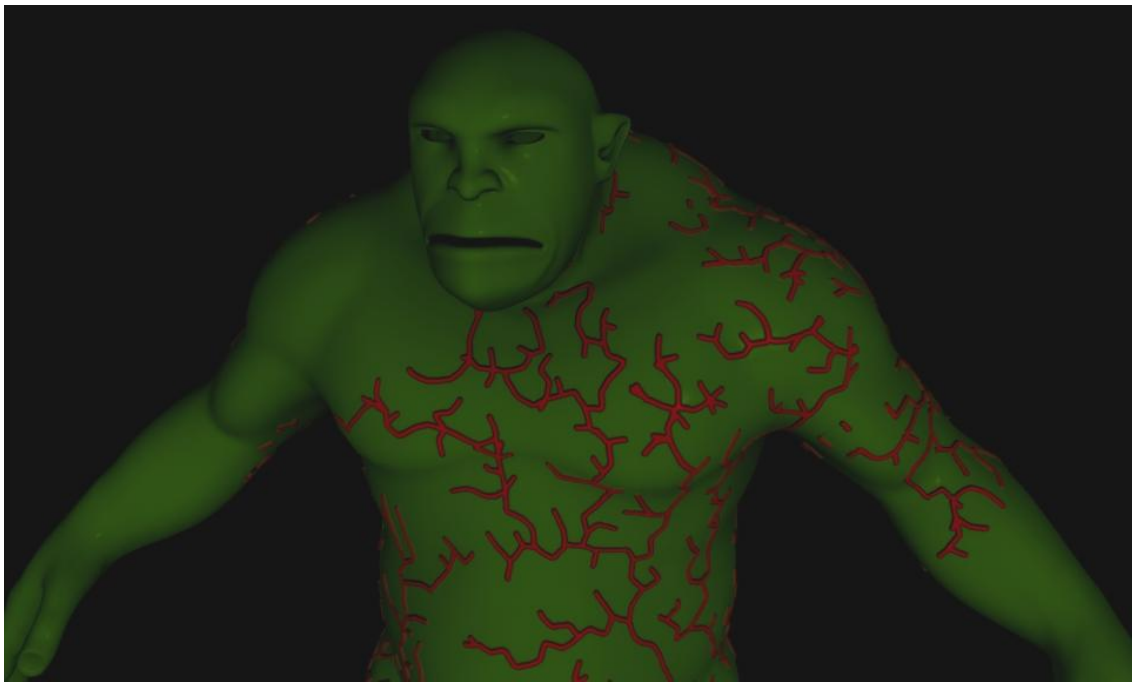

Figure 23: Using the texture map directly to control colour and displacement creating bulging veins.

Taking the fractal lines produced and adding a noisy surface to them adds further functionality to the patterns created. This means that although the underlying pattern remains, instead of crisp thin lines, blurred lines with noisy edges are produced. This is a simple effect to add in compositing by multiplying the blurred lines by a multiple noise patterns.

These noisy patterns can now be used to for rust propagation over a metal surface, or more creatively in a lava effect over a troll.

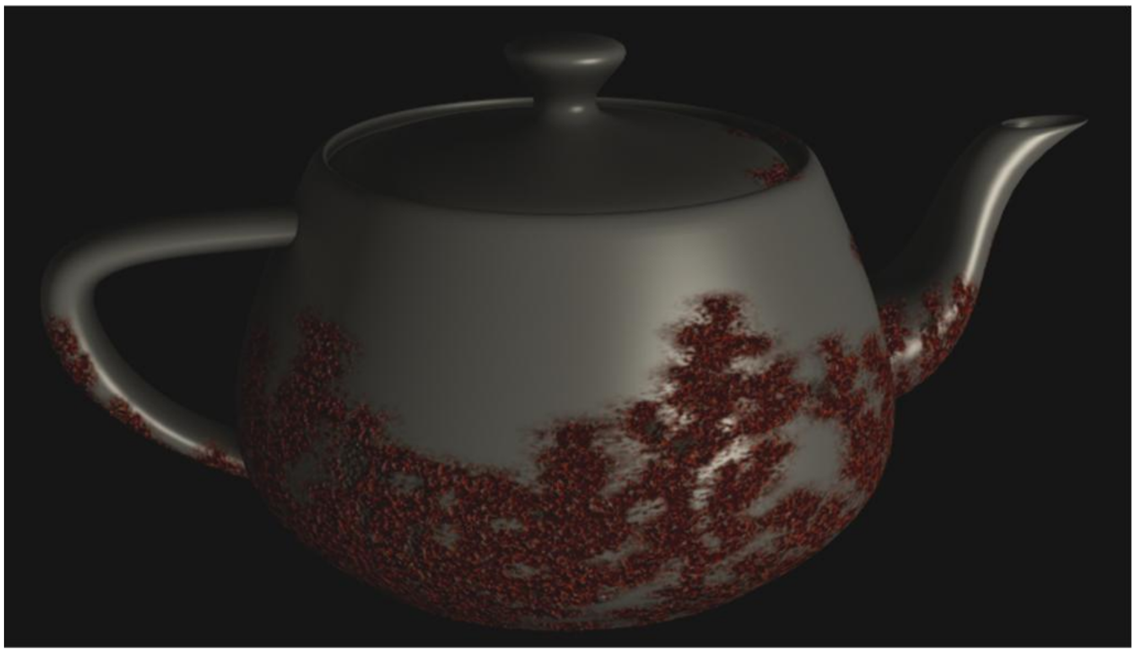

Figure 24: Using a noisy version of the texture map to animate rust propagation over a teapot.

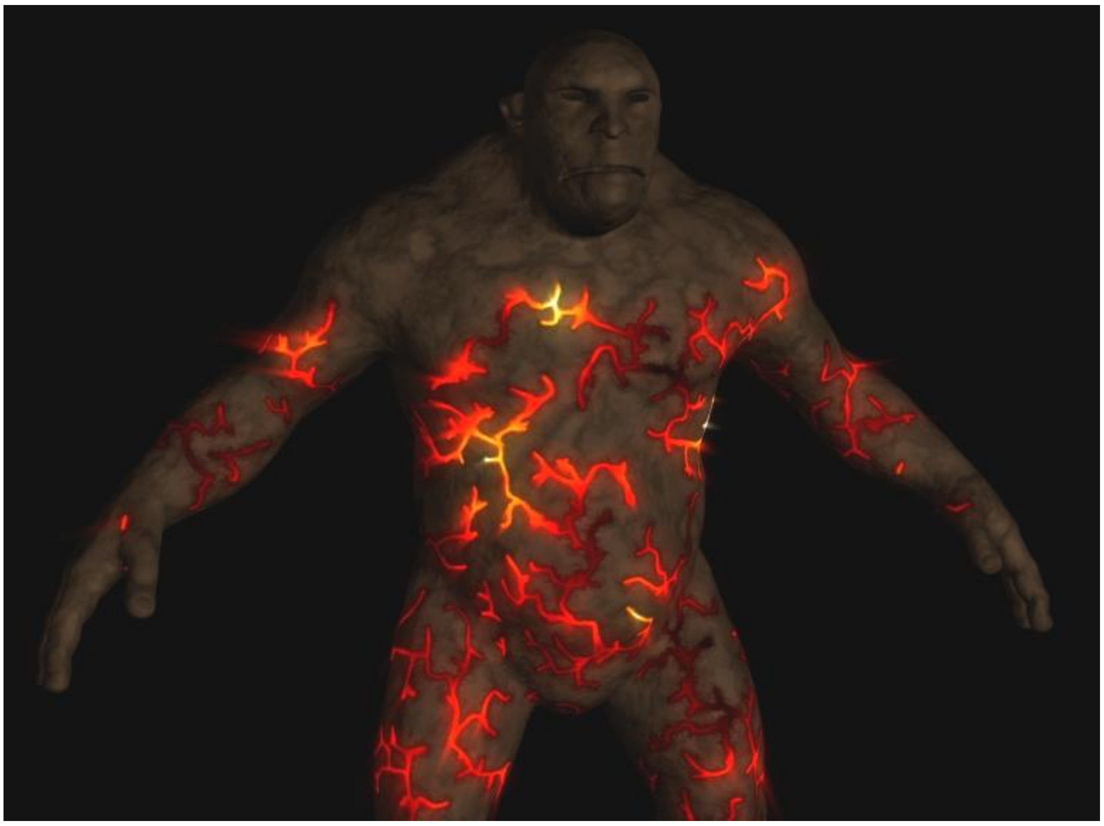

Figure 25: Using a noisy version of the texture map to create a lava effect over a troll.

8.0 Efficiency

Using DLA to generate the fractal patterns requires the process of simulating the particles over the object. Typical simulation times using the tool create in this project range between 2-3 minutes to generate a full pattern. Houdini's in built l-system generator, the most similar tool in Houdini, can generate simple fractal patterns almost instantaneously and complicated ones in less than 30 seconds.

A criticism of the tool created during this project would therefore be that it is much slower than standard tools. However 2-3 minutes is not an excessively long time to wait, often fluid simulations can take far longer. With practise a user can usually generate a pattern they are happy with within 3 attempts, meaning the basis of a shot can be established within less than 10 minutes. If this had been attempted manually without aid of a tool this could have taken several hours.

9.0 Conclusion

The tool that has been created successfully allows the user to input virtually any shaped piece of geometry and generate a series of fractal lines. These lines can then be manipulated to create geometry and be used to grow vine like plants over the object. This could be an incredibly useful tool for many projects as it creates realistic and non-procedural looking growth patterns over the object. The tool is highly adaptable and could be updated with different leaf or flower geometry, or used with different shaders to create a variety of looks depending on the production it is used for.

The tool is also adaptable in that it can be used to generate texture or displacement maps. Virtually all 3D software packages come with built in procedural noise patterns that can be used in shaders. However procedural textures often suffer from looking too procedural and don't look unique enough for the object they are placed on. The advantage of this tool is that the fractal patterns have been simulated on the object, and as such are generated in a way that is sensitive and unique to its topology.

Procedurally generated textures would also be incredibly difficult to grow and animate over an object, and asking a texture artist to paint potential several hundred frames worth of texture or displacement maps is a time costly task. The tool created does this automatically and has the advantage that if the user isn't happy with the look they can quickly create a new variation. The tool also means that it easy to adjust the frame range that the animation spans, whereas making these adjustments with a texture artist would be a very slow process.

A criticism and limitation of the tool is the size of geometry it can be used with. The larger the geometry the greater the number of particles are needed to simulate the patterns. When this number reaches a critical amount the simulation process becomes very slow and near impossible. A practical solution to this is that the geometry can be scaled down for the simulation process and then everything can be scaled back up once this is finished. However to keep a high enough level of detail on large objects this would not be feasible. Without a much more powerful computer using particles to simulate the patterns appears to create a limitation with the method. However, this method still works well for a large range of objects and creates good results.

As explained earlier, creating SDF volumes for intricate detail is another area that is a problem when using this tool. Finding an improved way to keep particles on the surface of an object is an area that requires further research if this tool was to be taken further. The tool’s design means that a new solution could be easily slotted into its structure and the rest of the system could continue to work seamlessly. For this reason I believe the tool has great potential to be taken further as it has a very solid core. Improvements or additions could easily be made making it more robust and diverse.

The tool achieves the projects goals; the outcome is a highly versatile creation that could be used for countless purposes. The wide scope of the possibilities of the tool made it impossible to achieve all of them but by branching into the two regions of generating geometry as well as affecting the shader level this project has created some very visual proofs of how the tool could be used in production.

To take the use of the tool further using the fractal patterns created to generate cracking of an object would be an exciting development. Time constraints of the project meant that this was an area that could not be fully looked into but the patterns that the tool can create would certainly be well suited to cracking or fracturing.

References

[1] Lindenmayer,1968, T*he algorithmic beauty of plants*, Available from http://algorithmicbotany.org/papers/abop/abop.pdf [Accessed 2 August 2009]

[2] Side effects software, 2009, *Houdini Master Education Edition.*(10.0.295)[computer program], Toronto: Side effects software.

[3] Witten & Sander, 1981,*Diffusion-Limited Aggregation, a Kinetic Critical Phenomenon,* Maryland, The American Physical Society, Availiable from : http://prola.aps.org/abstract/PRL/v47/i19/p1400_1 [Accessed 16 June 2009]

[4] Wirtz F,2007, *Diffusion-Limited Aggregation and its Simulation*,Düsseldorf Unterbilk, Available from: http://www.gut-wirtz.de/dla/ [Accessed 15 June 2009]

[5] Lomas,A. *Images of Aggregation*, London, Andy Lomas, Availiable from: http://www.andylomas.com/aggregationImages.html [Accessed 24 June 2009]

[6] Brown R, 1828,*A brief account of microscopical observations made in the months of June, July and August, 1827, on the particles contained in the pollen of plants; and on the general existence of active molecules in organic and inorganic bodies*, Availiable from : http://sciweb.nybg.org/science2/pdfs/dws/Brownian.pdf [Accessed 29 July 2009]

[7] Bourke P, 2004, *DLA - Diffusion Limited Aggregation*, Western Australia, The University of Western Australia, Available from: http://local.wasp.uwa.edu.au/~pbourke/fractals/dla/ [Accessed 15 June 2009]

[8] Hutabarat W, 2009, *Signed Distance Function*, Cambridge, The University of Cambridge, Available from : http://www-edc.eng.cam.ac.uk/~wh226/first_year_report/node37.html [Accessed 13 July 2009]