Master Thesis

Using High-end Lighting/Rendering and Procedural/Dynamic Animation Tools and Techniques in Production:

To Create a Short Film with a Convincing Portrayal of the Interaction of Light with a Melting Effect.

Joe Gaffney

Thanos Topouzis

MSc Computer Animation and Visual Effects

NCCA 2009

Table of Contents

Introduction - Initial Aims

Our aim was to use high-end Lighting/Rendering and Procedural/Dynamic Animation tools and techniques in production with the final outcome of producing a short entirely CG animated film. Also a by product from this aim was the generation of numerous tools and techniques that would aid in this production pipeline. We aimed to utilise the power of software and techniques for their strengths in the process of production(for example Houdini for its powerful ability for procedural animation and effects, RenderMan for its ability to handle complex scenes and geometry which where required in the project) to create a short film with a convincing portrayal of the interaction of light with a melting effect. Finally to work successfully as a team with the pipeline and to uphold the aims.

Pre – Production/ Concept Art

Constants and Variables

In order to proceed with our plan for our Master Thesis we had to find a concept that would fit our requirements. We had a lot of initial concepts but the one we prefered the most, was one that we were going to have a statue that was going to be melted by a magnifying glass. At that stage we were approaching the concept as a set of “constants” and “variables” in which the constants were the statue made of wax and the magnifying glass, and the variables were the scale of the statue and the environment that was going to be in.

Initially we though that the statue can be at an outdoor place, the scale of it it would be large and the magnifying glass could be -with a lateral way of thinking - “the hand of god”. For that reason early on in the process we started doing some tests to see the look of this concept. However we soon decided that this approach had a lot of potential problems in terms of which exactly was going to be the location and how we were going to recreate the environment which in this case it, most probably, had to be a texture projection, something that would have given us limitations on the long run.

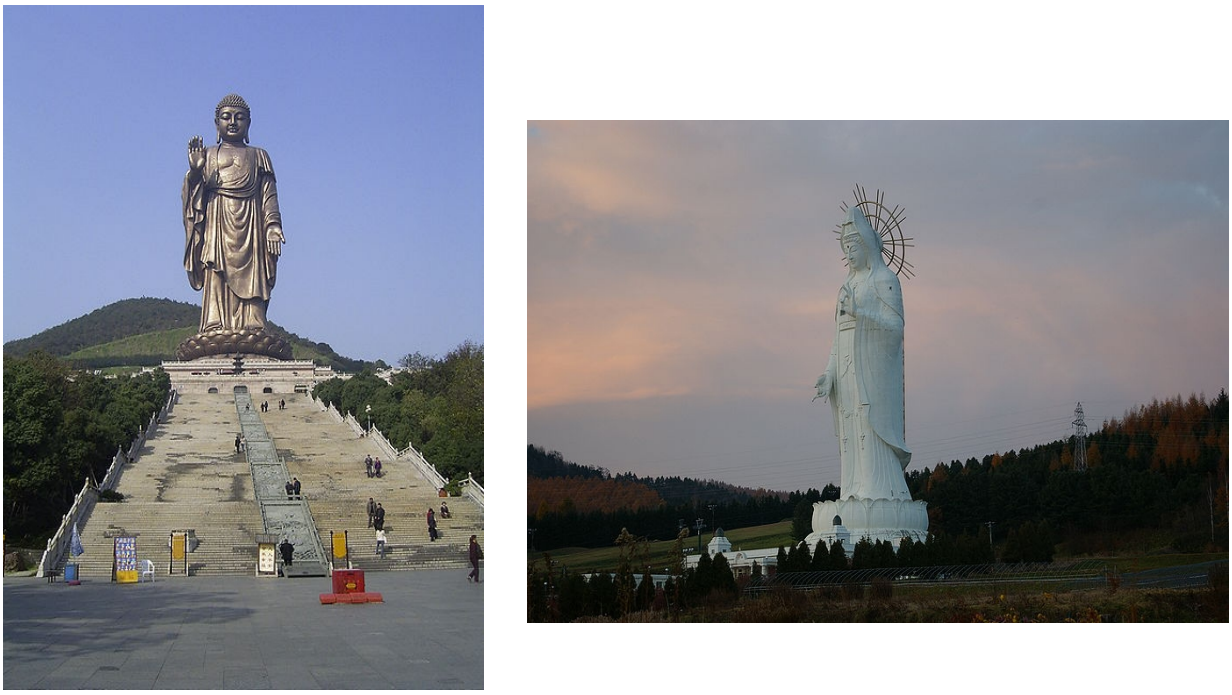


Fig. Research on outdoor environments for statues
(source http://getwonder.com/07/top14-14-biggest-statues-in-the-world.html)

Fig. Early tests with the statue in an outdoor environment

The next idea that we had was the statue to be under construction in an indoor place and to have scaffolding around it. The limitation of this idea was that we wouldn't have the sun as the main light source with which the magnifying glass would melt the statue..

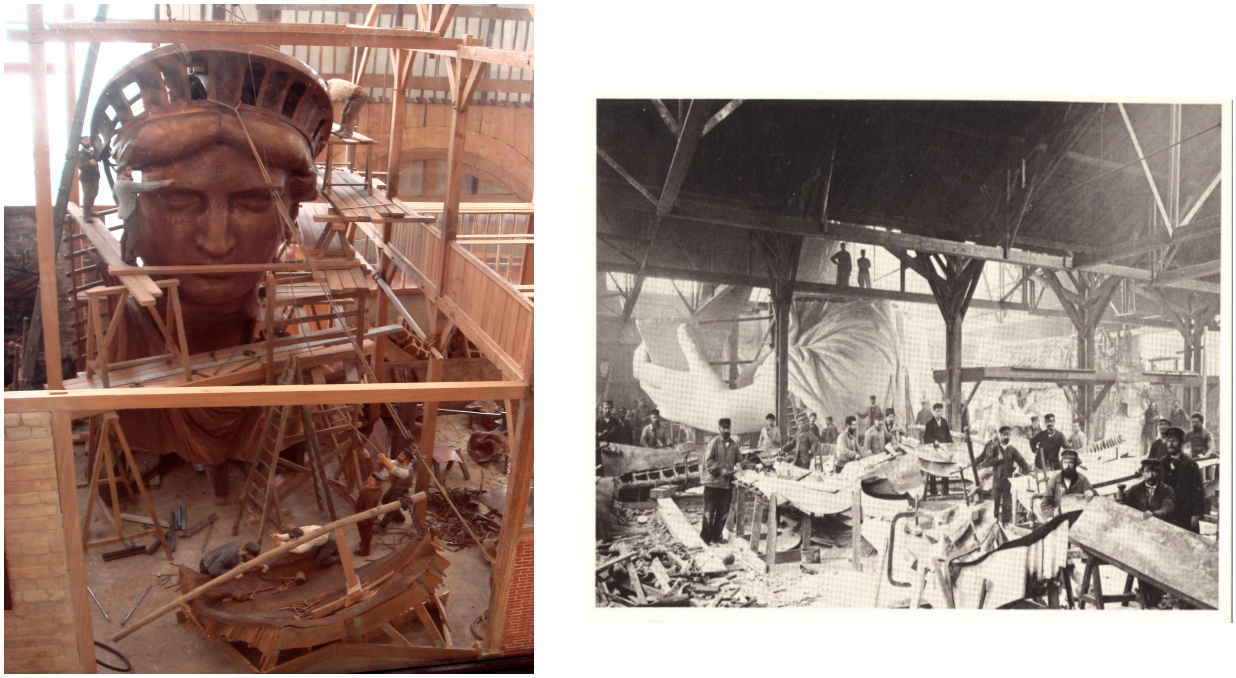


Fig. Research on indoor environments and scaffolding (google images)

The idea that we prefered the most was the scale of the statue to be really small and the environment to be a garden. That wouldn't give us any limitations in the shots that we were planning to do, and it would be an interesting task to develop the elements of the garden.

Fig. Research on having the statue in a garden.

(source google.com)

The wife that was once beautiful

After having decided what the concept for the environment would be we had to find the statue that would "tell a story" and fit our artistic needs. Before researching about the statue we already had in our minds that we wanted the statue to represent colours of the human nature and not to deal with any historical persons or to have to do with religion, civilizations and mythology.

One of our favourites sculptures was Auguste Rodin's "The gates of Hell", which comprises 186 figures[1], amongst them the famous figure "The Thinker". Studying the sculpture and its symbolism we found out some very interesting figures. One that drawn our attention was a sculpture called 'Celle qui fut la belle heaulmière' which is also known as 'The old Woman', 'The old Courtesan' and 'Winter'. Rodin modeled it after a 82 year old woman named Caira, because he was fascinated by the inevitable decline of human beings with its different mouldings of ugliness and personality. Rodin founded that what we call commonly "ugliness" in nature can in art become full of great beauty. In art, only that which has character is beautiful. Character is the essential truth of

any natural object [2].

Picture 8. Auguste Rodin's sculpture "The old Woman" (google images)

Paul Gsell associated the sculpture in his 'Conversations with Rodin' with the poem of François Villon - a monologue of the old helmet-maker's wife about her expired beauty:

"Ah, wicked old age
Why have you struck me down so soon?
[You] have stiffened me
so that I cannot strike
And with that kill myself!

When I think, alas! of the good times,
What [I] was, what [I] have become,
When [I] look at myself completely
naked
And I see myself so changed.
Poor desiccated thin, shriveled,
I nearly go mad!

What has happened to

my smooth brow,
My blond hair... .
My slender shoulders,
Small breasts, firm tight
High, clean, perfectly made
For love's pleasures; (.....)
This is the fate of human beauty!
Shrunken arms and clenched hands
[And] completely hunchbacked.
What breasts! All wizened
Like my hips... ."

Having studied this sculpture we got inspired for our own project idea. We though about modelling the statue but that would be unnecessary because we wanted to focus on different areas. So we started searching for some free models of this statue. We didn't find any free models of the statue but we were able to purchase it from gnomology.com.

Fig. The Zbrush model from gnomology.com

Having the statue we had our main element in the scene and we were ready to move on the next

stage which was the storyboards and animatic. Adopting Rodin's symbolism we decided to name the piece as “The wife that was once beautiful”.

Storyboard and Animatic

Before we do the final animatic we though about going back a few steps and test how this statue would look on our first concept. So we did a first rough animatic.(fig 10).

Fig. First Animatic

Nevertheless for one more time we weren't satisfied enough with the results so we decided to follow the concept that we had agreed in. At that point we started researching so as to start having a better idea of how the look of the garden would be, and what shots we were going to use.

Fig. Reference images (google images)

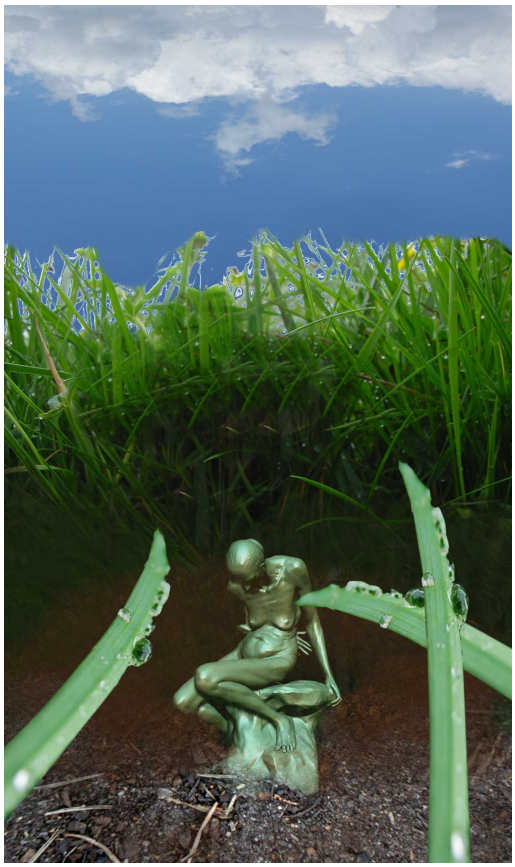
To get some inspiration we watched some films that they had similar subject like “The Ant Bully” (fig), “Bug's life” (fig) and “Honey I shrunk the kids”.

Fig. Reference screen-shots from “The Ant Bully”



Fig. Reference images from "bug's life"

After gathering all these informations and ideas we were ready to create our animatic. Our goal was to create a short piece trying to achieve "quality over quantity". For that reason we chose to have not more then six to seven shots. We also tried to achieve high cinematography standards choosing the framing of each shot carefully. The storyboards were as follows:



Shot1 – Long Shot

This is the master shot. We start looking at the top of the grass and then we pan down until we reach the ground, and see the statue.

Shot2 – Medium Long Shot

The camera is tilt and we see the magnifying glass approaching.

Shot 3 – Medium Long Shot

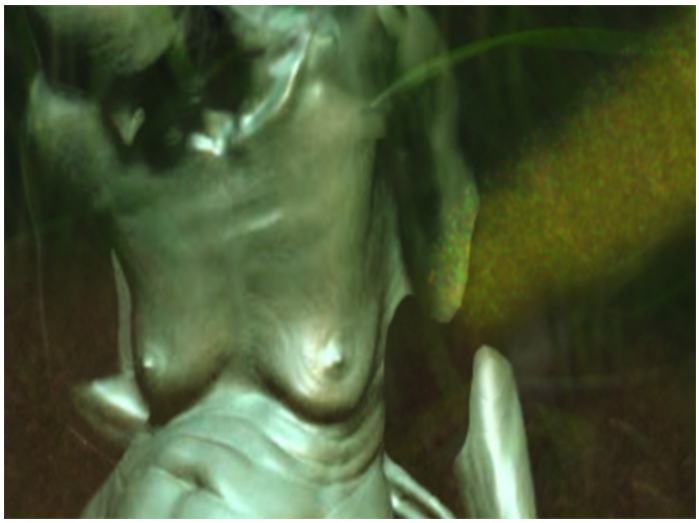
The camera comes back to the statue and we see a strong light beam melting part of the statue.

Shot 4 – Medium Close-Up

We are moving closer to the action with a medium CU to the shoulder of the statue that is melting.

Shot 5 – Medium Long Shot

The camera comes back to the statue and we see the light beam changing its focus point from the shoulder to the arm.



Shot 6 – Medium Close-Up

The camera cuts closer so that we see the arm melting.

Shot 7 – Long Shot

The camera pans-out while the statue is left alone, so as to give a sense of loneliness and insignificance.

During the process of developing the piece we had to change some of our initial storyboards in order to improve it.

We decided that we were going to use Maya to set-up the scene. We were mainly going to use an analysis of a plant arrangement done by Bill Fleming in his book “Digital Botany and Creepy insects” in which the usual arrangement is: one or two large tall plants in the background, several medium-sized plants in the middle ground, about one third of the height of the larger plants and surrounding the larger plants, and several smaller plants in the foreground, usually grass.[8]

Melting Research

Having finished with the animatic we started doing some research on the melting effect. Initially we were going to approach melting wax as a highly elastic and viscous material. One of the papers that were studied was Particle-based Viscoelastic Fluid Simulation
by Simon Clavet, Philippe Beaudoin, and Pierre Poulin..[9] In this paper they implement a method of fluid stickiness by adding an attraction term to the particles that are close to an object.

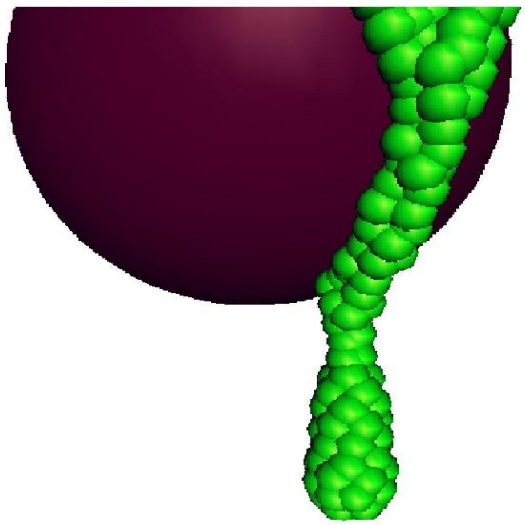
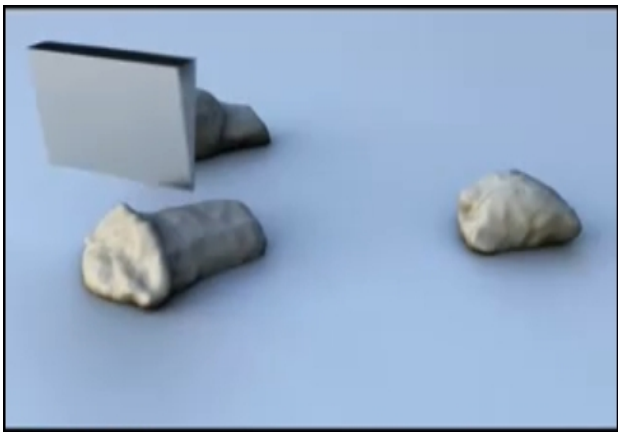


Fig Stickiness implemented in the paper particle-based Viscoelasticity by Clavet, Beaudoin, Poulin

However we were mainly planning to use the nodes provided by Houdini to implement methods and ideas. We where also planning to make a shooting under the same conditions to see how it works physically.



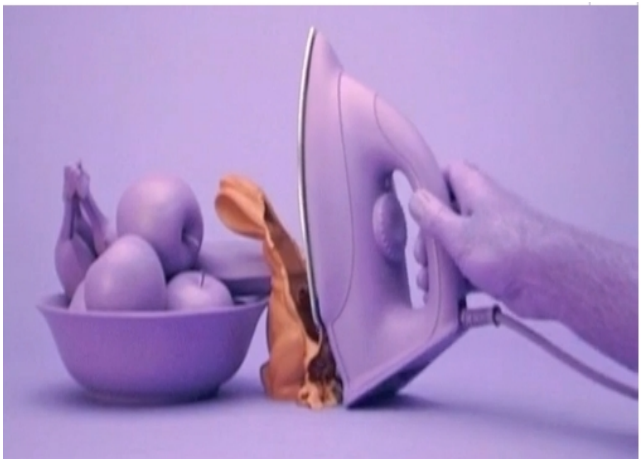



Fig Melting research

After hiring a DV camera and gathering all the props(wax candle, magnifying glass) we filmed one hour of footage in time-lapse so that we could capture how a wax candle lighten by a magnifying glass would melt. The results we got were accurate but at the same time not very interesting. So, early on in the process we decided that in order the short piece to look visually interesting we had to be creative and mainly follow our initial plan of having a highly visco-elastic surface.

Production Tools

Procedural and Dynamic Animation

From the beginning of our Masters Project we had clear in our minds that all the animation would be procedural and dynamic. For that reason we had to choose between Maya and Houdini as our main 3D software. Early on though we were quite sure that we were going to use Houdini so as to benefit from its procedural-ism and its powerful features in Fluid Dynamics and Particles. Also the task of creating a melting effect it would be rather difficult if instead of fluid dynamics, particles were used, as the “surfacing” of particles its supported in Maya 2009 which has the nucleus engine implemented in particles(the nparticles). That was a big drawback as Maya 2009 is not included in our available resources in the university. Another main reason for using Houdini is the ability to

create digital assets. This is a very powerful feature that would give us the ability of creating a fully parametrable scene.

Grass Generator Digital Asset

In the process of developing the garden we applied what we initially planned: create digital assets that would model procedurally all the asset and they would also be procedurally animated. The first asset that we had to develop was an asset that would generate grass which would be placed according to a pattern that the user would import. That way we could generate large amounts of varied grass geometry that would fill the area around the statue and make the scene look realistic.

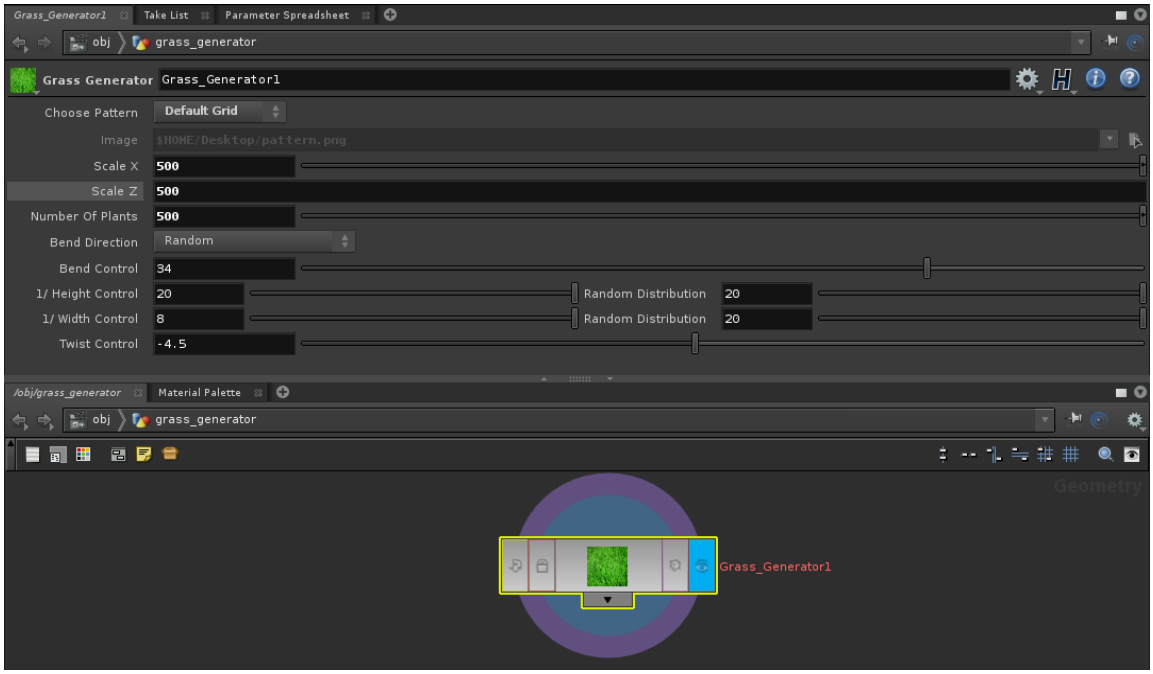


Fig. Grass Digital Asset

The options that are given to the user are:

Choose Pattern

The user has two options so as to choose the pattern. He can either use a default grid or he can import his own pattern. When the user selects this option a the option browse get enabled and the user can browse to find his image.

Scale X, Z

The user can scale the pattern in the X and Z axis

Number of Plants

The user can define the number of plants(or grass chunks) he wants to be copied to the selected pattern.

Bend Direction

The direction where the grass is bending can be either “random” either “all in one side”

Bend Control, Twist Control

These values define how bended and how twisted the grass is. They can take positive and negative values. There is randomization value that gives slightly different bendings to each grass chunk but the values stay always close to what the user choose.

Height Control, Width Control/ Random Distribution

The user can choose the height and the width of the grass. The random distribution value defines how big the variation would be.

The algorithm

The process that was followed in order to give this functionality to the digital asset is the following presented in pseudo code:

Begin

Define the pattern

If (pattern == Import Image) **then**

enable the file browser so as the user to use an image as a pattern

trace the image so as to get a geometry representation of the image.

Set the X, Z scale according to user's preferences

Scatter as many *points* on as the user defines as his desired number of plants

Model a *master* leaf for the grass.

Create an *instance* of the master leaf

Assign the values to the following variables according to user's input

bend_control, bend_direction, heigh control, height_random_distribution, width_control, width_rand_distribution, twist_control

For (1 to *points*) **do**

randNumLeaves = floor((rand($PT))*10) // assign a random number to
// define how many leaves each
// grass chunk will have

For (1 to *randNumLeaves*) **do**

Modify the instance of the master leaf as follows

randomScale = 4*rand($CY)/5 //random leaf scale

randomBend = if($CY%2==1,rand($CY)*(-10), rand($CY)*(10))
//random bending of the leaf

Scale the X,Y, Z value of the leaf according to *randomScale*

Bend the leaf(X,Y axis) according to the *randomBend* value

Merge the modified leaf with the *master* leaf creating the *grass*

End

Modify the *grass* as it follows:

l_width = rand($PT) * width_rand_distribution / width control
// Define the width of the grass

l_height = rand($PT)*height_random_distribution")/ /height_control"
// Define the height of the grass

l_twist = if(rand($PT)>0.7, rand($PT)*twist_control,rand($PT)*twist_control
// Define the amount of twist

l_bend = rand($PT)*bend_control // Define the bending amount

base_rotation if(bend_direction ==0, $PT*45, 0)

*uv_offse*t = if($PT%2==1,rand($PT)*5, rand($PT)*(-5)) //sets the uv X osffet

uv_offsetY = $PT%20+rand($PT) //those two steps offset the uv's so
// their positions in UV space can be

// randomized so as a procedural shader

//is applied

End

End

At that point a procedure overview will take place based on images to the key points of the algorithm:

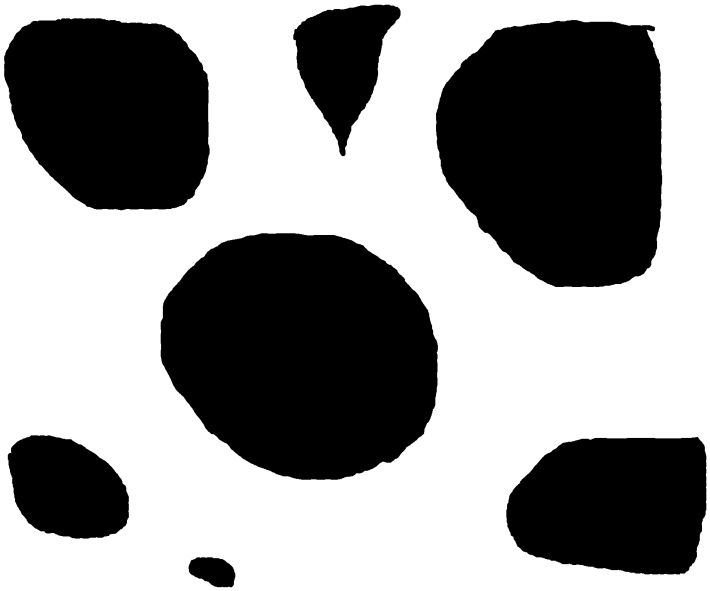


Fig. Import a Pattern

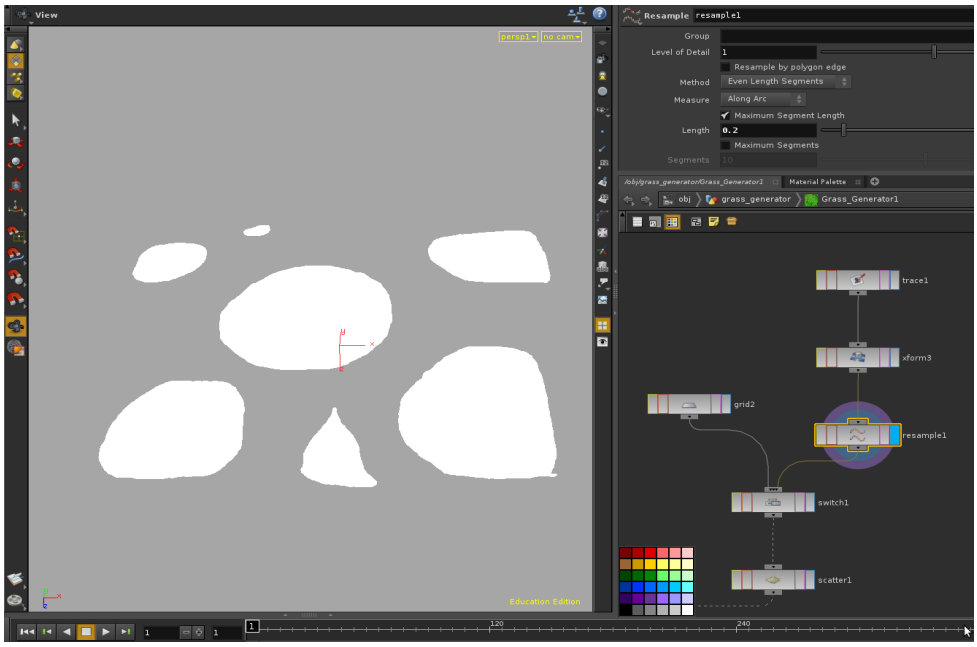


Fig. Trace the image

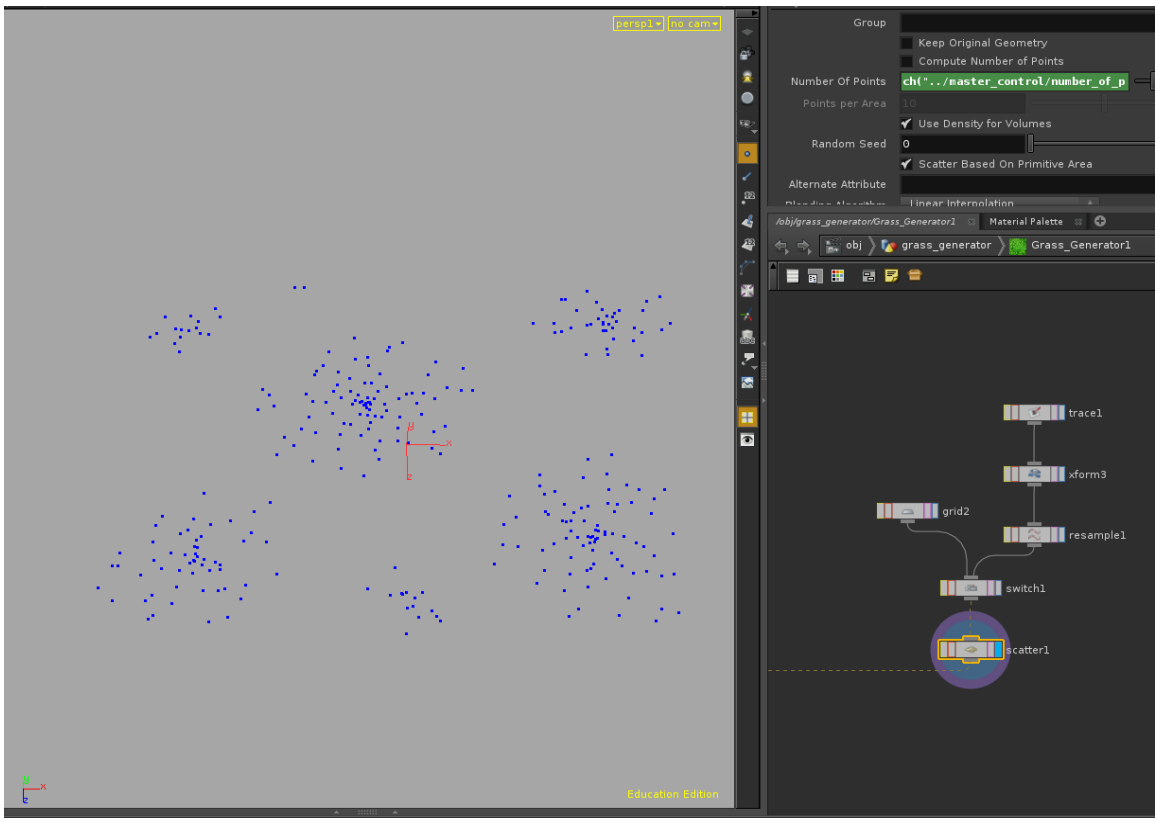


Fig. Scatter points on the surface

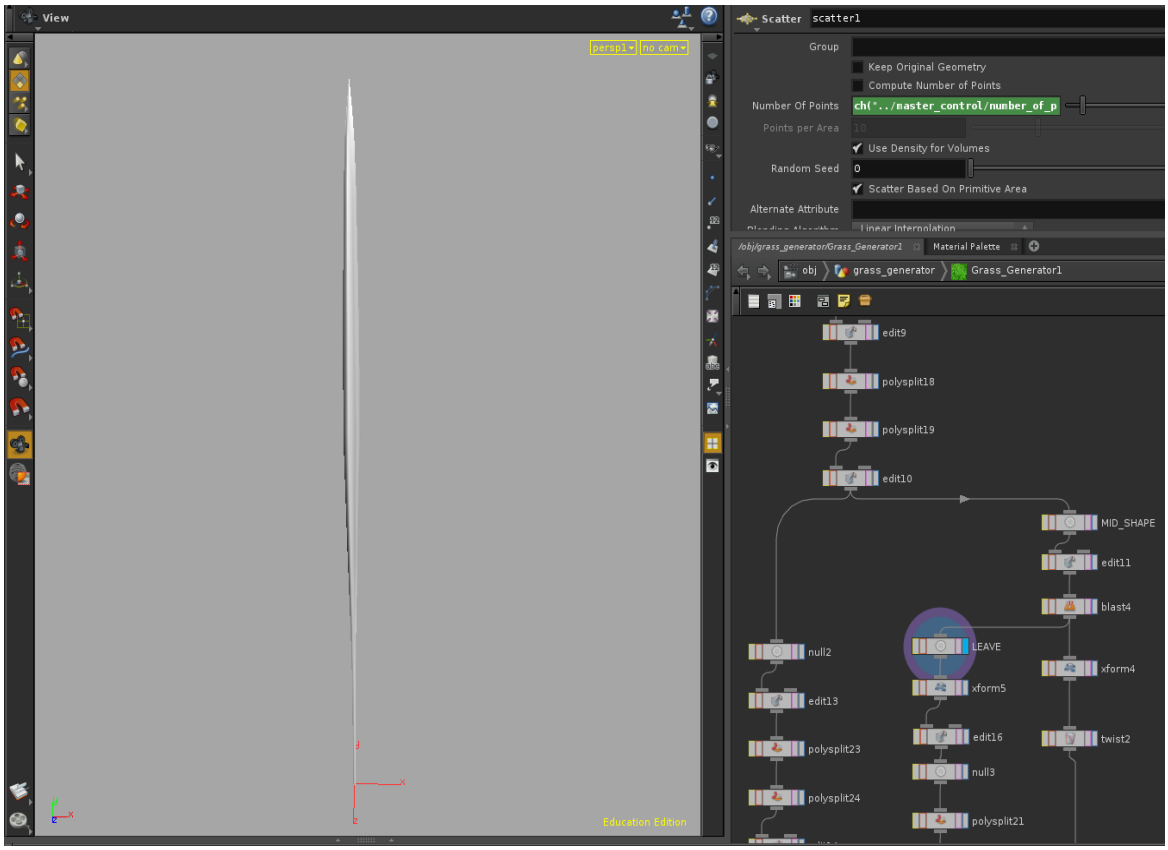


Fig Model a leaf

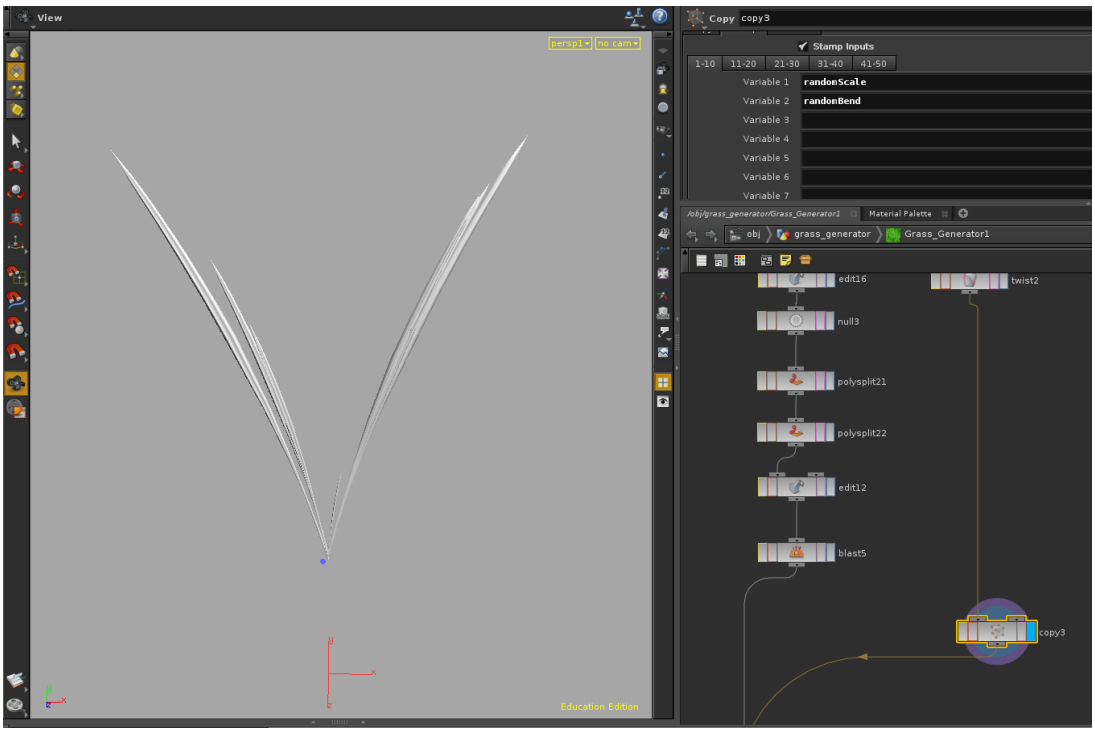


Fig. Create an instance of the leaf, make random copies of it and give them random scale and bending

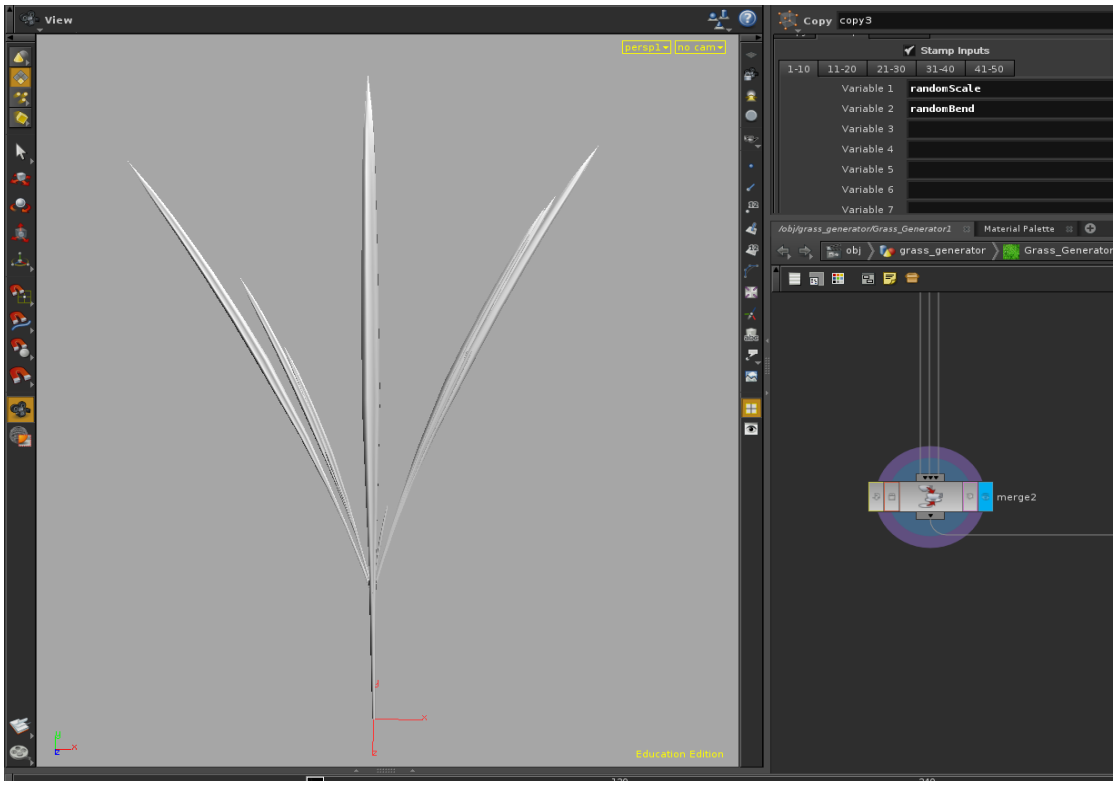


Fig. Merge the master leaf with the copies

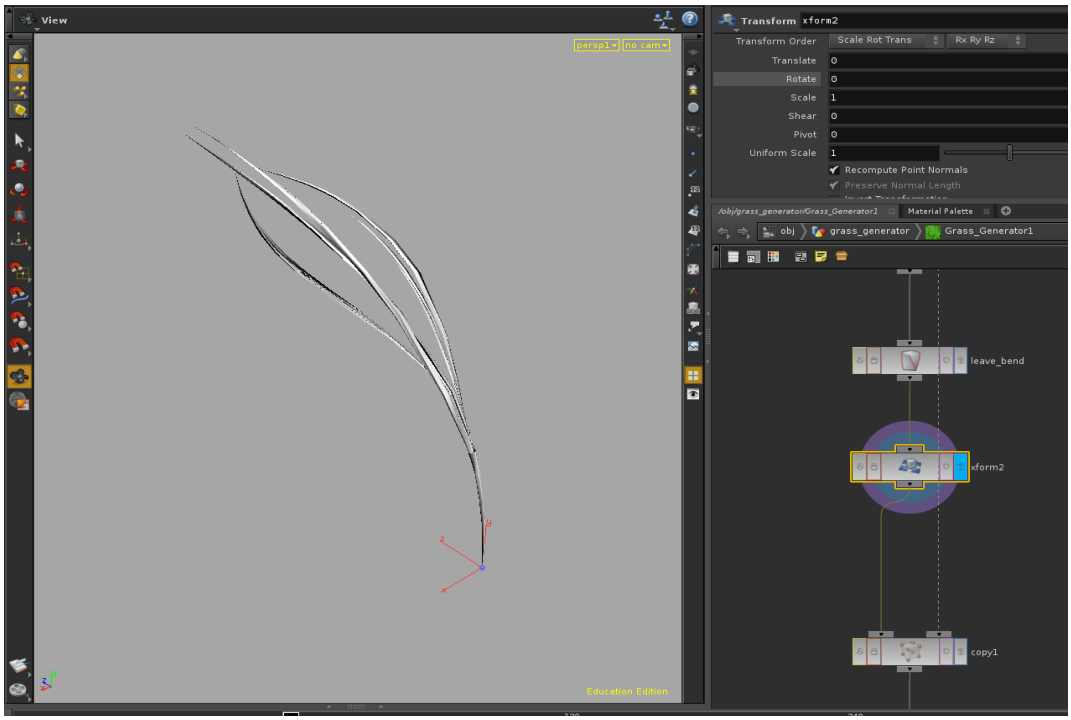


Fig. Bend and Twist, define the Height and the Width it according to user's preferences giving some randomization to them

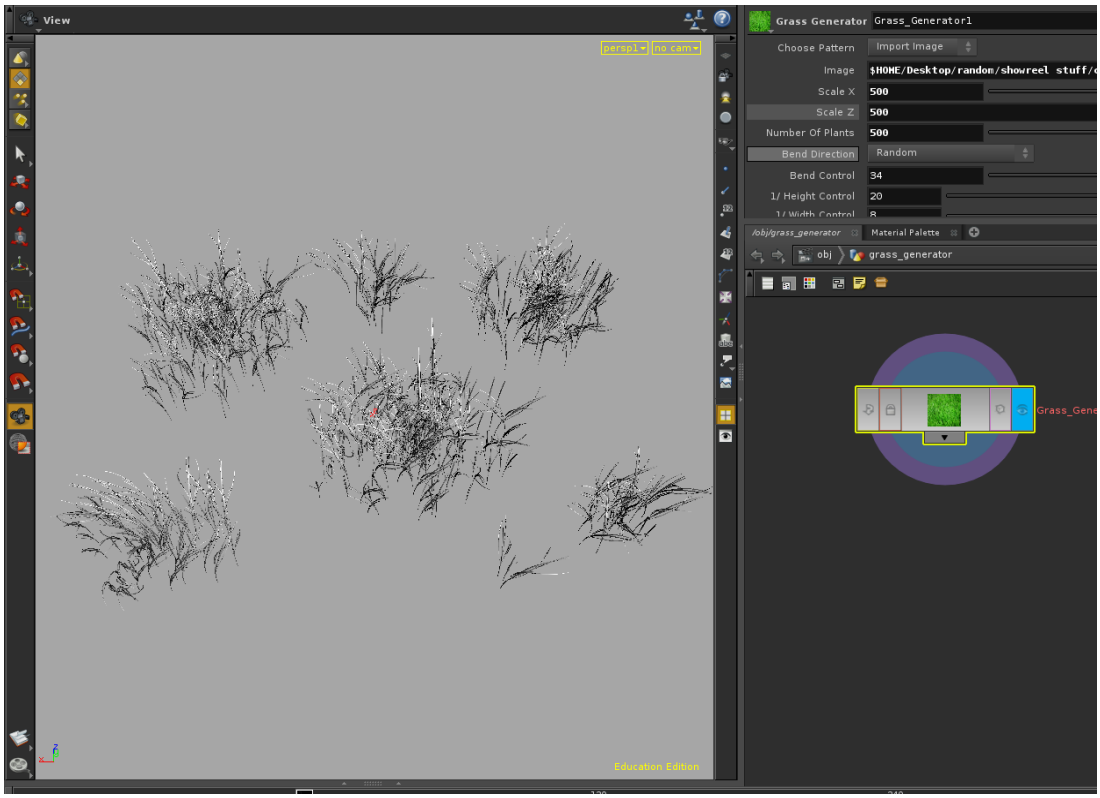


Fig. Copy the grass to the points

The grass then was animated using a lattice deformer and a spring and then it was exported to Maya (which was the main software we were using for scene management) as an FBX sequence.. Although the asset was giving good results, managing to procedurally model relatively convincing low-polygon grass, when the grass was imported to Maya there were some issues with the UV coordinates as well as some geometry issues, with slight geometry penetrations. So we decided to keep this asset and use it for the background grass where we wanted less details and no variation to the shader and modify the tool so as, instead of procedurally modelling the grass, to be able to take grass geometry(with proper uvs) place them like before and randomize the uvs. Having to work on this Digital Asset again, we though about taking it one step further by implementing the motion source (lattice and spring deformer) in build to the digital asset. The new asset was simply named as Grass Generator 2.

Grass Generator 2 Digital Asset

This digital asset keeps most of the functionality of the Grass Generator but it has some extra controls.

Geometry Tab

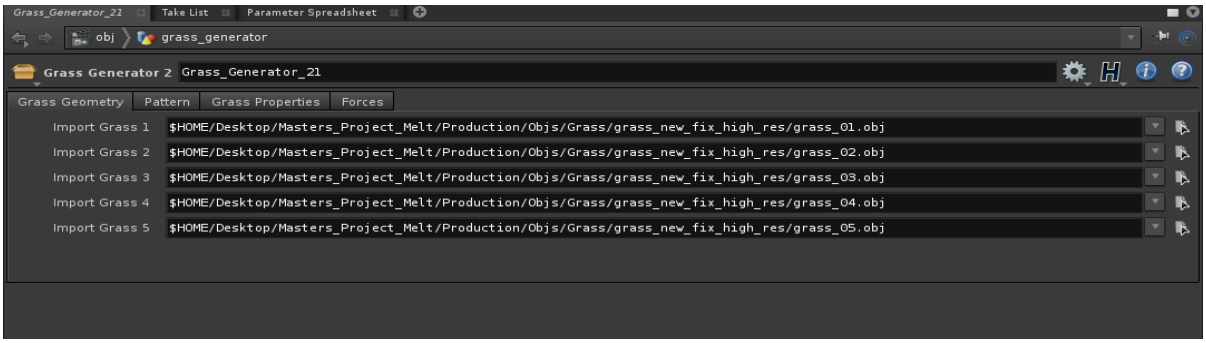


Fig. The geomety tab

In the Geometry Tab there are five geometry imports where the user can import his own grass geometry.

Pattern Tab

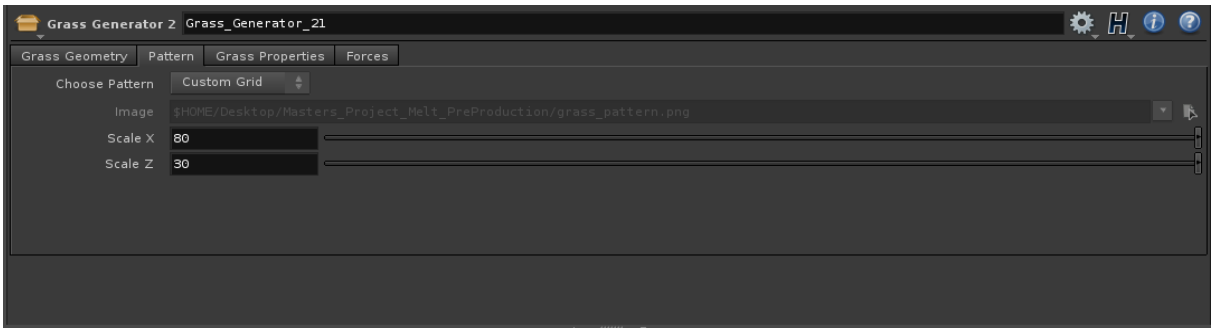


Fig. The Pattern Tab

The pattern tab has the same functionality as the Grass Generator.

Grass Properties Tab

Fig. Grass Properties Tab

This tab has the properties of bending, height and width control, twist, number of plants, that there are available at the Grass Genarator.

Force Tab

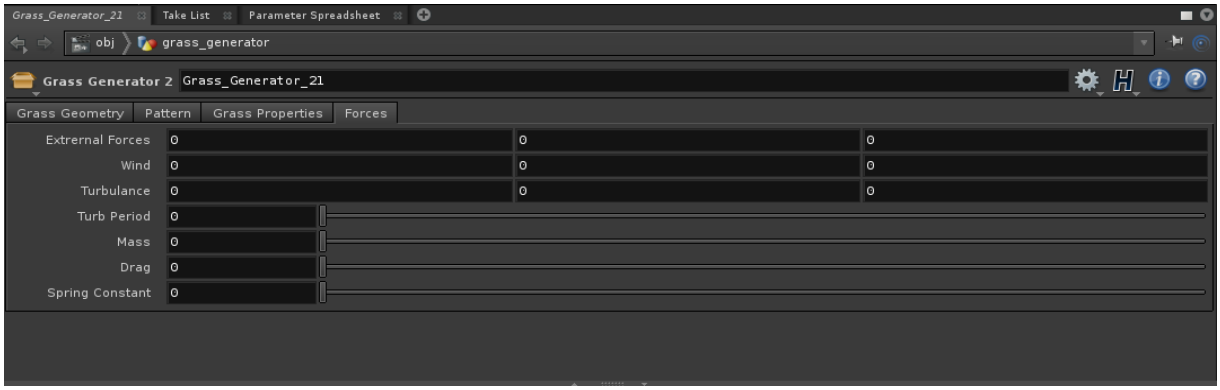


Fig. Forces Tab

This tab is linked to the lattice deformer and the spring. This way the user gets options about

external forces, wind, turbulence, mass, drag and spring constant. By putting expressions (for example the fit function) or by key framing the user can procedurally animate the grass and get a good looking result. That's what was used to the piece to animate the grass.

Algorithm

The algorithm is almost the same. Nevertheless we will present algorithmically the extra features of it.

Randomizing the Uvs

For 1 to *number_of_points* **do**

uv_offset = if($PT%2==1,rand($PT)*5, rand($PT)*(-5))

uv_offsetY = $PT%20+rand($PT)

Transform the Uvs on the X by *uv_offset* and on the Y axis by *uv_offset*

End

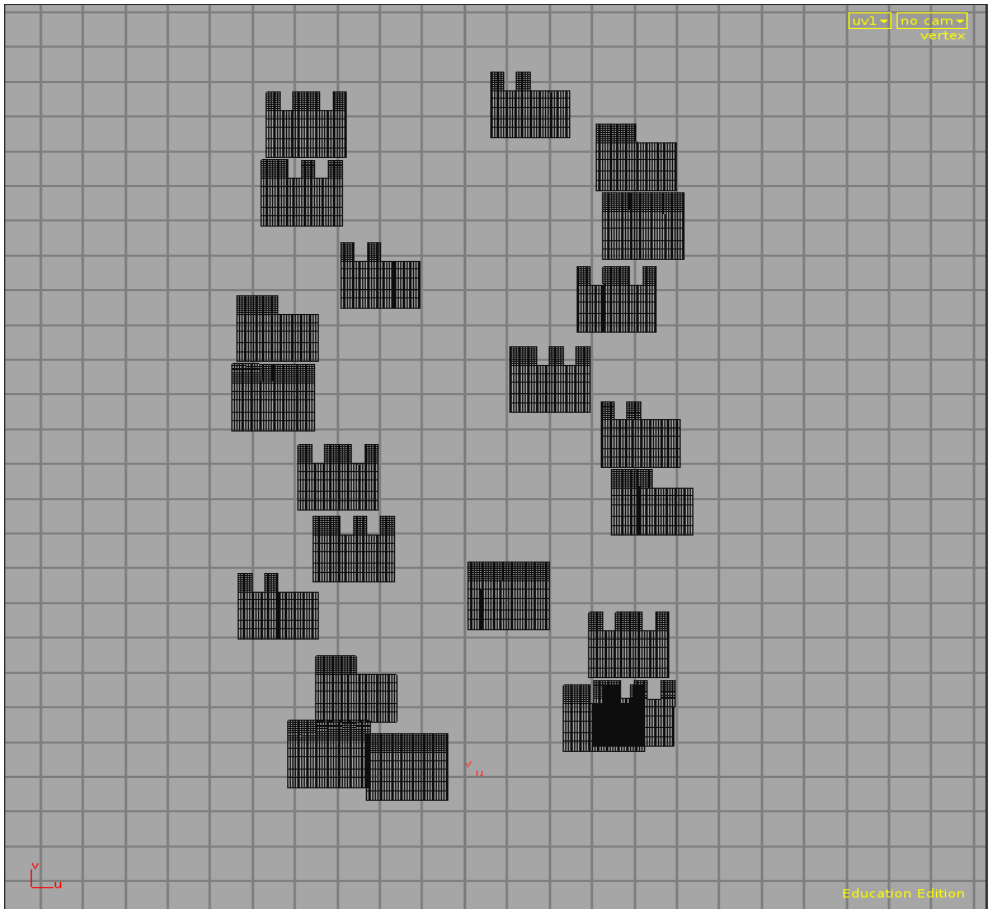


Fig. Random Uvs

Lattice Deformer and Spring

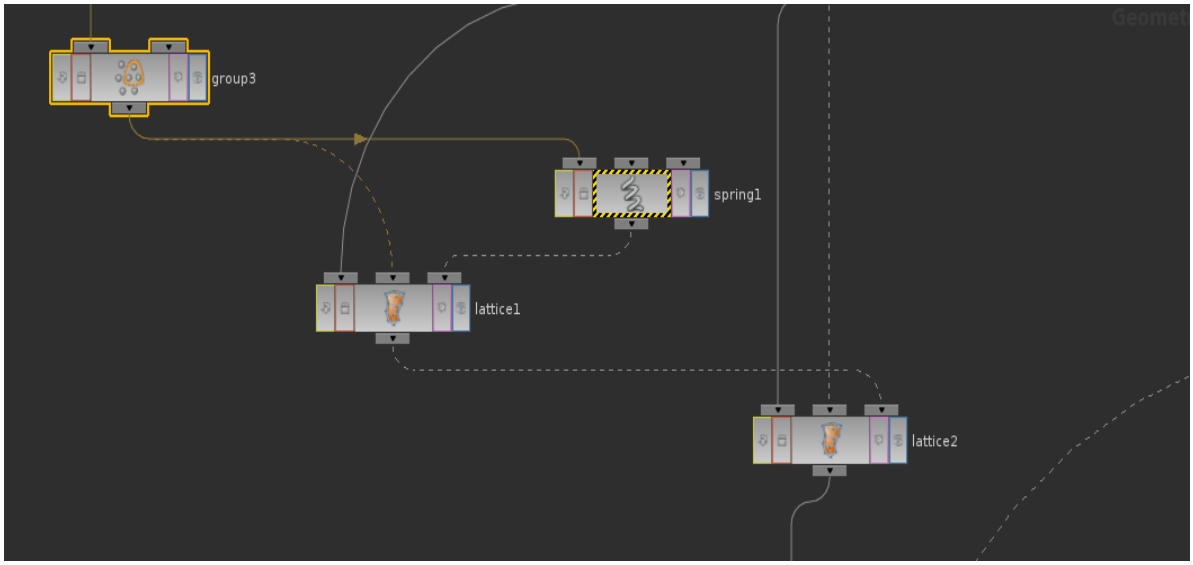
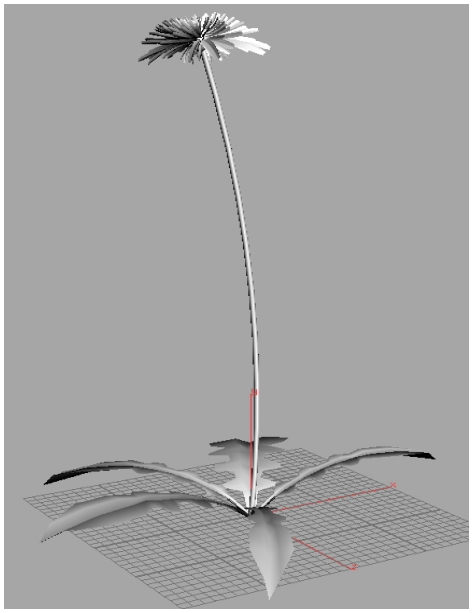


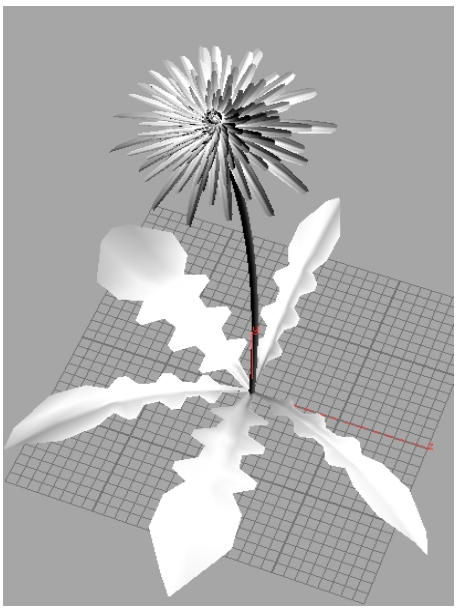
Fig The Lattice deformer network

A lattice Deformer is used to animate one chunk of grass using a spring. These data are than feed to each grasses lattice.

The grass was animated with the lattice in Houdini, than exported to Maya as an FBX sequence

The dandelion Digital Asset





Continuing with developing the assets for the garden we created an asset that would procedurally

model a dandelion flower. By that time we weren't sure how many flowers we would have in the scene. To produce the dandeline asset similar methods to the Grass Generator asset were used using variables and for loops (by copy stamping) to give randomness to the pedals(scale, width) and bounding boxes for the height of the stem and the positioning of the ovary.

The options that are given to the user are as follows:

General Controls Tab

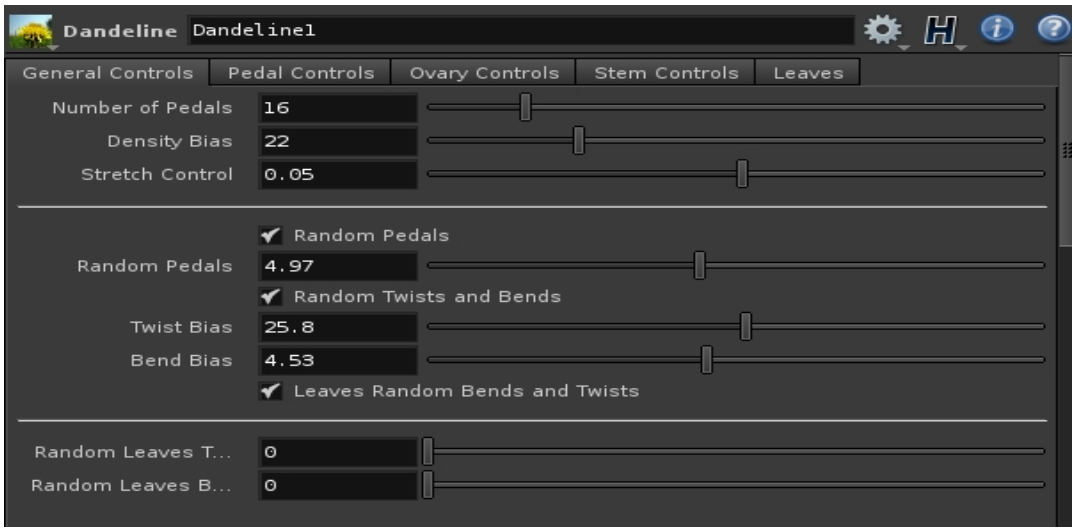


Fig General Controls Tab

Number of pedals: the number of pedals the dandeline has.
Density Bias: how dense are the pedals.
Strech Control: with this option the user can squash or strech all the pedals.
Random Pedals: the amound of randomness at the pedals
Twist Bias: How twisted the pedals are overall
Bend Bias: How much bended the pedals are overall
Random Leaves Twist: Randomly twists the leaves at the bottom
Random Leaves Bend: Randomly Bends the leaves at the bottom

Pedal Control Tab

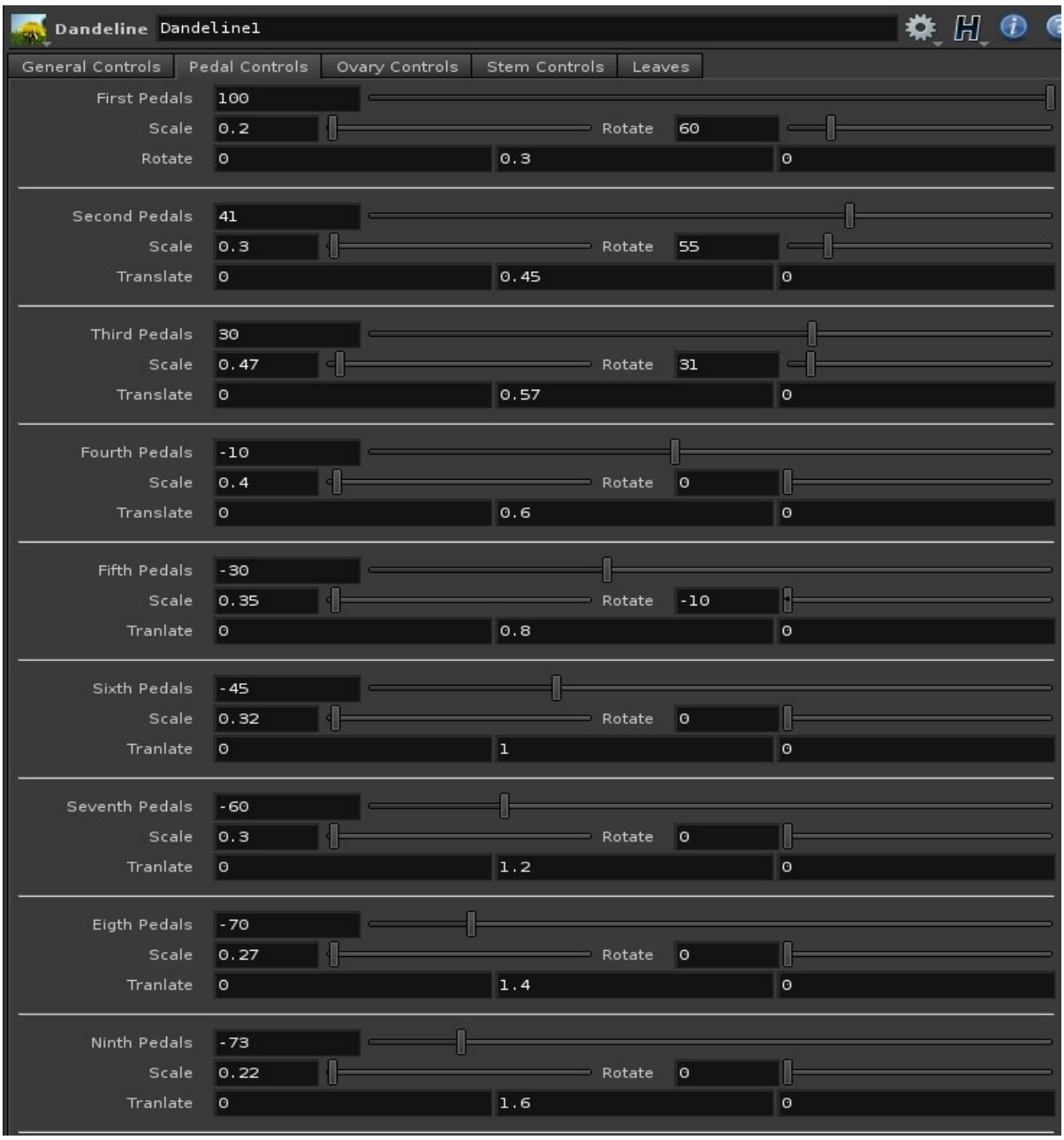


Fig Pedal Controls

This tab, has individual controls over the ten rows of pedals. In every row of pedals the user is given the following controls:

First Pedals: Controls the amount of bending of the row of pedals

Scale, Translate, Rotate: Scales, translates. rotates the row of pedals

Ovary Controls Tab

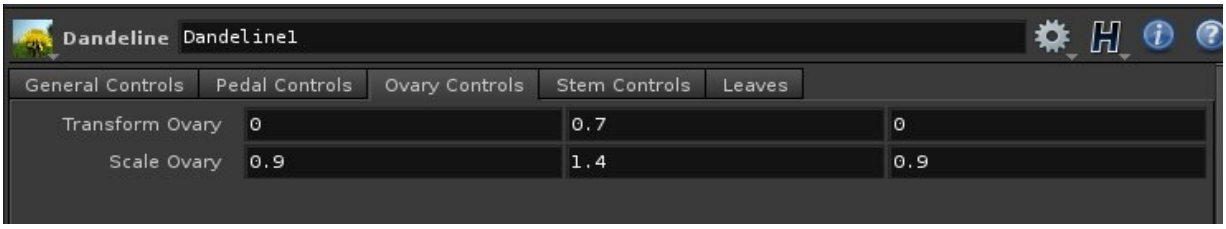


Fig Ovary Controls

Transform Ovary: Transforms the ovary

Scale Ovary: Scales the Ovary

Stem Control Tabs

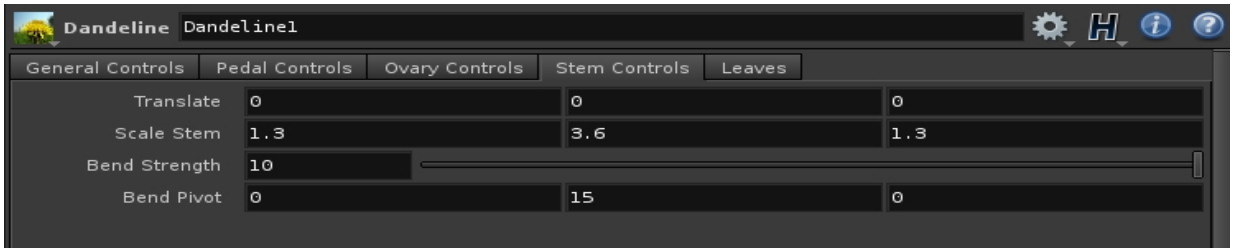


Fig Stem Controls Tab

Translate: Translates the stem

Scale Stem: Scale sthe stem

Bend Strength: How bended the stem is

Bend Pivot: Define the pivot of the stem

Leaves Tab

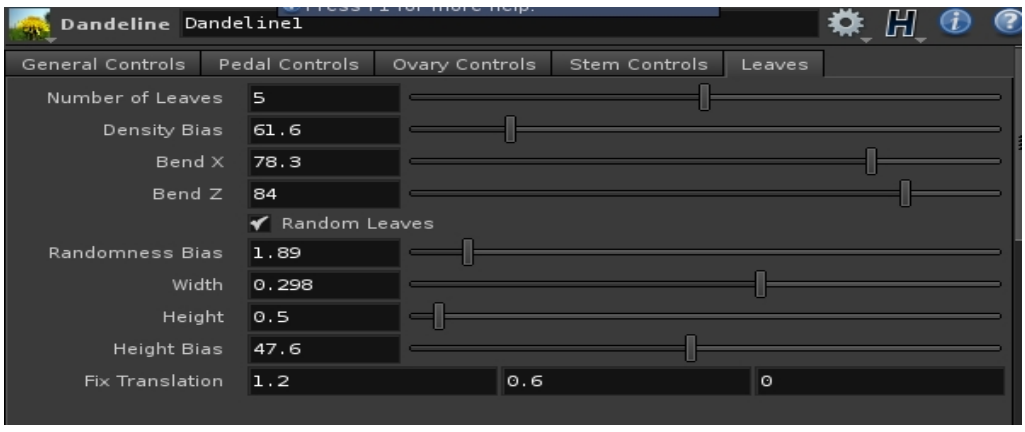


Fig. Leaves Tab

Number of Leaves: the user can define the number of leaves

Density Bias: How dense they are

Bend X,Z: Bends the leaves in X and Y position

Randomness Bias: How random the leaves are

Width: Width of the leaves

Height: Height of the leaves

Fix Translation: In case the user needs to fix some translation errors in the leaves is provided with this option.

The dandelion was exported as an Obj file from Houdini and imported into Maya. It was then animated using a bend deformer for the stem and a lattice deformer turned into a soft body for the pedals.

Melting Geometry

Overview

Our aim was to create a visually interesting melting effect and produce a digital asset than could be used in many cases for melting geometry. It also had to be efficient in terms of speed but mostly the visual outcome had to be good so it could be used in production. In order to find the right method to use we had to do a lot of tests and try different methods.

Smooth Particle Hydrodynamics

Our first approach was to use Smooth Particle Hydrodynamics(SPH) for the task. Using SPH we would have the ability to use the attributes of elasticity and viscosity. That would give us a nice looking result. One of our initial problems were that when we were converting the polygon surface to a particle fluid surface, the statue was loosing details. So we had to go at high levels of particle density.

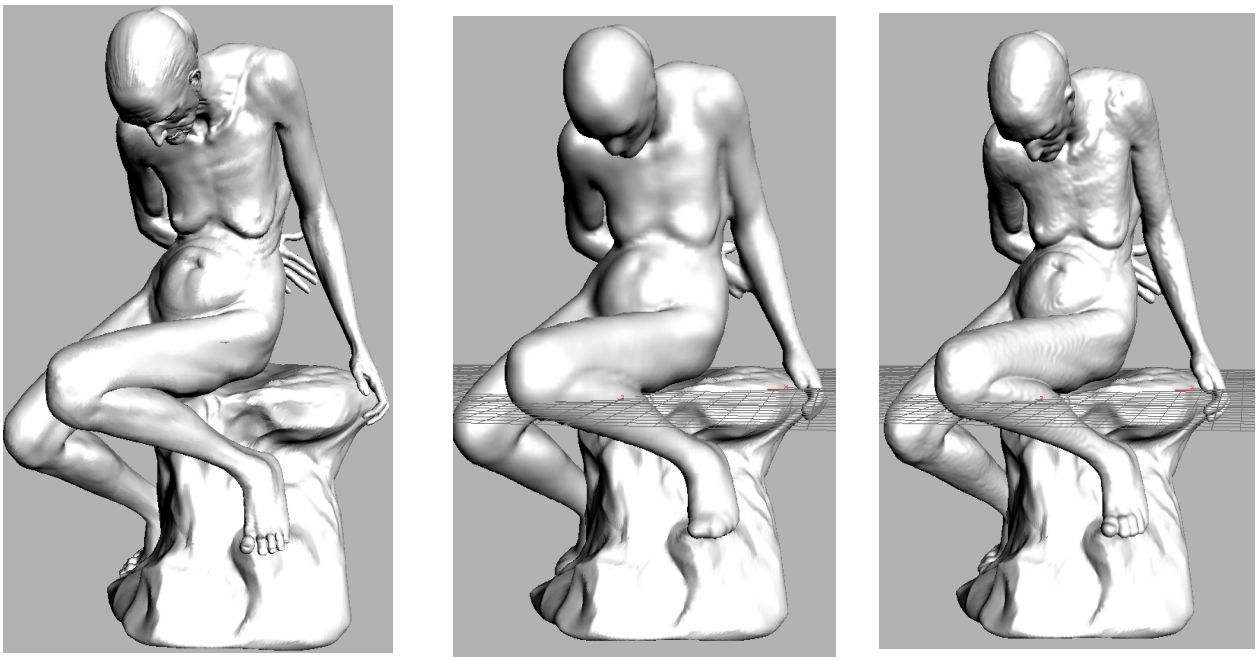


Fig a) Original Statue, b)Statue with 0.1 particle Density and c) statue with 0.03 particle Density

Although we were getting good looking results, at that level of particle density the simulations were very slow making the task almost impossible to finish we amount of time we had. Nevertheless the things we learned by studying the particle fluid solver, by diving in the network to get as more low level information as possible, they helped us a lot to adopt part of this knowledge later on in the process.

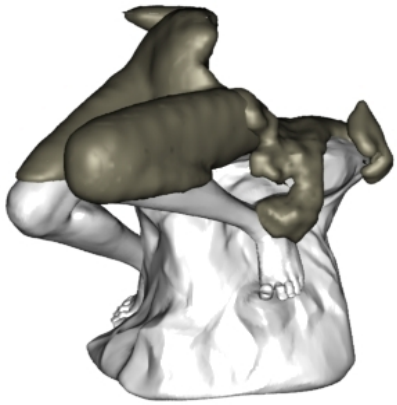


Fig. Test with Particle Based Fluids

Working in SOPs

Our second approach was to try working in Surface Operator (SOPs) trying to deform the surface based on the normals information. The normals were painted using the Comp SOP node(a node that is mainly used for hair) and then the geometry was deformed based on the color information and the normals.

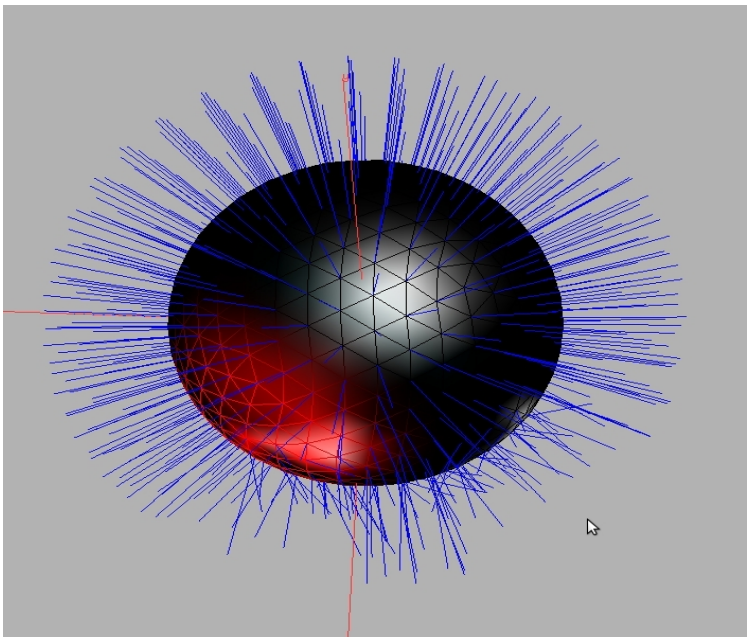


Fig Deformation according to normal information and color

Another node we could use to deform the geometry in SOPs was the Magnet SOP. Some metaballs instanced in particles would fit inside the statue using the Ray SOP an d than combining those two element we could get the melting effect. Nevertheless the method although quite cheap(in term of speed) it wasn't producing convincing results.

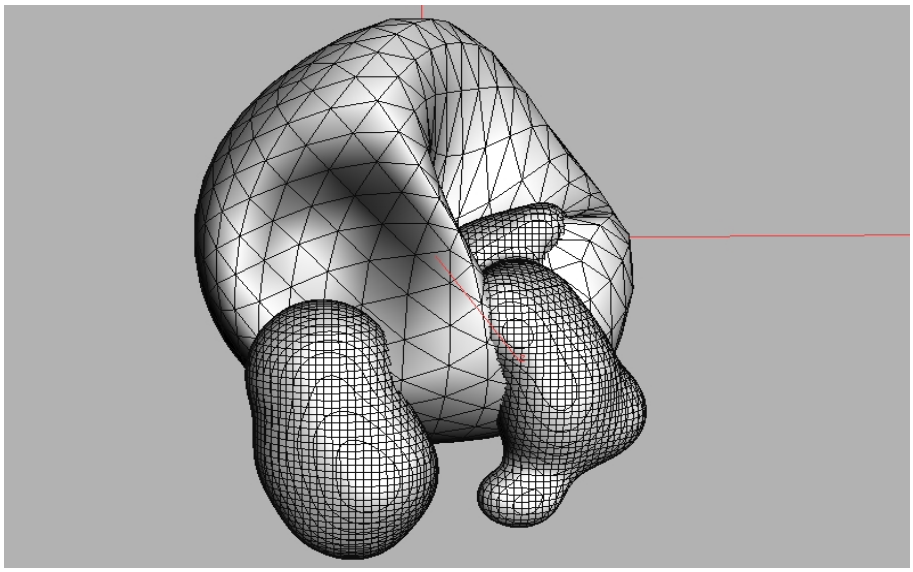


Fig Test result using SOPs

Working with POPs

Our next step was to try using POPs in order to achieve a cost effective result and at the same time would give us a visual outcome close to what we were having with the particle based Fluid Dynamics. The advantage was that we could use many of the features of the SPH without using any microsolvers but the disadvange was that we couldn't use the visco-elastic features. So we had to find a wake to reach as close to this particle behaviour as possible by only using the nodes available in POPs. Early tests showed us that we could count on this method and we proceeded to create the Melt Geometry Digital Asset.

Melting Geometry Digital Asset.

The melting digital asset works by following a volume with points, than surfacing the volume, using another geometry enables the selected particles checking for collisions. It has three inputs. The first input is the geometry the user wants to melt, the second input are the collision objects and the third the geometry who selects which particles are going to be moving. The options given to the user are the following:

Surface Attributes

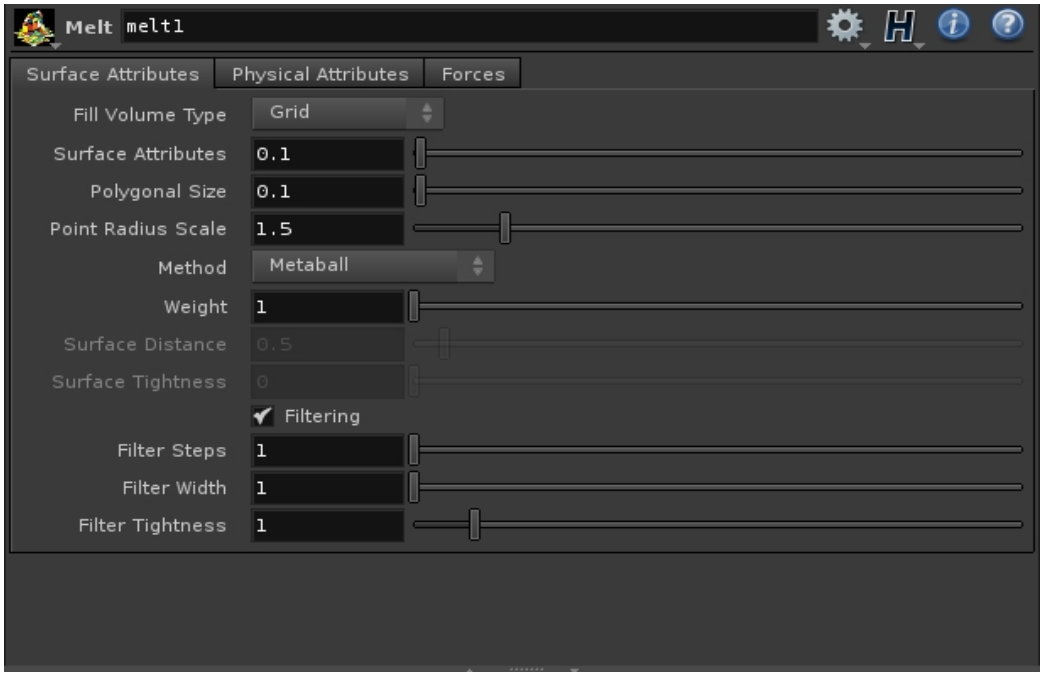


Fig Surface Attributes

Fill Volume Type: the user specifies the way the volume will be filled. It can either be tetrahedral(more dense volume) either grid.

Particle Density: This parameter controls the interaction distance between particles in the created Particle Fluid Object.

Polygonal Size: The polygon size to use when polygonizing the surface.

Point Radius Scale: Each particle in the input has associated with it a radius of influence in which it can affect the surface which is established by each particle's "pscale". Pscale scales the particle's radius of influence. Increasing this parameter causes each point on the generated surface to be sampled from a larger set of a particles and decreasing it causes each point on the generated surface to be sampled from a smaller set of particles.[3]

Method: Average position or Metaball

Weight: The weight of the metaball

Surface Distance: Controls the thickness of the surface around the particle field, This parameter is scaled internally by the Point Radius Scale parameter.[3]

Surface Tightness: The tightness of the surface

Filtering: It is used to smooth the surface.

Filtering Steps: Filtering is an iterative process. The smaller the particle radius scale is, the the more iteration it needs to smooth the surface.

Filter Width: To filter the surface, values from each grid cell and some number of adjacent cells are blended. This parameter controls the number of grid cells involved in each blend operation.[3]

Filter Tightness: This parameter control how close the surface is to the particles that create the surface.

Physical Attributes

Fig. Physical Attributes Tab

Mass: The mass of the particles

Viscosity: Not having the ability to use the viscosity microsolver we had to find something that would make the particles – and furthermore the surface- like they are viscus. To do that we had to use an interact POP node Setting the particle radius to “Use particle size/instance size” the like Radius of the particle is equal to a sphere big enough to enclose the instanced geometry object [4].

Effect Radius: The applied force decreases slowly up close, quickly in the middle, and then slowly at the outside edge.

This value lets you scale the computed radius.

Cling: Cling is implemented as a force that acts to counter forces pulling the particle away from the surface[5].

Forces Tab

Fig. Forces Tab

The forces that are implemented are the usual forces that can be used in POPs. One thing to point here is that the tool works better with a high value of drag (for example 10).

The algorithm

Constants *geometry_to_melt*

melter

collision

Begin

Create a volume of points of the geometry to be melted

Emit in the first frame as many particles as the points generated.($FF==1, $NPT)

Stop all the particles

Group the particles that are included in the *melter* geometry and its animation

Preserve group

Enable all the particles that belong to the group

Set the collision as “slide on collision”

Set the mass according to user's preferences

Apply forces,drag and cling according to user's preferences

Apply viscosity according to user preferences

Adjust the speed limit according to user preferences

Set the pscale value to pscale*2 so as the surface is smoother

If (method == metaballs) **then**

enable metaball weight

set metaball weight according to preferences

else

enable surface distance

enable surface width

set surface distance according to preferences

set surface width according to preferences

If (filtering==1) **then**

enable filter tightness

enable filter width

set filter tightness according to preferences

set filter width according to preferences

End

Follows a description of the algorithm with screenshots:

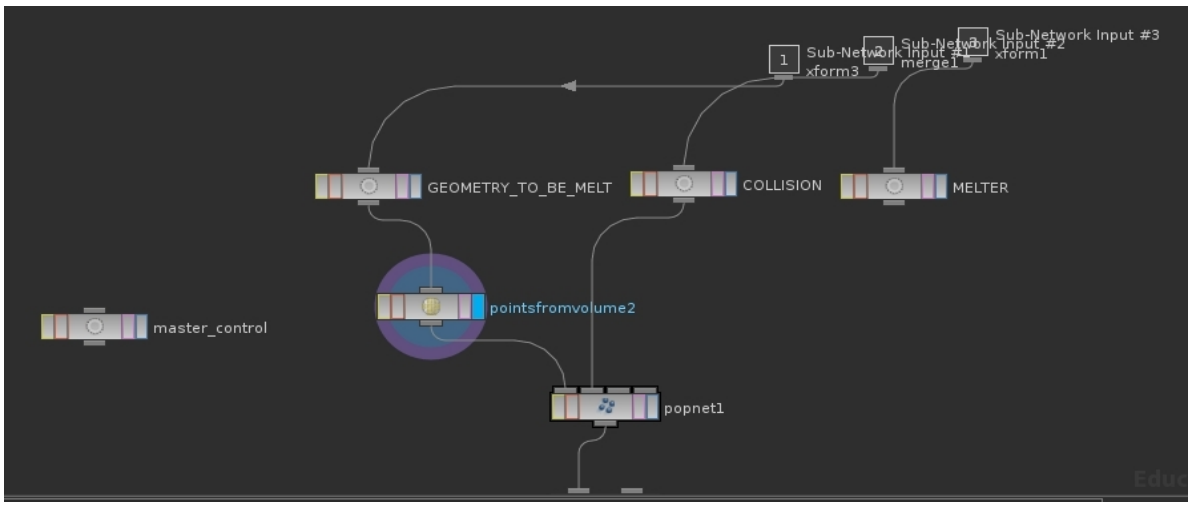


Fig the 3 inputs of the tool:geometry to melt, colision, melter

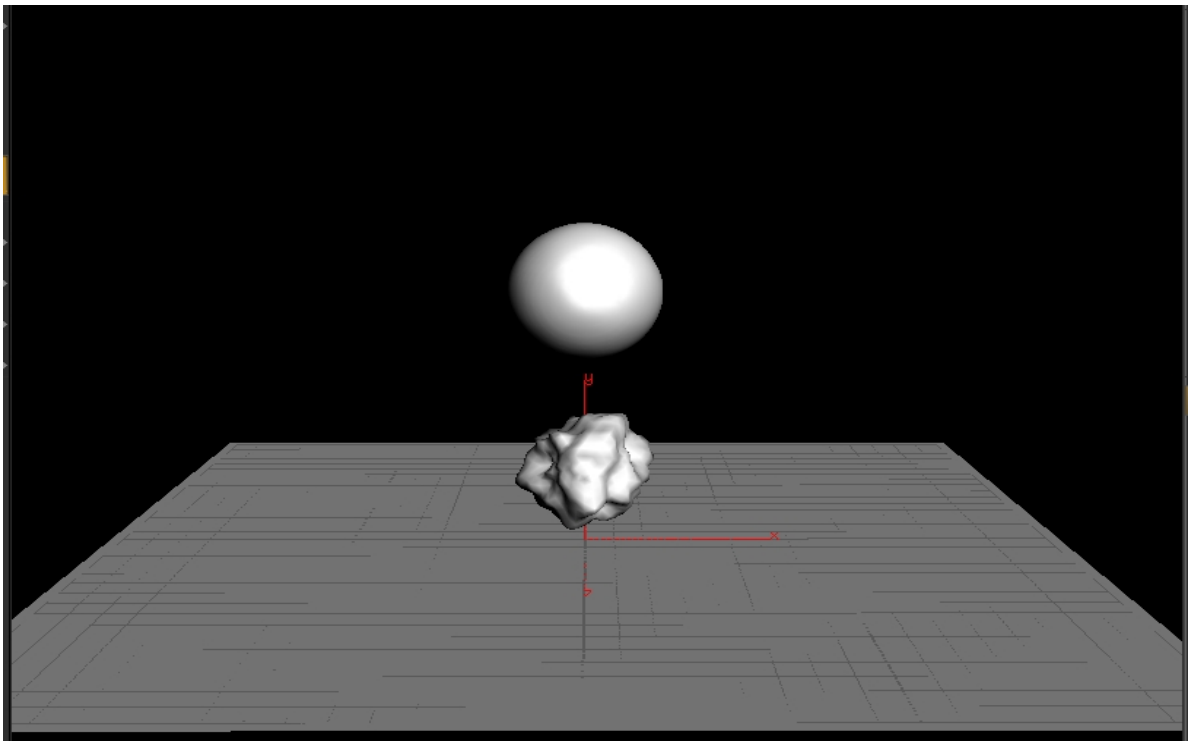


Fig. A sphere as the geometry to be melted and an object and the ground as collisions

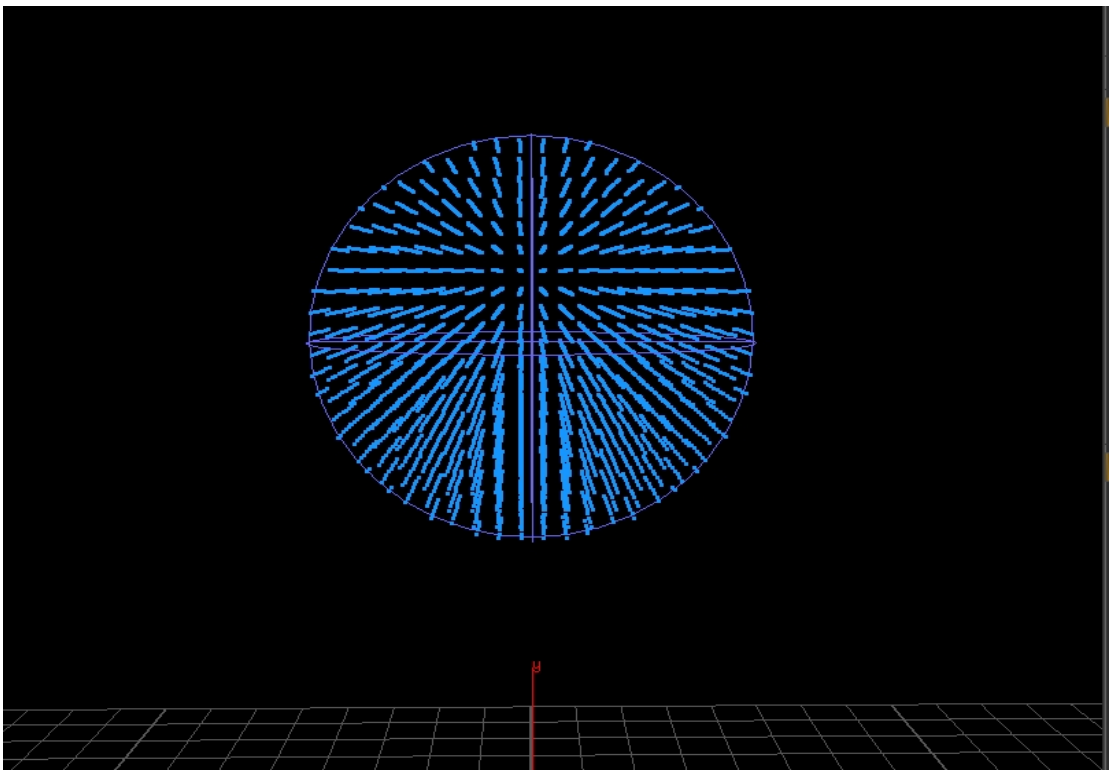


Fig. Filling a volume with points

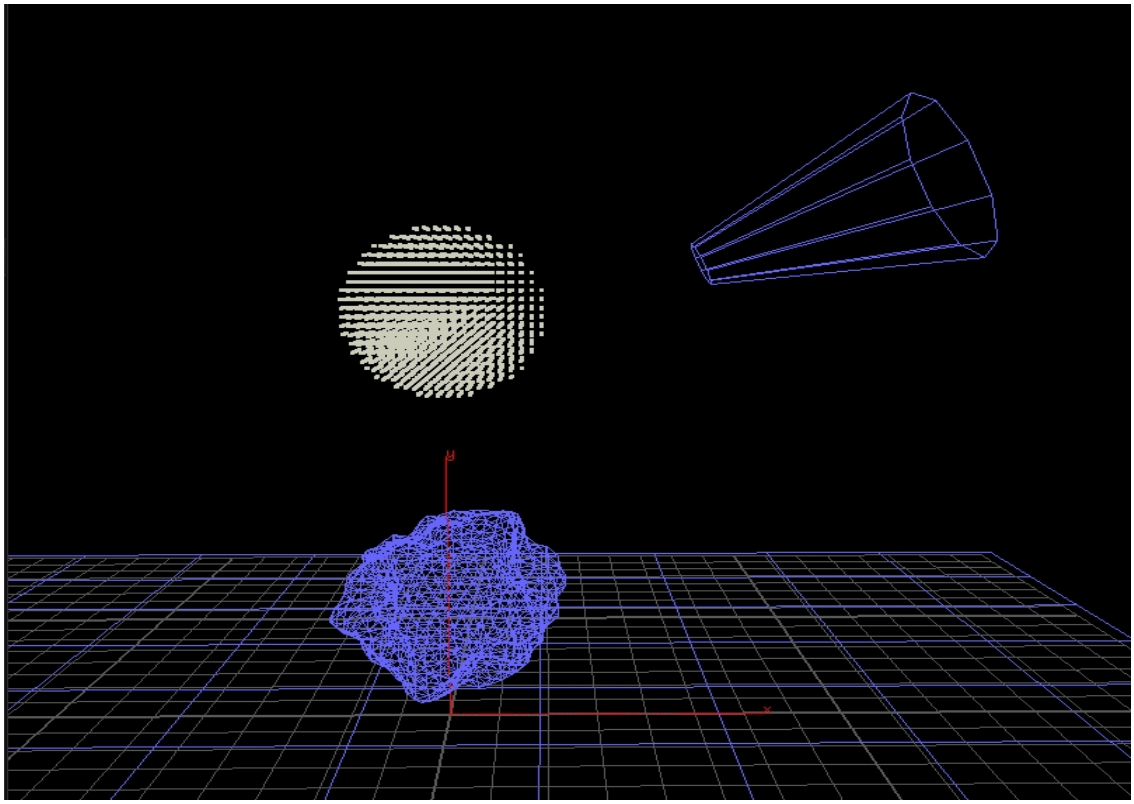


Fig. Emiting 1 particle from each point at the first frame, and using the melter to group the enabled particles

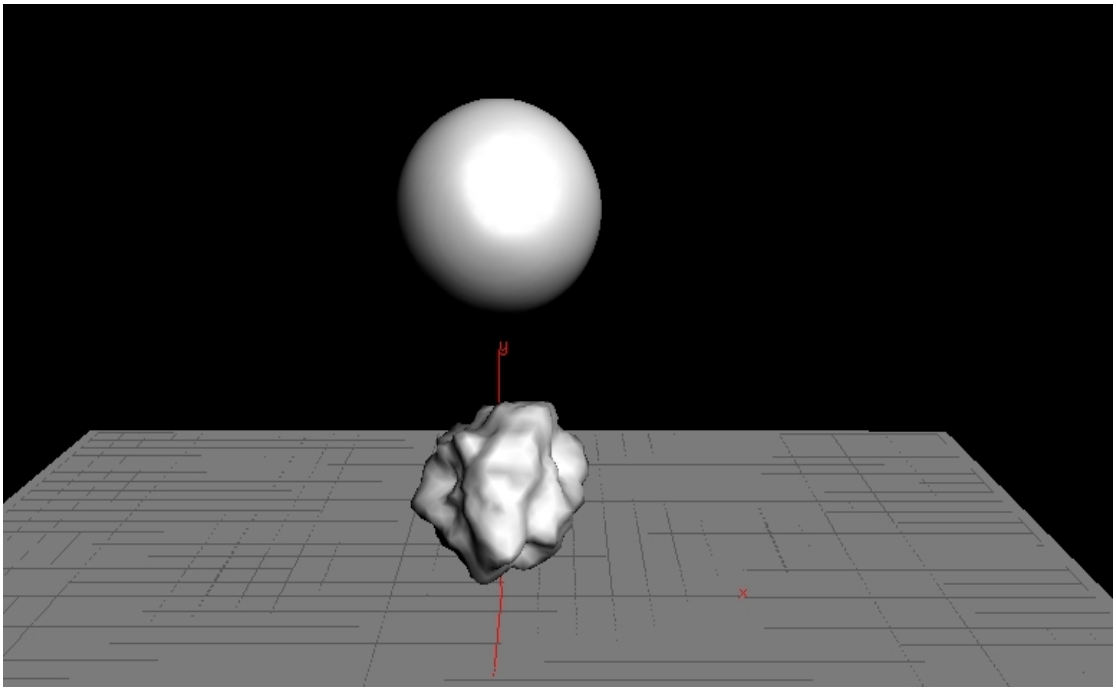


Fig the object, after is surfaced and filtered

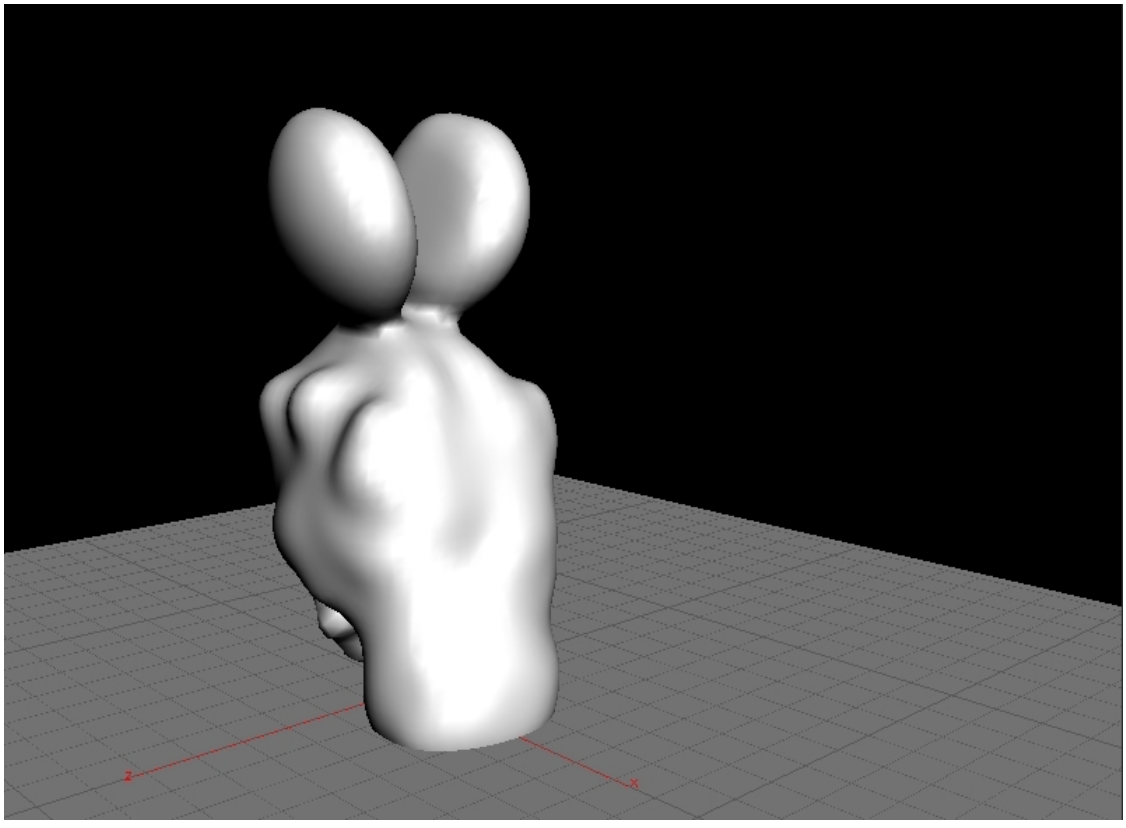


Fig The geometry melted

Melt Geometry – The Execution

Having the tool ready we were ready to use it in production for the piece. While using it we were getting feedback for the tool and we were constantly going back to improve it(either its functionality or its speed and efficiency).

In order to start simulating we created a plan of which parts were going to melt.and at which time. Our initial idea was to start melting the statue from its main body. For that reason we created an outline of the process. In order for it to look realistic we had to have a focus point. That focus point is the red area as it is seen in the following picture.

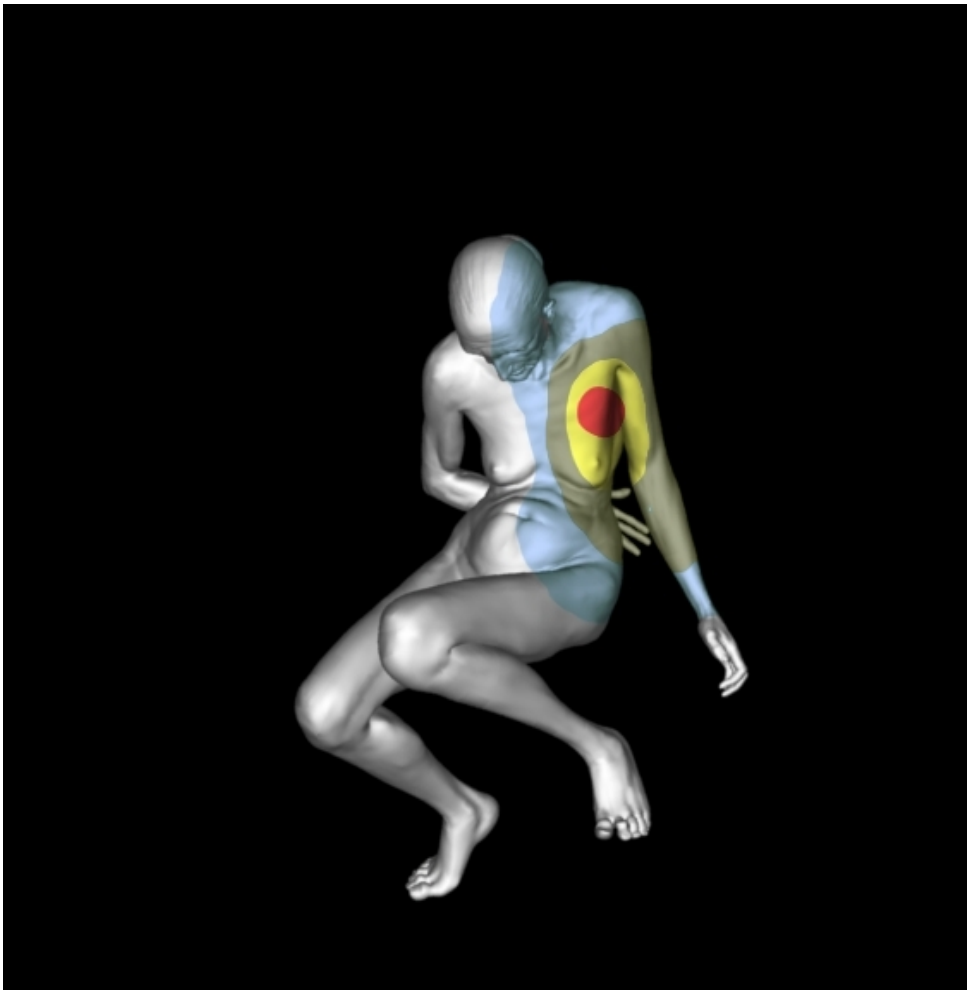


Fig. Areas to be melt starting from the red one.

However later on we had to change the plan to make it look more interesting and realistic. So we change the outline as it can be seen in the next figure.

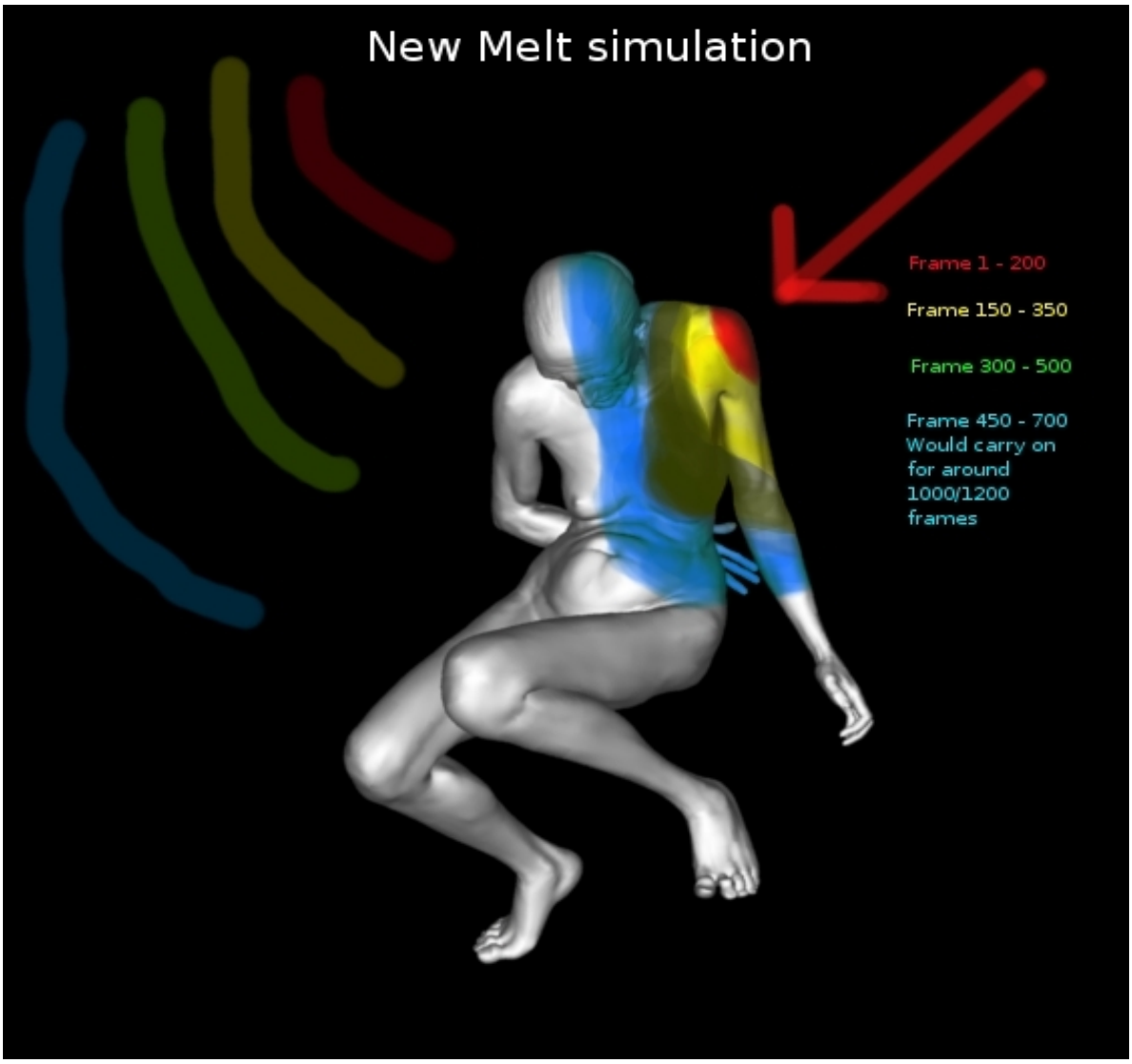


Fig The final outline

The way to achieve the dripping down and sliding to the surface behaviour was by making each “colour” as a separate geometry, converting it to a fluid surface and than using the next “colour” geometry as a collision surface. Then having the particles cached we were filling the whole statue with points and surfacing the simulated particles with the points all together so they seem that they are all at the same (implicit)surface.

The statue after being simulated was exported as rib archives from Houdini to RenderMan. The size of the files was big because of the large amount of geometry and we needed a lot of time for exporting them, so we had to be very careful and that everything was working good. Any mistake could seriously affect the end result of the project. Because the scene set up was done in Maya, initially we though about exporting the statue as FBX sequence in Maya. But the problem was that the FBX sequence cannot calculate the geometry properly when it comes to geometry that

its points positions are changing(or/and added) over the time.

Smoke

The smoke used in the project was done in Maya using fluid dynamics. In order to achieve the smoke emiting from the surface that's been melted and because we wanted to be precise and achieve a nice visual result, we exported these geometries as FBX sequences from Houdini and imported them to Maya. In order to match the cameras in maya we imported a .jpg sequence of the statue melting in maya and build a camera rig grouping the camera and the sequence together so we can match the position of the geometry melting and the geometry emiting. A short MEL script had to be written to solve a problem with the .jpg sequence:

```
string $file_name = ".jpg";
file1.frameExtension = $file_name;
```

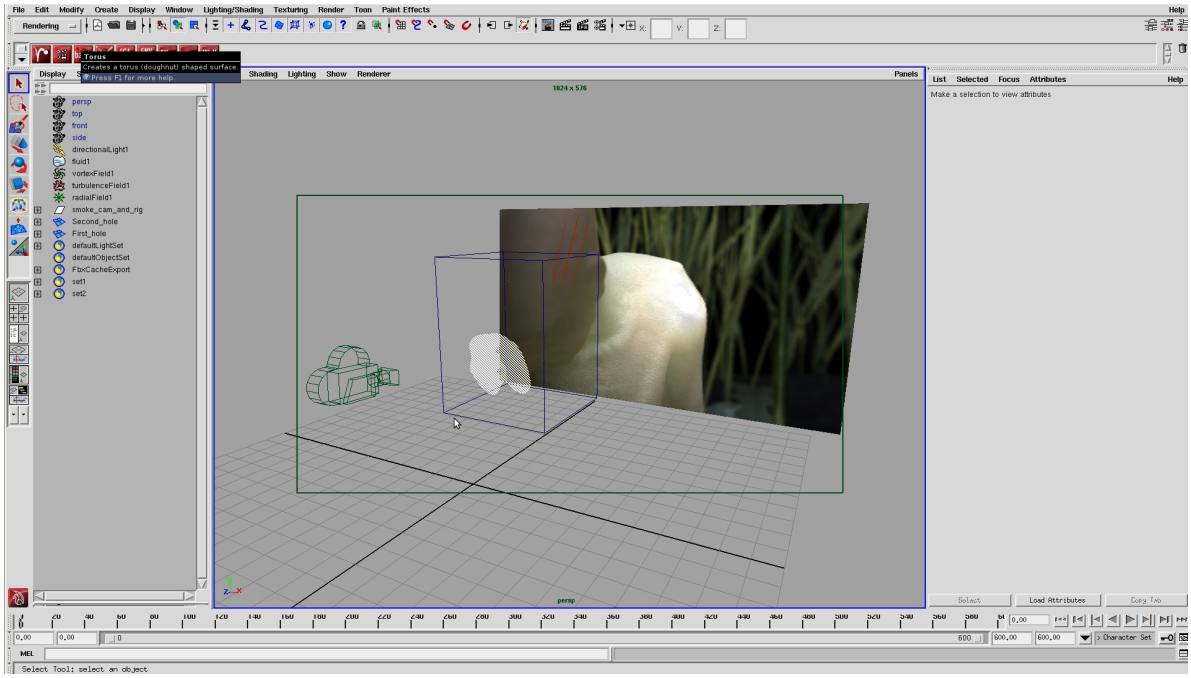


Fig Maya camera rig for matching the smoke

L-System Plants

Following our aim of experimenting with techniques through out the project we tried to create some secondary geometries for the garden. One of the things we tried was to create an L-systems plant. L-systems were conceived as a mathematical theory of plant development [6].

Premise : F(1)X

Rule1: X = F[-F[-(15)FJ][+(15)FJ]F[-(15)FK][+(15)FM]fFJ]F[+F[-(15)FJ][+(15)FM]F[-(15)FJ] [+(15)FM]FFJ]X

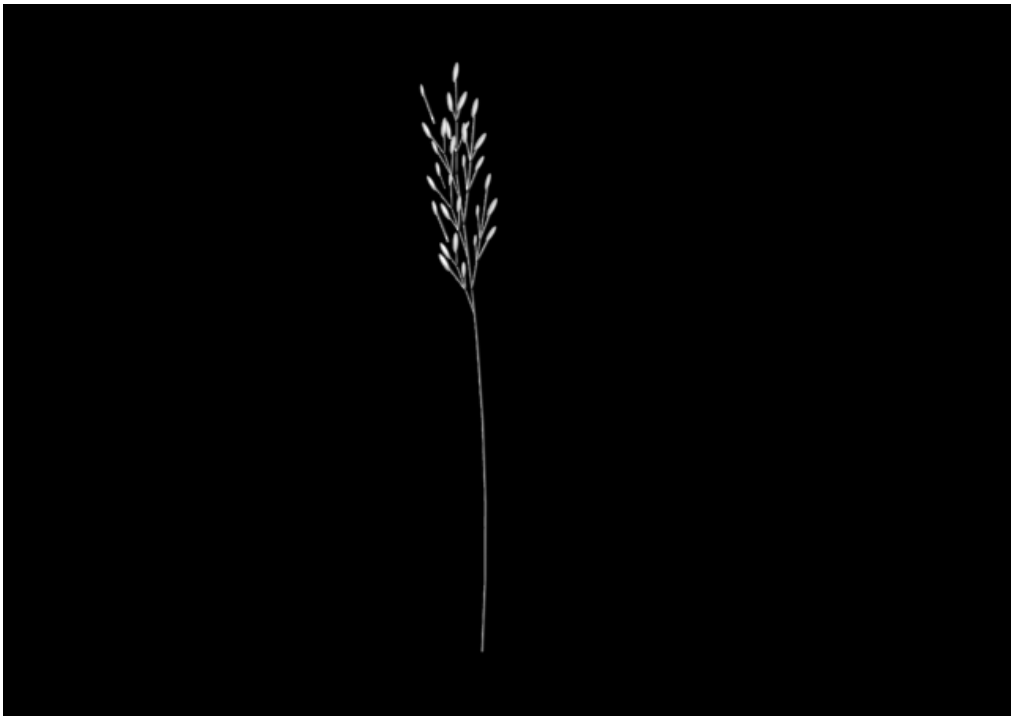


Fig. L-System Plant

However in order to get better results we had to study L-Systems more so we decided to leave L-Systems implementations.

Pipeline scene management RenderMan Python api

Geometry and effects are created and positioned in Maya and Houdini to allow them to match with each other in terms of scale and position they are then exported into RIB archives which can be read used RenderMan.

Many books and document-ion was refered to for information about RenderMan as it is difficult software to learn and use and has a massive range of features. These include :

PrMan docs, Jon Macey's on-line documentation [http://nccastaff.bournemouth.ac.uk/jmacey/Renderman/index.html], The RenderMan Shading Language Guide, The RenderMan Companion, Texturing and Modeling: A Procedural Approach, Advanced RenderMan: Creating CGI for Motion Picture, [http://www.fundza.com] Also the Escape studios RenderMan course-ware.

These are mentioned here as knowledge from all these sources has been absorbed during the course of the project to increases familiarity in using RenderMan and shaders as an Animation and rendering tool.

All of the rest of the scene management and scene descriptions is done using the Python RenderMan api. All geometry is loaded in with RIB archives(these would have been generated from Maya or RenderMan) which contain RIB data which describes the geometry. Shaders and parameters are applied to objects and translations to fix positions via python. Also the frame number is taken into account when reading in the RIBs to allow animation by loading in associating RIBs file.

An example of a function that loads in a object fixes translations and assigns shaders and allows a new rib to be loaded in each frame. Also as the ribs are loaded in in a loop it allows multiple objects to be loaded in with random parameter assigned to each individual shader. That is assigned to the object.

```
def Grass_forground(ri,frame,point_cloud_check,out_filename,rendertype) :
        random.seed(20)
        ri.TransformBegin()
        ri.Scale(-1.0,1.0,1.0)

        ri.TransformBegin()
        ri.TransformBegin()
        for i in range(1, 19):
                ri.AttributeBegin()
                ran_diffuse=random.uniform(0.9,1.1)
                ran_opacity=random.uniform(1,3)
                ran_tex=random.uniform(1,3)
```

```
        ran_diffuse_front=[random.uniform(1.0,1.1),random.uniform(0.8,0.9),random.uniform(0.8,0.9)]

        ran_diffuse_back=[random.uniform(1.0,1.0),random.uniform(0.9,1.1),random.uniform(1.0,1.1)]
                ran_specular=random.uniform(1.2,1.3)

        ran_spec_colour=[random.uniform(0.4,0.5),random.uniform(0.5,0.6),random.uniform(0.5,0.6)]
                if (rendertype == 1) :
                        ri.Surface("matte")
                if (rendertype == 2) :
                        ri.Surface("./shaders/new_grass/new_grass_surface_shader_v3_r1",{"float
Ks":ran_specular, "color specularcolor":ran_spec_colour, "float Kd":ran_diffuse,"float Ka":
[1.0],"color front_colour":ran_diffuse_front,"color back_colour":ran_diffuse_back,"string
texture_front":["./sourceimages/grass/grass_front_high_v1.tx"],"string  texture_back":
["./sourceimages/grass/grass_front_high_v%01d.tx" %(ran_tex)],"color opacity":[1.0,1.0,1.0],"float
roughness":[8.5], "float max_size":[5],"string  texture_opacity":
["./sourceimages/grass/grass_opacity_high_v%01d.tx" %(ran_opacity)],"float Kenv":[0.2],"string
EnvironmentMap":["./sourceimages/hdr/Meadow_non_float.tx"],"float oren_roughness":[5.0]})
                ri.Displacement("./shaders/grass/grass_displacement_shader_v1_r3",{ "float
freq_layer_one":[100], "float freq_layer_two":[40], "float depth_one":[0.025], "float depth_two":
[0.04]})
                ri.Attribute("displacementbound",{ri.COORDINATESYSTEM:["object"],"uniform float
sphere":[0.2]})
                ri.Attribute ("identifier",{"name": "grass_foreground"})

                ri.ReadArchive("./Archive/grass/new_foreground_grass/grass_sperate/OCgrass_piece_
%02d_.%04d.rib" %((i), (frame)))
                ri.AttributeEnd()
        ri.TransformEnd()

        ri.TransformEnd()

        ri.TransformEnd()
```

All of the rendering lighting, camera movement, order of events, shader loading and numerous other tasks are handled through Python shader writing is done with the Renderman Shading language. Apart from the modeling, grass animation and effects(which were completed in Houdini and Maya and made into RenderMan usable formats). This was a challenging task to complete as the Python documentation is very limited. A thorough understanding of RenderMan and lighting and rendering techniques was needed, the drive to experiment and to try to go about things in a way to makes things work for the production. As the pipeline was an unusual one and was more low level than a conventional animation production a great deal was learnt about the core of lighting and rendering and pipeline issues. A lot of useful tricks were discovered along the way and as every was done through code and scripts complete control was available and certain tasks were made more achievable. and the documentation should be of great value with the accompanying python scripts for anyone wanting to produce animation and rendering in this fashion.

Shaders RenderMan Shading Language

Numerous shaders needed to be developed to meet the needs of the project because of the complex variation of objects in the environment and the unusually juxtaposition of different

materials. Also many objects needed the power of procedural shading methods to give them life and to increase there realism.

Shaders are wrote in the Renderman Shading Language (RSL) which has a C like syntax which some added functionality and built in data types (points,normals,vectors) to help with the shader developing process. Shaders need to be compiled and turned into machine readable code. The shading language extension for the files is .sl when complied this turn into .slo files which are object code which then allows the shaders to be executed by RenderMan.

Individual shaders are essentially small sub-routines (functions) that allows users to extend the creative possibilities of the renderer; allowing endless ways of controlling the appearance of a 3D scene through the use of custom shaders. The only limit is the imagination, ability to write new, or adapt existing shaders and the creative flare at adjusting the parameters that control the visual outcome of the shader. [15]

Some pre-made shaders are used in the project as they suited our needs already, and because of the scale of the project it would be insensible and non-logical to write every shader for every object when it isn't necessary.

Shaders are divided into six types

- Light source shaders
- Surface shaders
- Displacement shaders
- Volume shaders
- Transformation shaders
- Imager shaders.

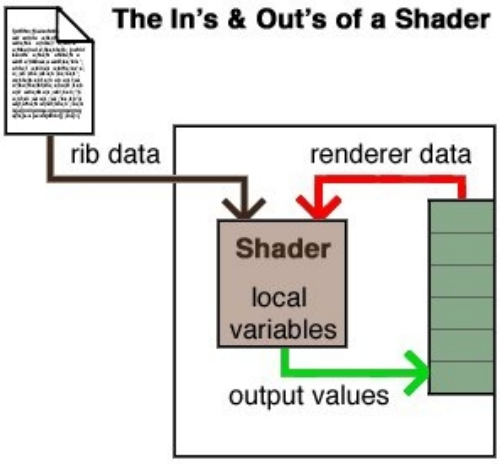


Fig (http://www.fundza.com/rman_shaders/overview/overview.html)

Only custom surface, displacement and light shaders were used/modified in the project more information on these are on the detailed analysis of the shaders.

Shaders calculate specific values at regular intervals (dependant on shading rate)across the surfaces being shaded by a renderer. RenderMan sub-divides each object in a 3D scene into a fine mesh of micro-polygons. Which pixel value is defined from the shading calculations of the

shader(which ever they might be a complex algorithm for lighting and reflections etc, or a simple maping of a colour).When Renderman is processing a rib file, or some other source of rib information, it makes data available to the shader so that the shader can calculate specific values eg. Light, camera positions etc. For example a displacement shader, calculates a displaced location and orientation for each micro-polygon by modifying the surfaces P value which is the current point being shaded. A surface shader, determines the apparent colour and opacity of each micro-polygon. [15]

The Shading Language data types:

- float
- string
- point, stores the xyz coordinates of a location in 3D space,
- normal, stores the xyz coordinates of a surface normal,
- vector, stores the xyz coordinates of a vector,
- color, represents the color and opacity of a light source or a surface,
- matrix, a list of 16 floats.

Integers are not supported by the language. [15]

Variables accessible from with inside the surface and displacement shaders that are given from the object shaded or from the scene camera view etc. These are mostly essential for the shader to have access to in order to calculate the surfaces appearance.

point P
Position of the point you are shading, Changing this variable displaces the surface

normal N
The surface shading normal (orientation) at P. Changing N yields bump mapping The true surface normal at P. This can differ from N; N can be overridden in various ways including bump-mapping and use-provided

normal Ng
normals, but Ng is always the true surface normal of the facet your are shading

vector I

The incident vector, pointing from the viewing position to the shading position

color Cs Os

The default surface colour and opacity, respectively

float u, v

The 2D parametric co-ordinates of P (on the particular geometric primitive you are shading)

float s, t

The 2D texturing co-ordinates of P. These values can default to u, v but a number of mechanisms can override these values

vector dPdu

The partial derivatives (tangents) of the surface at P

vector dPdv time

The time of the current shading sample

float du dv

An estimate of the amount that the surface parameters u and v change from sample to sample

vector L , color Cl

These variables contain the information coming from the lights and may be accessed from inside illuminance loops only

color Ci, Oi

The final surface colour and Opacity of the surface at P. Setting these two variables is the primary goal of a surface shader

[RenderMan Prman doc, Shader Execution Environment]

With access to these values and through use of your own algorithms it is possible to do an infinitesimal amount of operations on to surfaces, and to produce rich and detailed photo-realistic

results. Unless specified as being wrote developed or modified shaders are not are own.

Grass shaders

A shader was developed for the grass as it is a prominent objects in the piece and a variation was needed between the grass blades and a high degree of physical accuracy of the properties of the grass had to be reproduced or made to look convincing to the viewer.

As grass was studied quite closely in the pre production stage we had a good idea of the properties needed to create convincing grass. Details of all the important components of the grass surface and displacement shaders follow.

Random holes to show-ware & parts eaten controlled by modifying the opacity

Influenced by the RenderMan spatter shader which ships with RenderMan as a standard shader. Noise is created in shader space using the 3d point position P. Controlling its influence by a scale factor and checking if the value of the current influence noise on a point is greater than a threshold if its is it is applied an opacity value 0.0 else it is kept its default value of 1.0 opaque. There is also a similar check with an offset to the threshold which modify the colour value of the point. This gives the effect of having a hole on the surface with a surrounding colour gradation.

Code segment of grass surface shader

```
// sets holes opacity
opacity_holes = color(1.0,1.0,1.0) * opacity;
for (size=1; size<=max_size; size +=1)
{
      hole = noise(transform("shader",P)*scalefac);
      if (hole > threshold)
      {
      opacity_holes = hole_opacity;
      break;
      }
      if (hole > (threshold+threshold_adjust))
      {
      color_holes = color_holes_value;
      break;
      }
scalefac /= 2;
}
```

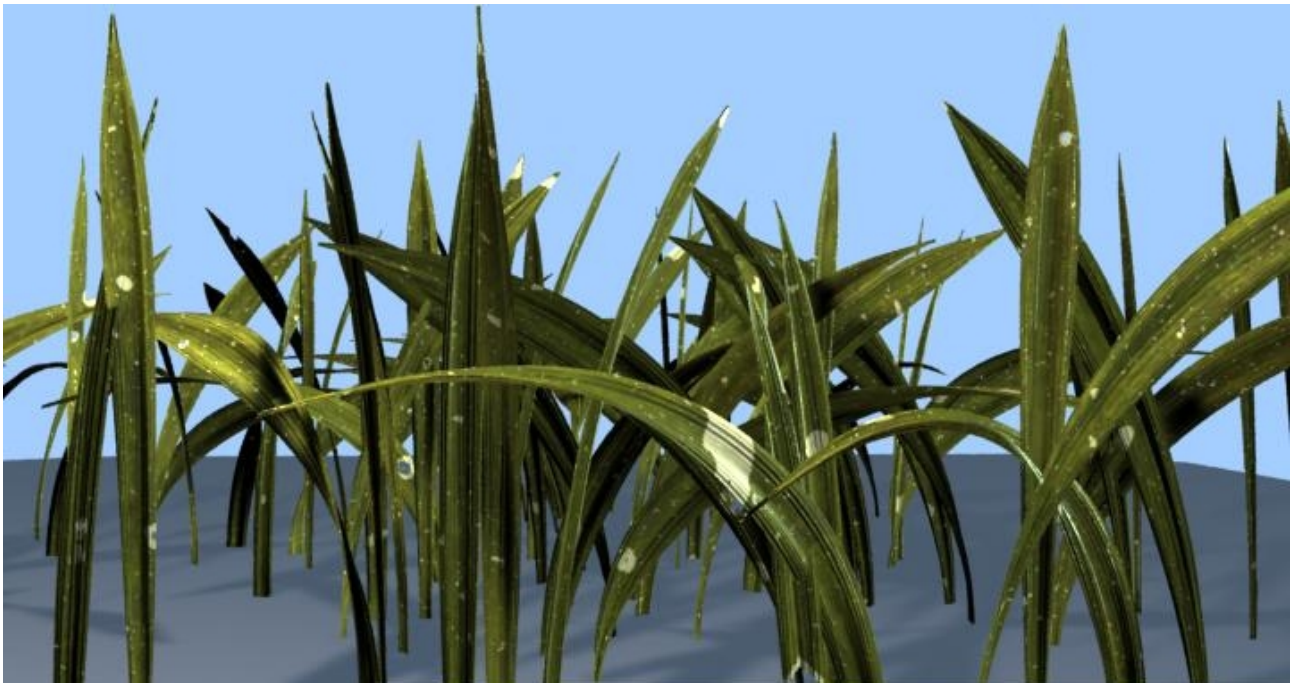


Fig. Example of shader transparency's based on noise

Unfortunately this feature could not be used in the actual production of the animation as the noise gave many issues because of issues of getting noise values in certain spaces. As the grass was animated and there was camera movements the transparent holes would always flicker because of the noise threshold checking, despite testing getting the noise in different combinations of spaces, shader, object,world, there was no right combination found. The shader could be fine if there was no camera move and animated but not together it was either one or the other. This was a very importing learning process in developing shaders for animation production. As the shaders may look fine in a still but many problems may occur during motion such as aliasing problems and texture swimming, unwanted noise etc.

A solution to avoid this problem and to still get the same perhaps better results was to use textures to define the opacity of the grass.

The opacity is define values based on a user created texture to define the alpha of the object based on U,V coordinates.

Code segment of grass surface shader

```
// sets holes opacity
opacity_holes = color(1.0,1.0,1.0) * opacity;
if(texture_opacity != "")
{
    opacity_holes = texture(texture_opacity);
}
// set opacity
```

```
out_opacity = opacity_holes;
```

Diffuse and pattern calculation

Initially noise patterns were used for the bases of the shader but these gave unsatisfactory results when viewed from close up and because of the shear amount of grass the noise patterns were noticeable. In stead a combination of photographic and hand painted texture maps were used instead as these gave a greater realism for the surface as the grass needed to hold up to close ups.

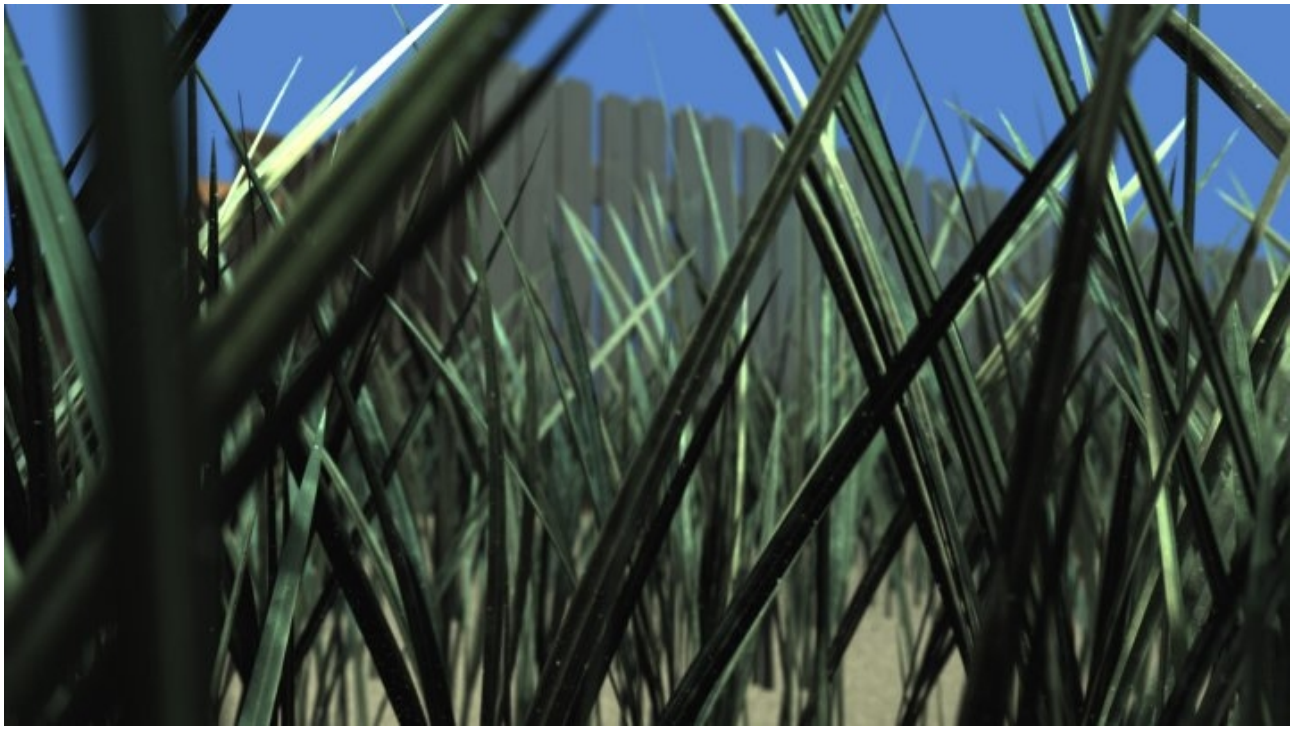


Fig. early shaders

Also when using the RenderMan txmake facility to convert textures into the RenderMan texture file format (.tx) you can have the added functionality of mip mapping having the Textures stores in various resolutions. So Renderman picks the appropriate resolution texture based on the objects distance from the camera. Which saves a lot of memory and is clever optimisation to have. Using the texture filter sinc when converting to .tx was found to be the best to use to retain the most detail and reduced aliasing of the textures.[insert tex example]

A more complex diffuse model was used to allow the model to hold in close up scenes and to give a more organic feel to the grass. Oren-Nayer diffuse model was used.

A nextra diffuse calculation of the non facing normal of the object to fake the effect of subsurface scattering was also implemented. This does too lighting calculations on the object one for the diffuse part of the shader. The diffuse for the front and back of a surface are calculated separately by feeding the front facing normal in one calculation and the back facing normal in to the other. Different Textures can also used to give an even greater realism. Then some of the values of the back are added to the front(visible part of the surface) which gives the illusion of light passing through.

Code segment of grass shader

```
/// Load textures if available
// texture data
if(texture_front != "")
{
    texture_front_colour = texture(texture_front);
}

if(texture_back != "")
{
    texture_back_colour = texture(texture_back);
}

/// Calculates the diffuse for both sides
//front_face_diffuse = diffuse(Nf) * ((texture_front_colour += color_holes)) *
front_colour;
front_face_diffuse = LocIllumOrenNayar(Nf,V,oren_roughness) *
texture_front_colour * front_colour;

//back_face_diffuse = diffuse(-Nf) * ((texture_back_colour += color_holes)) *
back_colour;
back_face_diffuse = LocIllumOrenNayar(-Nf,V,oren_roughness) *
texture_back_colour * back_colour;
```

The non facing and front facing sides are also modified by a colour multiplier. There is also a different component of the diffuse applied to each side of the object surface this works well on single planed geometry which gets viewed from both sides to give variation ideal for grass.

```
/// MODIFY ALTERNATE SIDE
/// ////////////////////////////////////////////////////////////////////////
//////////////////
// front side
if (Nf.N>0)
{
       // Final colour applied
       //out_diffuse = mix(front_face_diffuse,back_face_diffuse,0.7);
       out_diffuse = front_face_diffuse + (back_face_diffuse * 0.2);
}else
// back side
{
        // Final colour applied
       out_diffuse = (front_face_diffuse * 0.9) + (back_face_diffuse * 0.3);
       //out_diffuse = mix(back_face_diffuse,front_face_diffuse,0.7);
```

```
}
```

Specular environment reflections

The Grass needed to have some reflective and specular quality, but initial tests into using ray tracing reflections proved to be much to slow particularly on a large scale.

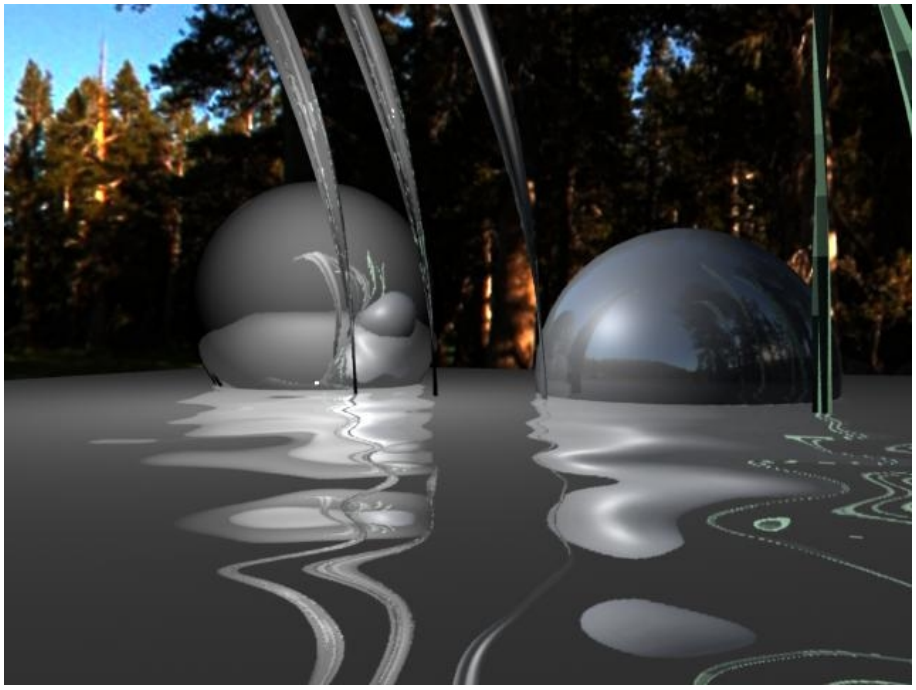


Fig. Specular environment reflections

So an alternative solution had to be found. As when studying grass it was found that they mainly reflected diffused elements of the sky environment weight towards the tint of blue. With also standard while specular high lighting when exposed directly in strong light sources.

So an environment map was used and loaded in with an environment map loading function which can load in latitude longitude environment maps. This is then multiplied with a weighting of its influence to the standard RenderMan specular function. To give the result.

Code segment from grass shader

```
color out_environment = color (1.0,1.0,1.0);
if(EnvironmentMap != "")
{
      out_environment = envmapping(N, I, EnvironmentMap, envspace, Kenv);
}
...
(specularcolor * out_environment * Ks * specular(Nf,V,roughness))
```

Horizontal lines displacement

Displacement was used to give the illusion of lines and ridges runing up the surface of the grass and to add greater detail of geometric believability to the object. It was developed with influence and modification of the RenderMan cloth shader.

Noise is calculated in one direction s and used to generated vertical repeating tiling lines which are used to modify the surface to produce interesting effect, with good results.

```
d_one = noise(freq_layer_one*s) -1;
d_two = noise(freq_layer_two*s) -1;

PP = transform("shader",P);
sN = normalize(ntransform("shader",N));
P = transform("shader","current",PP + sN*((depth_one*d_one)+(depth_two*d_two)));
```

Parameters of the shaders which can be modified by the user

Can be randomised on a per object basis by imputing random values with some sort of random changing value based on each iteration of what number object is being loaded. Very useful for large amounts of grass. The shader is adaptable for shading many other organics surfaces such as plants and flowers, and was used for this purpose in the project. But it is designed mainly to create realistic grass which is relatively cheap to render when dealing with large amount of objects.

A header files is included which contains additional shading model functions and environment maping functions wrote by others as they are tried and tested but don't come as default with RSL.

```
#include "shading_models_v1_r1.sl"
```

Surface

```
surface new_grass_surface_shader_v3_r1(
/// Contol multiplers
float Ks=1.0, Kd=1.0, Ka=1.0;

/// for opacity calulations
float hole_size = 0.01;
float max_size = 10.0;
float control_threshold = 0.8;
float threshold_adjust = -0.05;
color opacity = color (1.0,1.0,1.0);
color hole_opacity = color (0.0,0.0,0.0);
color color_holes_value = color (0.8,0.8,0.8);
string  texture_opacity = "";

/// for diffuse/pattern calulations
float oren_roughness = 5.0;
color   front_colour = color (0.8,0.9,0.7);
color       back_colour  = color (0.7,0.9,0.8);
```

```
string  texture_front = "";
string  texture_back = "";

/// for specular
color specularcolor = color(1.0,1.0,1.0);
float roughness = 0.2;

/// for environment reflection calulations
float Kenv = 0.3;
string EnvironmentMap = "";
string envspace = "world";
)
```

Displacement

```
displacement grass_displacement_shader_v1_r3(
float freq_layer_one=300;
float freq_layer_two=80;
float depth_one=0.02;
float depth_two=0.04;
)
```

Finally the summed out colours and opacity as computed in the shader

```
 Oi = out_opacity * opacity;
 Ci = Oi * (((Kd* out_diffuse) + (Ka*ambient())) +(specularcolor *
out_environment * Ks * specular(Nf,V,roughness)));
```

Rocks and ground shaders

Shader was developed to suit the need for the rocks and the ground in the piece the shaders needed to be able to create a wide to be able to have variation as it shades prominent and different objects.

A high degree of physical accuracy of the properties of the ground and rocks had to be reproduced or made to look convincing to the viewer.

Rocks were studied quite closely in the pre production stage as this was needed to create convincing shader. Details of all the important components of the ground and rock surface and displacement shaders follow.

Pattern texture generation

The RenderMan marble shader was looked into modified and parts were used to create a marble veining on the rock although subtle effect in the end it does add procedural variation.

```
/// for marble viening
point PP;
float width, cutoff, fade, f, turb, maxfreq = 16;

PP = transform("shader", P) * veining;

width = (max(sqrt(area(PP)), 1e-7));
//width = PP/2;
cutoff = clamp(0.5 / width, 0, maxfreq);

turb = 0;
for (f = 1; f < 0.5 * cutoff; f *= 2)
{
  turb += abs((noise(PP * f) * 2 - 1)) / f;
}

fade = clamp(2 * (cutoff - f) / cutoff, 0, 1);
turb += fade * abs((noise(PP * f) * 2 - 1)) / f;

turb *= 0.5;

layer_color = (spline(turb,
                   color (0.9, 0.9, 0.9),
                   color (0.8, 0.8, 0.8),
                   color (0.5, 0.5, 0.5),
                   color (0.4, 0.4, 0.4),
                   color (0.6, 0.6, 0.6),
                   color (0.3, 0.3, 0.3),
                   color (0.2, 0.2, 0.2),
                   color (0.1, 0.1, 0.1)))*color_multi_one;
layer_opac = layer_color;
surface_color = ((surface_color) * (1 - (layer_opac)) + (layer_color) *
(layer_opac));
```

The RenderMan spatter shader and the theory behind it was looked into modified and parts were used to create a repeating spots of colour to the surface to add procedural variation. Also a modification offer the specular component of the shader was achieved with his effect to be used as a multiplier during the specular component calculation stage

```
/// spots
varying float spot,size,scalefac,threshold = Threshold;
float layer,temp_layer, max_size = sizes;

surface_spec = color (0.0,0.0,0.0);//specularcolor;

for (layer = 1; layer <= Layers_of_spots; layer += 1)
{
/// add variation of colours to the spots
      scalefac = 1/specksize;
      for (size=1; size<=max_size; size +=1)
      {
            spot = noise(transform("shader",P)*scalefac);
            if (spot > threshold)
            {
                  surface_color += ((color
```

```
(max(0.7,1.0),max(0.7,1.0),max(0.7,1.0))) * (spattercolor * (layer *0.5)) *
1.0);
                  if ( layer >= 2 )
                  {
                    surface_spec += ((color
(max(0.1,0.3),max(0.1,0.3),max(0.1,0.3)))) * float random();
                  }
                  break;
                  //}
            }
            scalefac /= 2;
      }
max_size /= 2;
threshold *= 0.8;
scalefac /= 2;
}
surface_color *= spattercolor;
```

Textures are also used as a multiplier with the procedural patter generated to add additional variation and realism to the shader this combination of procedural components and texture components worked really well in the end to add a greater detail of realism and randomness.

Finally the summed out colours and opacity as computed in the shader

Oren-nayer was used for diffuse calculations to add a more organic clay like feel and environment reflection mapping was used as a multiplier to the reflections.

```
surface_color *= texture_front_colour;
Oi=Os;

Ci = ((Kd * (LocIllumOrenNayar(Nf,V,roughness)*surface_color)) + (specularcolor
*  specular(Nf,V,spec_roughness) * envcolor * Ks)* Oi);
```

Procedural displacement

A simulation of turbulence or fractal noise was used to create the bumpy and protruding surfaces of rocks and a great amount of detail using displacement can be achieve on relatively low resolution geometry.

This be achieved by using the noise() within a loop. On each iteration of the loop the value returned from noise() is added to the result of the previous iteration. Successfully higher frequencies but smaller amplitudes are used for iteration. The visual result is richer because the shading can

appear to mimic natural surfaces ie. large bumps have small bumps which in turn have even smaller[16]

```
/// initialise variables
float   proc_displace = 0;
normal  n = normalize(N);
point   p = transform(space, P);
float   j, f = Freq, amplitude = Amplitude;

/// loop that calculates the noise displacement in layers
for(j = 0; j < Layers; j += 1)
{
    proc_displace += (noise(p * f) -0.5) * amplitude;
    f *= 2;
    amplitude *= 0.5;
}
```
[16]

Parameters of the shaders which can be modified by the user

Can be randomised on a per object basis by imputing random values with some sort of random changing value based on each iteration of what number object is being loaded.

Surface

```
surface rock_surface_v2_r4(
float Ka= 0.5;
float Kd= 1.0;
float roughness= 10.0;
float spec_roughness = 2.0;
float Ks= 0.5;
float veining = 40.0;
color diffusecolor = ( 0.7,0.7,0.6);
color color_multi_one = ( 0.922,0.922,0.922);
color color_multi_two = ( 1.0,1.0,1.0);
color specularcolor = ( 0.1,0.1,0.1);
float Threshold = 0.7;
float Layers_of_spots = 1;
float specksize = 0.01;
float sizes = 5;
color spattercolor = color (1,1,1);
float Kenv = 0.3;
string  envname = "";
string envspace = "world";
string  texture_front = "";)
```

Displacement

```
displacement ground_displacement_v2_r1(
float Km = 0.02;
float Freq = 5;
float Amplitude = 30;
```

```
float Layers = 8;
string   space = "object";)
```

Multi purpose shader

A multi purpose shader was developed to deal with the none organic objects in the scene such as the house and fence in the back ground of the image, although a lot of the detail is not seen in these objects as they are often heavily out of focus in the piece.

It relies on texture for pattern generation and uses a environment map as a multiplier to the specular calculations. Simple diffuse calculations as used to speed up the shading process as the object this shader is used for are in the distance of the scene.

```
normal Nf;
vector V;

Nf = faceforward( normalize(N), I );
V = -normalize(I);

color texture_one_colour, texture_opacity_colour, texture_two_colour,
out_environment = color(1.0,1.0,1.0);

// texture data
if(texture_one != "")
{
    texture_one_colour = texture(texture_one);
}

if(texture_two != "")
{
    texture_two_colour = texture(texture_one);
}

color out_diffuse = ((texture_one_colour * texture_two_colour) * diffusecolor);

/// Opacity

if(texture_opacity != "")
{
    texture_opacity_colour = texture(texture_opacity);
}

/// ENVIRONMENT LIGHTING

if(EnvironmentMap != "")
{
    out_environment = envmapping(N, I, EnvironmentMap, envspace, Kenv);
}

Oi = 1;//texture_opacity_colour * opacitycolor;
Ci = ( ((Ka*ambient() + Kd*diffuse(Nf) * out_diffuse)) +
```

```
            ((specularcolor * out_environment) * Ks *
specular(Nf,V,roughness)) ) * Oi;
```

Parameters of the shaders which can be modified by the user

Can be randomised on a per object basis by imputing random values with some sort of random changing value based on each iteration of what number object is being loaded. For example this was sued to give variation to the individual fence planks.

Surface

```
surface multi_surface_v1_r1( float Ks=.5, Kd=.5, Ka=1, roughness=.1;
                        color diffusecolor=1;
                        color specularcolor=1;
                        color opacitycolor=1;
                        string  texture_one = "";
                        string  texture_two = "";
                        string  texture_opacity = "";
                        float Kenv = 0.3;
                        string EnvironmentMap = "";
                        string envspace = "world";)
```

Magnifying glass shader

The shader used for the magnifying glass was the RenderMan “shinymetal” shader and for the lens to get refractions the RenderMan “glassrefr” shader was used. For this shader to work ray tracing had to be used as this shader relies on the functionality of ray-tracing being turned on for trace samples. As ray-tracing is expensive the magnifying glass was rendered separate.

Sky

The shader was rendered with a shader that ignores lighting calculations and applies a texture out as the colour(Ci) and gives the opacity(Oi) a value of 1.0.

Statue shader

The statue shader was very important as the statue it the most important object in the piece and

requires a lot of light interaction. For the look of the statue we wanted it to look wax like and have quality's of wax such as translucency but in a more stylised way as to better fit the piece and environment.

Subsurface scattering was essential was essential for the shader and various methods were looked into to achieve this method, and various techniques were looked into to find the optimum method. The shader also needed to work well on dynamically deforming geometry.

The method that gave the best results computational and look wise was a two step process that utilised RenderMan's ability to generate point cloud data which can be read from shader to preform various task such as ambient occlusion, colour bleeding, sub-surface scattering(SSS).

Normally this to be generated through RenderMan ans this is the fastest method to generate point clouds. Steps involved generating a point cloud turning this into a brickmap which can be read through shaders.

Unfortunately as we were using the python Api as a interface to RenderMan(it is a relatively new addition to RenderMan only coming out in 2008), the point cloud api for is not yet fully supported. This presented a problem which required a work around.

The solution was to use a shader that generates a point cloud on a per object bases this case being the statue. This needs to be done once per frame if the camera or object moves. The shader used to do this follows. Data management was also a problem as point cloud data files can be relatively large and care not to run out of disk space was also a careful consideration in the project particularly when rendering the project.

```
surface bake_radiance_t(string filename = "", displaychannels = ""; color Kdt =
1)
{
        color irrad, rad_t;
        normal Nn = normalize(N);
        float a = area(P, "dicing"); // micropolygon area

        /* Compute direct illumination (ambient and diffuse) */
        irrad = ambient() + diffuse(Nn);

        /* Compute the radiance diffusely transmitted through the surface.
           Kdt is the surface color (could use a texture for e.g. freckles) */
        rad_t = Kdt * irrad;

        /* Store in point cloud file */
        bake3d(filename, displaychannels, P, Nn, "interpolate", 1,
               "_area", a, "_radiance_t", rad_t);

        Ci = rad_t * Cs * Os;
        Oi = Os;
}
```

[from PrMan doc,Illumination, Translucency and Subsurface Scattering]

This generates a point cloud in .ptc file a function was created in python to read in geometry that needed have a point cloud generated, and also read in the file name and frame number. This also would load in the camera and lights of the scene that effect are wanted to affect the point cloud, this is quite important as some lights need to be omited from affecting the point cloud. Also if shadow maps are used in the lights they need to be calculated before the point cloud generation step as shadows from the lights are needed to block light to retain the believability of the lighting in the scene.

```
def create_point_cloud(Name,SceneFunc,frame):
      print "Rendering point cloud %s.rib" %(Name)# + frame_number
      ri.Begin("__render")

      ri.Display("temp_point_cloud_file.%04d." %(frame) + Name + ".tif", "file",
"rgba")
      #ri.Display(Name +  ".rib", "file", "rgba")
      ri.Clipping(1,100)
      # Specify PAL resolution 1:1 pixel Aspect ratio
      ri.Format(res_x,res_y,1)

      ri.PixelSamples(4,4)
      ri.ShadingInterpolation("smooth")


      ri.Attribute("visibility", {"int transmission": 1})
      ri.Attribute("trace",  {"int displacements" : [1] , "bias" : [.01]})
      ri.Attribute( "cull", {"hidden" : [0]})   # don't cull hidden surfaces
        ri.Attribute( "cull", {"backfacing" : [0]})   # don't cull backfacing
surfaces
        ri.Attribute( "dice", {"rasterorient" : [0]}) # turn viewdependent
gridding off

      ri.DisplayChannel("float _area")
      ri.DisplayChannel("color _radiance_t")

      #Will be loaded as funciton defintion
      ri.Projection(ri.PERSPECTIVE,{ri.FOV:37.7777})

      #loads cameras into scene
      cameras(ri,frame)
      ri.WorldBegin()

      ri.ShadingRate([1])
      scene_lights(ri,3)
      ri.Illuminate("environment_light",0)
      ri.Illuminate("occlusion_light",0)
      ri.Illuminate("Ambient",0)
      ri.Illuminate(Fg_distance_light,0)
      ri.Illuminate(Bg_distance_light,0)
      ri.Illuminate("Light_front_fill",1)
      ri.Illuminate("Light_backlight",1)
      ri.Illuminate(fg_SpotName_01,1)
      ri.Illuminate(fg_SpotName_02,1)
      ri.Illuminate(mag_SpotName,1)
      ri.Illuminate(shad_inner_mag_SpotName,1)
```

```
      ri.Illuminate(shad_outter_mag_SpotName,1)

      ri.AttributeBegin()
      ri.Surface("./shaders/sss/bake_radiance_t",{ "string filename":
[out_filename + Name + ".%04d.ptc" %(frame)], "string displaychannels":
["_area,_radiance_t"]})
      print Name
      SceneFunc(ri,frame,point_cloud_check,out_filename,rendertype,shot)
      ri.AttributeEnd()

      ri.WorldEnd()
      ri.End()
      print " Done make brick %s.ptc" %(Name)
```

The next step involves reading the point cloud file from a shader. The shader follows that reads in the point cloud. This is also a really good feature to this shader as it avoid the conversion of the point cloud into a brickmap which is a slow process it is very optimised. It also has a smoothing option that allows you to smooth the data of the point cloud if it is not dense enough to still achieve good effect. The shader has a lot of parameter that allow control of colour and amount light can travel through the shader.

```
surface render_ssdiffusion(uniform string filename = "";
                        color albedo = color(0.830, 0.791, 0.753);   // marble
                      color dmfp = color(8.51, 5.57, 3.95);   // marble
                        float ior = 1.5;   // marble
                    float unitlength = 1.0;   // modeling scale
                        float smooth = 0;   // NEW for PRMan 14.0 !! (see sec. 4)
                    float maxsolidangle = 1.0;   // quality knob: lower is better
                        float Ka = 1, Kd = 0, Ks = 1, roughness = 0.1)
{
        normal Nn = normalize(N);
        vector V = -normalize(I);
        color amb, diff, spec, sss = 0;

        // Compute direct illumination (ambient, diffuse, and specular)
        amb = Ka * ambient();
        diff = Kd * diffuse(Nn);
        spec = Ks * specular(Nn, V, roughness);

        // Compute subsurface scattering color
        sss = subsurface(P, N, "filename", filename,
                         "albedo", albedo, "diffusemeanfreepath", dmfp,
                         "ior", ior, "unitlength", unitlength,
                         "smooth", smooth,
                    "maxsolidangle", maxsolidangle);

        // Set Ci and Oi
        Ci = (amb + diff + sss) * Cs + spec;
        Ci *= Os;
        Oi = Os;
}
[from PrMan doc,Illumination, Translucency and Subsurface Scattering ]
```

This is read in and applied to object which is rib archived geometry file in the following way this also show the final parameter used on the shader. The statue is also applied a version of the

ground/rocks displacement shader with a high frequency and low amplitude to give it richer look when viewed up close and to diffuse the and high lights better.

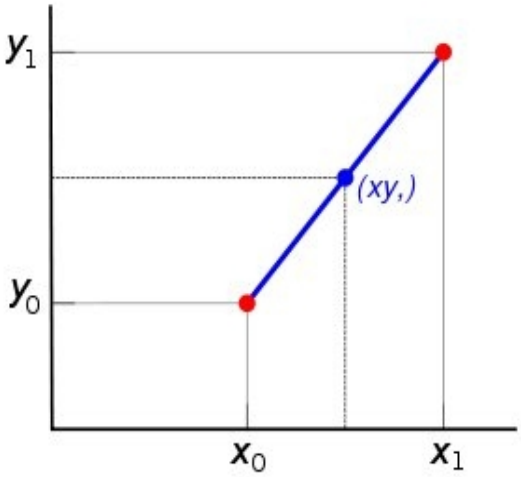
```
ri.Surface("./shaders/sss/render_ssdiffusion",{   "uniform   string   filename":
[out_filename + Name + ".%04d.ptc" %(frame)],
 "color albedo": [0.746, 0.741, 0.428],
"color dmfp":  [6.96,  6.40,  1.50],
"float ior": [1.5],
"float unitlength": [1.0],
"float smooth": [1],
"float maxsolidangle": [1],
"float Ka": [0.4],
"float Kd" : [0.35],
"float Ks": [0.4],
"float roughness": [0.2]})

ri.Displacement("./shaders/ground/ground_displacement_v2_r1",{    "float    Km":
[0.004], "float Freq":[100], "float Amplitude":[2.5], "float Layers":[8],"string
space":["shader"]})
ri.Attribute("displacementbound",{ri.COORDINATESYSTEM:["object"],"uniform  float
sphere":[0.2]})
if(statue_type == 1):
                ri.ReadArchive("./Archive/statue/statue_shot_three_four/first_an
d_second_hole_ribs.0001.rib")
```

Another note that is important is that the effect of the subsurface scattering changes based on the distance of the object form the camera and size on screen even if the lights don't move(scene dependent). This meant that the lights that effected the statue had to be modified for each shot to match which was a challenging lighting task.

```
### spot light fg one
fg_coneAngle_01_intensity_default_shot_01 = 50000
fg_coneAngle_01_intensity_statue_shot_01 = 12000
if (shot == 1) :
      fg_coneAngle_01_intensity_statue_scatter_shot_01 = 20000
if (shot == 2) :
      fg_coneAngle_01_intensity_statue_scatter_shot_01 = 65000
if (shot == 3) :
      fg_coneAngle_01_intensity_statue_scatter_shot_01 = 33000
if (shot == 4) :
      fg_coneAngle_01_intensity_statue_scatter_shot_01 = 50000
if (shot == 5) :
      fg_coneAngle_01_intensity_statue_scatter_shot_01 = 25000
if (shot == 6) :
      fg_coneAngle_01_intensity_statue_scatter_shot_01 = 23000
```

Cameras

As the camera had to be used by various function as well as in the main scene loop it made sense to make them in there own function. And as there need to be movement to some of the shots with precise control interpolation, smooth interpolations had to wrote in python to control the movement these functions take in a start value, and an end value and interpolate them based on the start and end frame of when we wanted to specify the movement. If the start frame is not reached the start value is reached and value remains static and will only start to change once the start frame is reached. If it is after the end frame the the end value is returned.

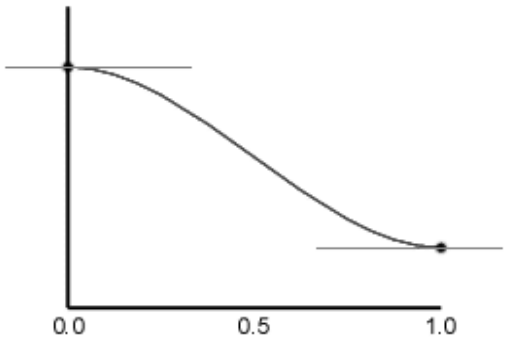


$$y = y_a + (x - x_a)\frac{(y_b - y_a)}{(x_b - x_a)}$$

Fig. Linear Interpolation (http://en.wikipedia.org/wiki/Interpolation#Linear_interpolation)

```
def lerp(start_value,end_value,start_frame,end_frame,frame):
      if(frame < start_frame):
            return start_value
      if(frame > end_frame):
            return end_value
      else :
            frame_range = end_frame - start_frame
            t = (frame - start_frame*1.0) / frame_range
            return ((1-t)*start_value) + (t*end_value)
```

A smooth interpolation was also needed to blend in the motion of the cameras beging and end movement. Cubic interpolation was chosen for this as it stops the sudden sudden start stop motion in the gradient.



Cubic interpolation function

$k(u) = k(0)\times(2u^3-3u^2+1) + k(1)\times(3u^2-2u^3)$

(http://www.gamedev.net/reference/articles/article1497.asp)

```
def smoothstep(start_value,end_value,start_frame,end_frame,frame):
      if(frame < start_frame):
            return start_value
      if(frame > end_frame):
            return end_value
      else :
            frame_range = end_frame - start_frame
            t = (frame - start_frame*1.0) / frame_range
            print "%f" %(t)
            x = start_value * ( 2*(t*t*t) - 3*(t*t) + 1 )  +    (  end_value * (
(3*(t*t)) - (2*(t*t*t))  )   )
            return x
```

A noise function was also generated to have subtle motion on the cameras if needed, this was greatly reduced in the final piece however as we prefered the more smooth controlled cameras.

```
def noise(amount,frame):
      return ((math.sin(frame + random.uniform(1,100)) * amount))
```

Example of the camera function and how different values are assigned to each shot and shows the interpolation functions in action, also how motion blur is set for shots with movement by evaluating the movement of the camera a frame ahead as well and putting them in a motion block. Depth of field is applied out of the camera function as it is only needed in the main pass and is not need in the point cloud and shadow passes. Setting up the cameras to produce good cinematography inside RenderMan was rather a challenge (without the visual luxuries you get inside of Maya of actually being able to visualise your scenes with out rendering), lots of tweaks were needed to get satisfactory results but as the project progressed moving round the scene with out any visuals and

just values seemed almost intuitive.

```
def cameras(ri,frame):
      #fixes rotation of scene
      noise_tran_x = noise(0.002,frame)
      noise_tran_y = noise(0.002,frame)
      noise_tran_z = noise(0.002,frame)

      noise_rot_x = noise(0.002,frame)
      noise_rot_y = noise(0.002,frame)
      noise_rot_z = noise(0.002,frame)

      ri.Rotate(180,0,1,0)
      if (shot == 1) :
            cam_x = 1.0

            cam_y = smoothstep(45,1,200,500,frame)
            cam_y_nf = smoothstep(45,1,200,500,frame+1)
            #cam_y = 1
            #cam_y_nf = 1
            print "%f" %(cam_y)
            cam_z = -36.4
            ri.Shutter(0,1)
            ri.MotionBegin([-0.5,1])
            ri.Translate(cam_x,-cam_y,cam_z)
            ri.Translate(cam_x,-cam_y_nf,cam_z)
            ri.MotionEnd()
            ri.Rotate(-6,1,0,0)
            ri.Rotate(-1,0,1,0)
      if (shot == 2):
            cam_x = 3
            cam_y = 5
            cam_z = -8
            ri.Translate(cam_x,-cam_y,cam_z)
            #ri.Translate(0,-3,-45)
            ri.Rotate(-40,1,0,0)
            ri.Rotate(-85,0,1,0)
      if (shot == 3) :
            cam_x = 0
            cam_y = 4
            cam_z = -15
            ri.Translate(cam_x,-cam_y,cam_z)
            ri.Rotate(-15,1,0,0)
            ri.Rotate(-3,0,1,0)
      if (shot == 4) :
            cam_x = 1.3
            cam_y = 6
            cam_z = -5
            ri.Translate(cam_x,-cam_y,cam_z)
            ri.Rotate(-9,1,0,0)
            ri.Rotate(-1,0,1,0)
      if (shot == 5) :
            cam_x = 2.1
            cam_y = 0.0
            cam_z = -6
            ri.Translate(cam_x,-cam_y,cam_z)
            ri.Rotate(-6.5,1,0,0)
            ri.Rotate(6,0,1,0)
      if (shot == 6) :
            cam_x = -0.3
```

```
cam_y = smoothstep(-3.5,-4.5,700,950,frame)
cam_z = smoothstep(-14.5,-17.5,700,950,frame)
cam_z_nf = smoothstep(-14.5,-17.5,700,950,frame+1)
ri.MotionBegin([-0.5,1])
ri.Translate(cam_x,cam_y,cam_z)
ri.Translate(cam_x,cam_y,cam_z_nf)
ri.MotionEnd()
ri.Rotate(-25,1,0,0)
ri.Rotate(-3,0,1,0)
```

Lighting

Lighting was a challenging aspect as lights needed be placed mainly and a lot of work needed to be done manually that is taken for granted when using an animation system as an interface to RenderMan(for instance setting up shadows). But using the Python api did allow more fidelity in control once everything had begun to be set up.

Using ray trace lighting was avoid as it is slow and less optimised and would not be able to handle the complexity of the scenes. Conventional lights with some having shadows enabled through depth maps. Shadow maps for the lights need to compute each from for the lights which effect moving lights. However we did reuse shadow maps on shots were possible as optimisation.

A function was made to compute shadow maps as this allow extensibility if more show emiting lights were needed.

```
def ShadowPass(Name,From,To,coneAngle,SceneFunc,frame,res) :
      frame_number = str(frame)
      print "Rendering Shadow pass %s.z" %(Name) + frame_number
      ri.Begin("__render")
      ri.Display(Name + ".%04d" %(frame) + ".z", "zfile", "z")
      ri.Clipping(0.01,6000)
      # Specify PAL resolution 1:1 pixel Aspect ratio
      ri.Format(res,res,1)
      # now set the projection to perspective
      ri.Projection(ri.PERSPECTIVE,{ri.FOV:coneAngle*(360/math.pi)})
      #now move to light position
      ri.Hider( "hidden", {"jitter" : [0]})
      ri.ShadingRate(1)
      ri.PixelFilter( ri.BOX ,1.0, 1.0)
      ri.PixelSamples(1,1)

      direction = map(lambda x,y : x-y , To,From)
      AimZ(direction)
      ri.Translate(-From[0],-From[1],-From[2])
      # now draw the Scene
      ri.WorldBegin()

      SceneFunc(ri,frame,point_cloud_check,out_filename,rendertype,shot)
      ri.WorldEnd()

      ri.MakeShadow(Name + ".%04d" %(frame)+ ".z", Name + ".%04d" %(frame) +
```

```
".shad")
	ri.End()
	print " Done MakeShadow %s.shad" %(Name)
```

[http://nccastaff.bournemouth.ac.uk/jmacey/Renderman/index.html]

The lighting set up for the seen required some of the lights values/names to be declared global as to allow other functions to have access to certain information easier. There is also a scene light function which generates the all the lights. This function makes use of the previous interpolation and noise functions generated to animate the camera on certain lights.

There is also a control value to decide what value the lights have, as certain objects had to have different amounts of values from the light particularly the statue as it has a subsurface scattering lighting information added in the the point cloud calculation as-well as the usual diffuse specular calculations. So these each have different values depending on what the light was currently shading. For instance a default light value for the rest of the scene (a value of 1 is passed in), the statue diffuse specular light(a value of 2 is passed in), and sss calculation lights(a value of 3 is passed in).

The whole function is in the appendix but some segment showing one light follows to demonstrate

```
def scene_lights(ri,value):

Light_control = value
...
some missing
..

### Spot light
	if ( Light_control == 1 ):
		fg_coneAngle_02_intensity = fg_coneAngle_02_intensity_default_shot_01
	if ( Light_control == 2 ):
		fg_coneAngle_02_intensity = fg_coneAngle_02_intensity_statue_shot_01
	if ( Light_control == 3 ):
		fg_coneAngle_02_intensity = fg_coneAngle_02_intensity_statue_scatter_shot_01


ri.LightSource( "shadowspot", {ri.HANDLEID:fg_SpotName_01,"point from" : fg_SpotFrom_01, "point
to" : fg_SpotTo_01,"float intensity" : fg_coneAngle_01_intensity,"string shadowname" :out_filename +
fg_SpotName_01 + ".%04d" %(frame) + ".shad","float coneangle" : fg_coneAngle_01,"float
conedeltaangle" : [0.05], "color lightcolor": fg_coneAngle_01_colour, "float  beamdistribution":
[2],"float samples":[128],"float width":[32]})
```

The lighting is set up with two direction lights with low intensity on for a front fill with a blue tint and a back lighting fill with a red tint this gives a good base lighting of the scene. There are two spot lights that effect the fore ground with strong intensity and falloffs that produce also use show maps to cast shadows. These lights essentially represent the sun light. A lot of work was done in developing the lighting look off the scene and this set up seemed to work the best.

For the magnifying glass light there was a lot of trickery involved to fake the look. The set up has is made up of three spot lights which have the same point too and from position so they work

together. The first light has a narrow cone angle and a high intensity to produce the bright focal point area. The second has a wide cone angle to produce additional light but with a lower intensity and the third light has an even wide cone angle and has a negative intensity value to cheat the effect of shadows. These lights also work with the subsurface calculations.

```
#### magnifying glass light
mag_SpotName="mag_Spot"
mag_SpotFrom=[-20,20,25]
#mag_coneAngle=0.0075
mag_coneDeltaAngle=0.05
mag_colour=[1.0,0.9,0.8]
mag_intensity_default=600

#### magnifying glass shadow lights
shad_inner_mag_SpotName="shad_inner_mag_Spot"
shad_inner_mag_colour=[0.75,0.7,0.7]
shad_inner_mag_coneAngle=0.14
shad_inner_mag_coneDeltaAngle=0.21
shad_inner_mag_intensity_default=1850

#### magnifying glass shadow lights
shad_outter_mag_SpotName="shad_outter_mag_Spot"
shad_outter_mag_colour=[1.0,1.0,1.0]
shad_outter_mag_coneAngle=shad_inner_mag_coneAngle+0.02
shad_outter_mag_coneDeltaAngle=0.21
shad_outter_mag_intensity_default=-1100
```

Rendering

Was controlled with user input to control the file name, frame range, resolution, shading rate, pixel samples, shot number, computer shadows/ point clouds, and render layer eg foreground mid-ground.

For example to render a shot for high resolution for a the first hundred frames you would run the script with these flags proceeding it

```
./scene_v9_4_begining_of_shot_2_etc.py final_v1_r1_shot_01_fg_beauty_  1 100
1024 576 0.3 16 1 1 2 1
```

Images were rendered out in 32 bit floating point with the exr file format, Mitchell filtering was used as this gives the sharpest aliased free result. Depth of field was set with on a per frame basis by setting the focal distance and f stop. Objects in the scene were split into layers of the statue, foreground, mid-ground back ground sky, occlusion, environment and masking elements to add in compositing this can be controlled with the render layer flag mostly. Shadows and point clouds are

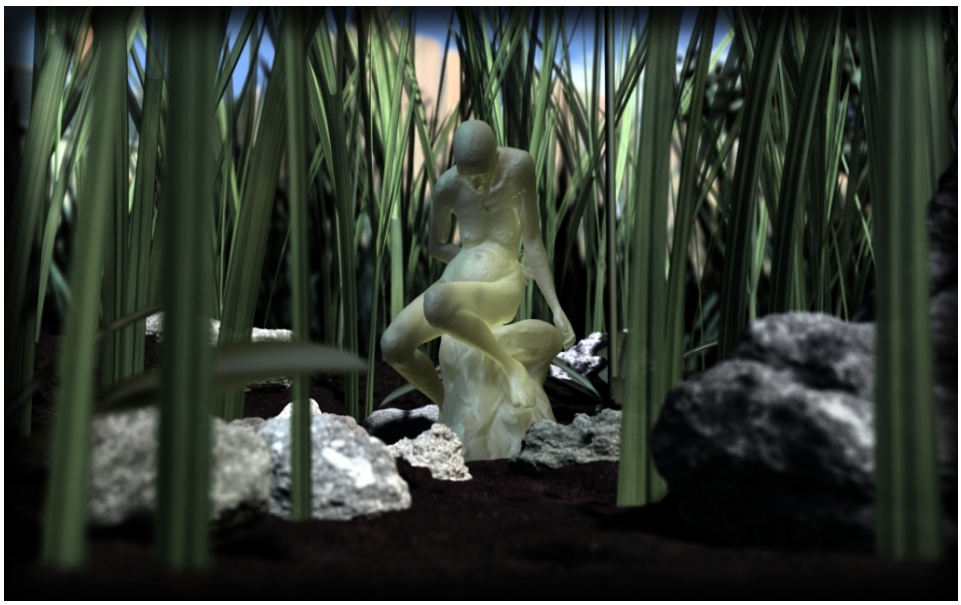
produced before the main rendering begins and can rendered on there own first with out the scene being rendered. It is easy to control the quality of the images as you have access to the resolution shading rate and pixel samples from the flags in the script so you can do quick test previews or high res tests. Or set a batch of renders off by by loging in to multiple machines using ssh or manually loging in and runing the scripts over the required frame range and shot. Render directly from the python script instead of making the script convert to a RIB files as this would be a relative hit on disk space resources and little gain is given as the RIB wouldn't really needed to be modified at this point. Also with the render controls of the scripts it is easy to distribute renders.

Compositing

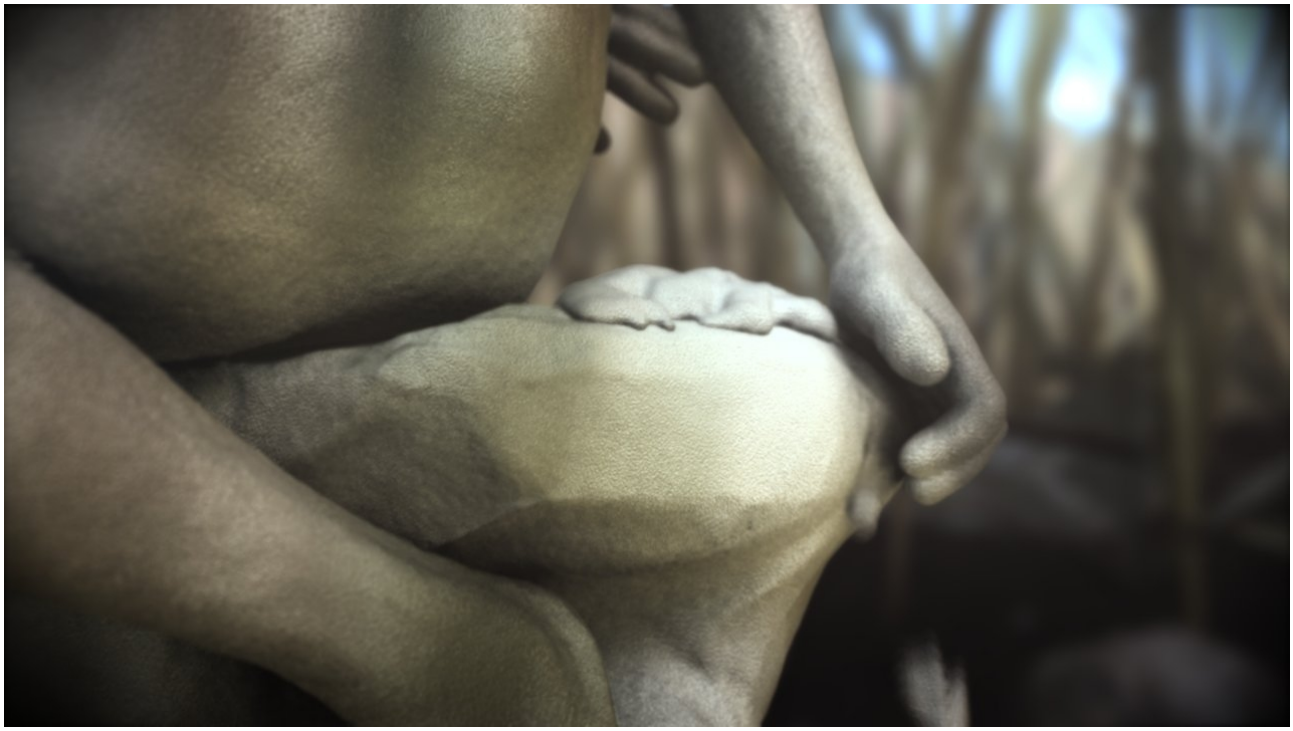
Compositing was mainly done in Shake with Nuke for support for certain features. Some colour correction and post adjustments/enhancements was done and of course combining layers and passes of the renders and smoke which was rendered separately in Maya. Shake was also used for the editing as it allowed us to keep the images in 32 bit colour space and limited degrading the quality of the image by putting it through other software as the we would have had to convert to 8 bit prior to the final export.

Look Development process showing end result for an example shot from animation

Various tests an experiments on the look development of the piece**.**



Final Stills from Animation



Conclusion

Using the aforementioned tools and techniques we managed to get very comfortable using the pipeline between RenderMan, Houdini and Maya which we tested in production. We also broaden our knowledge and skills and in the end we produced a visually interesting project from an artistic view and a technical standpoint having created, what we believe, a unique result.

Bibliography

[1] Rodins Works: The Gates of Hell, Wikipedia The Free Encyclopaedia http://en.wikipedia.org/wiki/The_Gates_of_Hell

[2] Rodin Works:She Who was the Helmet-Maker's once beautiful Wife http://www.rodin-web.org/works/1884_helmetmakers_wife.htm

[3] Houdini Documentation: Particle Fluid Surface

[4] Houdini Documentation: Interact Node

[5] Houdini Documentation: Property Node

[6] Przemyslaw Prusinkiewicz, Aristid Lindenmayer, The Algorithmic Beauty of Plants , Springer; 1 edition (October 11, 1991)

[7] Graig Zerouni, Houdini On The Spot: Power User Tips and Techniques, Focal Press, ElseVier, First Edition, 2007

[8] Bill Fleming, Digital Botany and Creepy Insects, Mastering 3D Graphics, Wiley Computer Publishing John Wiley & Sons, Inc, First Edition 2000

[9] Simon Clavet, Philippe Beaudoin, and Pierre Poulin, Particle-based Viscoelastic Fluid Simulation, LIGUM, Dept. IRO, Université de Montréal, Eurographics/ACM SIGGRAPH Symposium on Computer Animation (2005) K. Anjyo, P. Faloutsos (Editors) #

[10] PrMan docs, JonMacey's online documentation http://nccastaff.bournemouth.ac.uk/jmacey/Renderman/index.html

[11] Rudy Cortes, Saty Raghavachary, The RenderMan Shading Language Guide, Thomson Course Technology, 2008

[12] Steve Upstill, The RenderMan Companion : A programmer's guide to realistic computer

Graphics, Addison-Wesley Professional 1990.

[13] David S. Ebert, E. kenton Musgrave, Darwyn Peachey, Ken Perlin, Steve Worley, Texturing and Modeling : a procedural approach, Morgan Kaufman Publishers, Third Edition, 2002

[14] Antony A. Apodaca, Advanced RenderMan: Creating CGI for Motion Pictures, Morgan Kaufman Publishers, First Edition, 1999

[15] http://www.fundza.com/rman_shaders/overview/overview.html

[16]http://www.fundza.com/rman_shaders/displacement/index.html

Notes: The model of the house in the background was a model that was created and used by Joe Gaffney for a previous project. The model of the statue in created by Anand Gopinath and purchased from www.gnomology.com. Some textures was used from www.cgtextures.com

Appendix

Python scripts

scene_v13_2_final.py

```
#!/usr/bin/python
#Joe Gaffney MSc 2009
# for bash we need to add the following to our .bashrc
# export PYTHONPATH=$PYTHONPATH:$RMANTREE/bin
##############################################################################################################
##########
# Main python script which handles scene management and lighting and rendering of the project etc
"""############################################################################################################
##########
#
Parameters
#scriptname (filemname) test_render_v1_r1 (start_frame) 1 (end_frame) 1 (resolution x) 720
(resolution y) 404
(shading rate) 5 (pixel samples) 2 (shot) 1 (compute shadows point clouds) 1 (rendertype) 1
(renderlayer) 1
# TO RUN
# LOW QUALITY
# ./scene_v13_2_final.py final_v1_r1_shot_01_fg_beauty_  1 100 720 404 10 4 1 1 2 1
# HIGH QUALITY HD
# ./scene_v13_2_final.py final_v1_r1_shot_01_fg_beauty_  1 100 1024 576 0.3 16 1 1 2 1
##############################################################################################################
#######"""

"""############################################################################################################
##########
# IMPORTED MODULES
##############################################################################################################
########"""
import getpass,time,random,math,os,sys,fileinput,prman
from archive_v6_r1_final import *

"""############################################################################################################
#########
# READ IN VALUES
##############################################################################################################
#######"""
out_filename = sys.argv [1]
start_frame = int(sys.argv [2])
end_frame = int(sys.argv [3])
res_x = int(sys.argv [4])
res_y = int(sys.argv [5])
shading_rate = float(sys.argv [6])
pixel_samples = int(sys.argv [7])
current_shot = int(sys.argv [8])
compute_shadows = int(sys.argv [9])
in_rendertype = int(sys.argv [10])
renderlayer = int(sys.argv [11])

rendertype = 1
"""############################################################################################################
#########
# Create an instance of the RenderMan interface
##############################################################################################################
#######"""
ri = prman.Ri() #
ri.Option("rib", {"string asciistyle": "indented"})

"""############################################################################################################
#########
# Asign global varibles
##############################################################################################################
#######"""
cam_move_y = 0
shot = current_shot
point_cloud_check = 0
"""############################################################################################################
```

```
#########
# Lights
#############################################################################################################
#######"""

"""#############################################################################################################
#########
# Foreground lights
#############################################################################################################
#######"""
### Back light
Back_light_default_shot_01 = 0.4
Back_light_intensity_statue_shot_01 = 0.2
if (shot == 1) :
        Back_light_intensity_statue_scatter_shot_01 = 0.3
if (shot == 2) :
        Back_light_intensity_statue_scatter_shot_01 = 0.45
if (shot == 3) :
        Back_light_intensity_statue_scatter_shot_01 = 0.45
if (shot == 4) :
        Back_light_intensity_statue_scatter_shot_01 = 0.15
if (shot == 5) :
        Back_light_intensity_statue_scatter_shot_01 = 0.45
if (shot == 6) :
        Back_light_intensity_statue_scatter_shot_01 = 0.45

### Front light
Front_light_default_shot_01 = 0.4
Front_light_intensity_statue_shot_01 = 0.4
if (shot == 1) :
        Front_light_intensity_statue_scatter_shot_01 = 0.3
if (shot == 2) :
        Front_light_intensity_statue_scatter_shot_01 = 0.45
if (shot == 3) :
        Front_light_intensity_statue_scatter_shot_01 = 0.45
if (shot == 4) :
        Front_light_intensity_statue_scatter_shot_01 = 0.15
if (shot == 5) :
        Front_light_intensity_statue_scatter_shot_01 = 0.45
if (shot == 6) :
        Front_light_intensity_statue_scatter_shot_01 = 0.45

### spot light fg one
fg_coneAngle_01_intensity_default_shot_01 = 50000
fg_coneAngle_01_intensity_statue_shot_01 = 12000
if (shot == 1) :
        fg_coneAngle_01_intensity_statue_scatter_shot_01 = 20000
if (shot == 2) :
        fg_coneAngle_01_intensity_statue_scatter_shot_01 = 65000
if (shot == 3) :
        fg_coneAngle_01_intensity_statue_scatter_shot_01 = 33000
if (shot == 4) :
        fg_coneAngle_01_intensity_statue_scatter_shot_01 = 50000
if (shot == 5) :
        fg_coneAngle_01_intensity_statue_scatter_shot_01 = 25000
if (shot == 6) :
        fg_coneAngle_01_intensity_statue_scatter_shot_01 = 23000

### spot light fg two
fg_coneAngle_02_intensity_default_shot_01 = 44000
fg_coneAngle_02_intensity_statue_shot_01 = 10000
if (shot == 1) :
        fg_coneAngle_02_intensity_statue_scatter_shot_01 = 16000
if (shot == 2) :
        fg_coneAngle_02_intensity_statue_scatter_shot_01 = 60000
if (shot == 3) :
        fg_coneAngle_02_intensity_statue_scatter_shot_01 = 30000
if (shot == 4) :
        fg_coneAngle_02_intensity_statue_scatter_shot_01 = 46000
if (shot == 5) :
        fg_coneAngle_02_intensity_statue_scatter_shot_01 = 25000
if (shot == 6) :
        fg_coneAngle_02_intensity_statue_scatter_shot_01 = 25000

#### Forground_lights
fg_SpotName_01="fg_Spot1"
fg_SpotFrom_01=[-70,110,110]
```

```
fg_SpotTo_01=[0,-5,0]
fg_coneAngle_01=500
fg_coneAngle_01_colour=[0.9,0.8,0.85]

fg_SpotName_02="fg_Spot2"
fg_SpotFrom_02=[-60,100,80]
fg_SpotTo_02=[0,-5,-10]
fg_coneAngle_02=500
fg_coneAngle_02_colour=[0.8,0.9,0.85]

#### magnifying glass light
mag_SpotName="mag_Spot"
mag_SpotFrom=[-20,20,25]
#mag_coneAngle=0.0075
mag_coneDeltaAngle=0.05
mag_colour=[1.0,0.9,0.8]
mag_intensity_default=600

#### magnifying glass shadow lights
shad_inner_mag_SpotName="shad_inner_mag_Spot"
shad_inner_mag_colour=[0.75,0.7,0.7]
shad_inner_mag_coneAngle=0.14
shad_inner_mag_coneDeltaAngle=0.21
shad_inner_mag_intensity_default=1850

#### magnifying glass shadow lights
shad_outter_mag_SpotName="shad_outter_mag_Spot"
shad_outter_mag_colour=[1.0,1.0,1.0]
shad_outter_mag_coneAngle=shad_inner_mag_coneAngle+0.02
shad_outter_mag_coneDeltaAngle=0.21
shad_outter_mag_intensity_default=-1100

"""#############################################################################################################
#########
# Lights background
#############################################################################################################
#######"""
### Foreground distance shadow light
Fg_distance_light = "fg_distance_light"
Fg_distance_light_From=[-80,200,150]
Fg_distance_light_To=[0,0,0]

Bg_distance_light = "bg_distance_light"
Bg_distance_light_From=[-80,200,150]
Bg_distance_light_To=[0,0,0]

### foreground light
Fg_distance_intensity_default_shot_01 = 2.1
Fg_distance_intensity_statue_shot_01 = 0.15
Fg_distance_intensity_statue_scatter_shot_01 = 0.5

### background light
Bg_distance_intensity_default_shot_01 = 1.7
Bg_distance_intensity_statue_shot_01 = 0.15
Bg_distance_intensity_statue_scatter_shot_01 = 0.5


"""#############################################################################################################
#########
# Jon Maceys funciton code to reorientate the camera for shadow map calculaitons
http://nccastaff.bournemouth.ac.uk/jmacey/Renderman/index.html
#############################################################################################################
#######"""
def AimZ(direction) :
        if (direction[0]==0 and direction[1]==0 and direction[2]==0) :
                return
        #
        # The initial rotation about the y axis is given by the projection of
        # the direction vector onto the x,z plane: the x and z components
        # of the direction.

        xzlen = math.sqrt(direction[0]*direction[0]+direction[2]*direction[2])
        if (xzlen == 0) :
                if(direction[1] <0) :
                        yrot = 0
                else :
                        yrot =180
```

```
        #              yrot = (direction[1] < 0) ? 180 : 0
        else :
               yrot = 180*math.acos(direction[2]/xzlen)/math.pi;


         # The second rotation, about the x axis, is given by the projection on
         # the y,z plane of the y-rotated direction vector: the original y
         # component, and the rotated x,z vector from above.

        yzlen = math.sqrt(direction[1]*direction[1]+xzlen*xzlen)
        xrot = 180*math.acos(xzlen/yzlen)/math.pi     # yzlen should never be 0

        if (direction[1] > 0) :
               ri.Rotate(xrot, 1.0, 0.0, 0.0)
        else :
               ri.Rotate(-xrot, 1.0, 0.0, 0.0)
        #The last rotation declared gets performed first
        if (direction[0] > 0) :
               ri.Rotate(-yrot, 0.0, 1.0, 0.0)
        else :
               ri.Rotate(yrot, 0.0, 1.0, 0.0)

"""#############################################################################################################
#########
Modified from jon Maceys notes on Renderman and python
http://nccastaff.bournemouth.ac.uk/jmacey/Renderman/index.html
#############################################################################################################
#######"""
def DistanceShadowPass(Name,From,To,SceneFunc,frame) :
        frame_number = str(frame)
        print "Rendering distance Shadow pass %s.z" %(Name) + frame_number
        ri.Begin("__render")
        ri.Display(Name + ".%04d" %(frame) + ".z", "zfile", "z")
        ri.Clipping(1,6000)
        # Specify PAL resolution 1:1 pixel Aspect ratio
        ri.Format(1024,1024,1)
        # now set the projection to perspective
        ri.Projection(ri.PERSPECTIVE,{ri.FOV:37.7777})
        #ri.Projection(ri.ORTHOGRAPHIC)
        #now move to light position
        # create a vector for the Spotlight to and from values
        ri.Hider( "hidden", {"jitter" : [0]})
        ri.ShadingRate(1)
        ri.PixelFilter( ri.BOX ,1.0, 1.0)
        ri.PixelSamples(1,1)
        #ri.ScreenWindow( -2, 1.3, -1.8, 2.2)

        #ri.Option( "limits", {"bucketsize": [32,32]})
        #ri.Option( "limits", {"gridsize" :[32]})
        #ri.ShadingInterpolation("smooth")

        # to do this we subtract each of thelist elements using a lambda function
        # this is the same as doing the code below I will leave it to you as to which you
        # find more readable
        #direction =[To[0]-From[0],To[1]-From[1],To[2]-From[2]]

        direction = map(lambda x,y : x-y , To,From)
        AimZ(direction)
        ri.Translate(-From[0],-From[1],-From[2])
        # now draw the Scene
        ri.WorldBegin()

        SceneFunc(ri,frame,point_cloud_check,out_filename,rendertype)
        ri.WorldEnd()

        ri.MakeShadow(Name + ".%04d" %(frame)+ ".z", Name + ".%04d" %(frame) + ".shad")
        ri.End()
        print " Done MakeShadow %s.shad" %(Name)

"""#############################################################################################################
#########
Modified from jon Maceys notes on Renderman and python
http://nccastaff.bournemouth.ac.uk/jmacey/Renderman/index.html
#############################################################################################################
#######"""
def ShadowPass(Name,From,To,coneAngle,SceneFunc,frame,res) :
```

```
        frame_number = str(frame)
        print "Rendering Shadow pass %s.z" %(Name) + frame_number
        ri.Begin("__render")
        ri.Display(Name + ".%04d" %(frame) + ".z", "zfile", "z")
        ri.Clipping(0.01,6000)
        # Specify PAL resolution 1:1 pixel Aspect ratio
        ri.Format(res,res,1)
        # now set the projection to perspective
        ri.Projection(ri.PERSPECTIVE,{ri.FOV:coneAngle*(360/math.pi)})
        #now move to light position
        ri.Hider( "hidden", {"jitter" : [0]})
        ri.ShadingRate(1)
        ri.PixelFilter( ri.BOX ,1.0, 1.0)
        ri.PixelSamples(1,1)

        direction = map(lambda x,y : x-y , To,From)
        AimZ(direction)
        ri.Translate(-From[0],-From[1],-From[2])
        # now draw the Scene
        ri.WorldBegin()

        SceneFunc(ri,frame,point_cloud_check,out_filename,rendertype,shot)
        ri.WorldEnd()

        ri.MakeShadow(Name + ".%04d" %(frame)+ ".z", Name + ".%04d" %(frame) + ".shad")
        ri.End()
        print " Done MakeShadow %s.shad" %(Name)

"""##############################################################################################################
#########
# Function that generates point clouds
##############################################################################################################
#######"""
def create_point_cloud(Name,SceneFunc,frame):
        print "Rendering point cloud %s.rib" %(Name)# + frame_number
        ri.Begin("__render")

        ri.Display("temp_point_cloud_file.%04d." %(frame) + Name + ".tif", "file", "rgba")
        #ri.Display(Name +  ".rib", "file", "rgba")
        ri.Clipping(1,100)
        # Specify PAL resolution 1:1 pixel Aspect ratio
        ri.Format(res_x,res_y,1)

        ri.PixelSamples(4,4)
        ri.ShadingInterpolation("smooth")


        ri.Attribute("visibility", {"int transmission": 1})
        ri.Attribute("trace",  {"int displacements" : [1] , "bias" : [.01]})
        ri.Attribute( "cull", {"hidden" : [0]})   # don't cull hidden surfaces
         ri.Attribute( "cull", {"backfacing" : [0]})   # don't cull backfacing surfaces
         ri.Attribute( "dice", {"rasterorient" : [0]}) # turn viewdependent gridding off

        ri.DisplayChannel("float _area")
        ri.DisplayChannel("color _radiance_t")

        #Will be loaded as funciton defintion
        ri.Projection(ri.PERSPECTIVE,{ri.FOV:37.7777})

        #loads cameras into scene
        cameras(ri,frame)
        ri.WorldBegin()

        ri.ShadingRate([1])
        scene_lights(ri,3)
        ri.Illuminate("environment_light",0)
        ri.Illuminate("occlusion_light",0)
        ri.Illuminate("Ambient",0)
        ri.Illuminate(Fg_distance_light,0)
        ri.Illuminate(Bg_distance_light,0)
        ri.Illuminate("Light_front_fill",1)
        ri.Illuminate("Light_backlight",1)
        ri.Illuminate(fg_SpotName_01,1)
        ri.Illuminate(fg_SpotName_02,1)
        ri.Illuminate(mag_SpotName,1)
        ri.Illuminate(shad_inner_mag_SpotName,1)
        ri.Illuminate(shad_outter_mag_SpotName,1)
```

```
        ri.AttributeBegin()
        ri.Surface("./shaders/sss/bake_radiance_t",{ "string filename": [out_filename + Name + ".
%04d.ptc" %(frame)], "string displaychannels": ["_area,_radiance_t"]})
        print Name
        SceneFunc(ri,frame,point_cloud_check,out_filename,rendertype,shot)
        ri.AttributeEnd()

        ri.WorldEnd()
        ri.End()
        print " Done make brick %s.shad" %(Name)

"""#########################################################################################################
#########
# Linear interpolaiton function
#########################################################################################################
#######"""
def lerp(start_value,end_value,start_frame,end_frame,frame):
        if(frame < start_frame):
                return start_value
        if(frame > end_frame):
                return end_value
        else :
                frame_range = end_frame - start_frame
                t = (frame - start_frame*1.0) / frame_range
                return ((1-t)*start_value) + (t*end_value)

"""#########################################################################################################
#########
# Cubic interpolation function
#########################################################################################################
#######"""
def smoothstep(start_value,end_value,start_frame,end_frame,frame):
        if(frame < start_frame):
                return start_value
        if(frame > end_frame):
                return end_value
        else :
                frame_range = end_frame - start_frame
                t = (frame - start_frame*1.0) / frame_range
                print "%f" %(t)
                x = start_value * ( 2*(t*t*t) - 3*(t*t) + 1 )  +    (  end_value * (   (3*(t*t)) -
(2*(t*t*t))  )   )
                return x

"""#########################################################################################################
#########
# Noise function
#########################################################################################################
#######"""
def noise(amount,frame):
        return ((math.sin(frame + random.uniform(1,100)) * amount))


"""#########################################################################################################
#########
# Camera definiton function
#########################################################################################################
#######"""
def cameras(ri,frame):
        #fixes rotation of scene
        noise_tran_x = noise(0.002,frame)
        noise_tran_y = noise(0.002,frame)
        noise_tran_z = noise(0.002,frame)

        noise_rot_x = noise(0.002,frame)
        noise_rot_y = noise(0.002,frame)
        noise_rot_z = noise(0.002,frame)

        ri.Rotate(180,0,1,0)
        if (shot == 1) :
                cam_x = 1.0

                cam_y = smoothstep(45,1,200,500,frame)
                cam_y_nf = smoothstep(45,1,200,500,frame+1)
                #cam_y = 1
                #cam_y_nf = 1
```

```
                print "%f" %(cam_y)
                cam_z = -36.4
                ri.Shutter(0,1)
                ri.MotionBegin([-0.5,1])
                ri.Translate(cam_x,-cam_y,cam_z)
                ri.Translate(cam_x,-cam_y_nf,cam_z)
                ri.MotionEnd()
                ri.Rotate(-6,1,0,0)
                ri.Rotate(-1,0,1,0)
        if (shot == 2):
                cam_x = 3
                cam_y = 5
                cam_z = -8
                ri.Translate(cam_x,-cam_y,cam_z)
                #ri.Translate(0,-3,-45)
                ri.Rotate(-40,1,0,0)
                ri.Rotate(-85,0,1,0)
        if (shot == 3) :
                cam_x = 0
                cam_y = 4
                cam_z = -15
                ri.Translate(cam_x,-cam_y,cam_z)
                ri.Rotate(-15,1,0,0)
                ri.Rotate(-3,0,1,0)
        if (shot == 4) :
                cam_x = 1.3
                cam_y = 6
                cam_z = -5
                ri.Translate(cam_x,-cam_y,cam_z)
                ri.Rotate(-9,1,0,0)
                ri.Rotate(-1,0,1,0)
        if (shot == 5) :
                cam_x = 2.1
                cam_y = 0.0
                cam_z = -6
                ri.Translate(cam_x,-cam_y,cam_z)
                ri.Rotate(-6.5,1,0,0)
                ri.Rotate(6,0,1,0)
        if (shot == 6) :
                cam_x = -0.3
                cam_y = smoothstep(-3.5,-4.5,700,950,frame)
                cam_z = smoothstep(-14.5,-17.5,700,950,frame)
                cam_z_nf = smoothstep(-14.5,-17.5,700,950,frame+1)
                ri.MotionBegin([-0.5,1])
                ri.Translate(cam_x,cam_y,cam_z)
                ri.Translate(cam_x,cam_y,cam_z_nf)
                ri.MotionEnd()
                ri.Rotate(-25,1,0,0)
                ri.Rotate(-3,0,1,0)


"""#############################################################################################
#########
# Scene light defintion function
#############################################################################################
#######"""
def scene_lights(ri,value):
        Back_light_intensity = 0
        Front_light_intensity = 0
        fg_coneAngle_01_intensity = 0
        fg_coneAngle_02_intensity = 0
        Fg_distance_intensity = 0
        Bg_distance_intensity = 0
        mag_intensity=0
        mag_intensity_scatter=0
        Light_control = value
        ### Distant light fg
        if ( Light_control == 1 ):
                Fg_distance_intensity = Fg_distance_intensity_default_shot_01
        if ( Light_control == 2 ):
                Fg_distance_intensity = Fg_distance_intensity_statue_shot_01
        if ( Light_control == 3 ):
                Fg_distance_intensity = Fg_distance_intensity_statue_scatter_shot_01

        ### Distant light bg
        if ( Light_control == 1 ):
                Bg_distance_intensity = Bg_distance_intensity_default_shot_01
```

```
        if ( Light_control == 2 ):
                Bg_distance_intensity = Bg_distance_intensity_statue_shot_01
        if ( Light_control == 3 ):
                Bg_distance_intensity = Bg_distance_intensity_statue_scatter_shot_01

        ### Back light
        if ( Light_control == 1 ):
                Back_light_intensity = Back_light_default_shot_01
        if ( Light_control == 2 ):
                Back_light_intensity = Back_light_intensity_statue_shot_01
        if ( Light_control == 3 ):
                Back_light_intensity = Back_light_intensity_statue_scatter_shot_01

        ### Front light
        if ( Light_control == 1 ):
                Front_light_intensity = Front_light_default_shot_01
        if ( Light_control == 2 ):
                Front_light_intensity = Front_light_intensity_statue_shot_01
        if ( Light_control == 3 ):
                Front_light_intensity = Front_light_intensity_statue_scatter_shot_01

        ### Spot light
        if ( Light_control == 1 ):
                fg_coneAngle_01_intensity = fg_coneAngle_01_intensity_default_shot_01
        if ( Light_control == 2 ):
                fg_coneAngle_01_intensity = fg_coneAngle_01_intensity_statue_shot_01
        if ( Light_control == 3 ):
                fg_coneAngle_01_intensity = fg_coneAngle_01_intensity_statue_scatter_shot_01

        ### Spot light
        if ( Light_control == 1 ):
                fg_coneAngle_02_intensity = fg_coneAngle_02_intensity_default_shot_01
        if ( Light_control == 2 ):
                fg_coneAngle_02_intensity = fg_coneAngle_02_intensity_statue_shot_01
        if ( Light_control == 3 ):
                fg_coneAngle_02_intensity = fg_coneAngle_02_intensity_statue_scatter_shot_01

        ##########################################################
        #Light defintions
        ##########################################################
        ri.LightSource("./shaders/occlusion/occlusionlight_v1_r1", {ri.HANDLEID:"occlusion_light",
"float  samples": [128], "float maxvariation":[0.002], "float maxdist":[10]})
        #ri.LightSource("./shaders/occlusion/environmentlight_v1_r1",
{ri.HANDLEID:"environment_light", "float  samples": [64], "string envmap":
["./sourceimages/hdr/Meadow_non_float.tx"]})
        ri.LightSource("./shaders/occlusion/environmentlight_v1_r1",
{ri.HANDLEID:"environment_light", "float  samples": [32],"float maxvariation":[0.02], "float
maxdist":[10], "string envmap":      ["./sourceimages/hdr/Meadow_non_float.tx"]})
        #ri.LightSource("./shaders/occlusion/environmentlight_v1_r1",
{ri.HANDLEID:"environment_light", "float  samples": [32],"float maxvariation":[0.002], "float
maxdist":[50], "string envmap":      ["./sourceimages/hdr/Reno_non_float.tx"]})

        ri.LightSource( "shadowdistant", {ri.HANDLEID:Fg_distance_light, "point
from":Fg_distance_light_From,  "point to":Fg_distance_light_To, "float intensity":
Fg_distance_intensity, "color lightcolor": [0.9,0.9,0.85], "string shadowname" :out_filename +
Fg_distance_light + ".%04d" %(frame) + ".shad", "float samples":[64],"float width":[2]})

        ri.LightSource( "shadowdistant", {ri.HANDLEID:Bg_distance_light, "point
from":Bg_distance_light_From,  "point to":Bg_distance_light_To, "float intensity":
Bg_distance_intensity, "color lightcolor": [0.9,0.9,0.85], "string shadowname" :"", "float samples":
[32],"float width":[1]})

        ri.LightSource( "distantlight", {ri.HANDLEID:"Light_front_fill", "point from":[10,10,10],
"point to":[-10,-10,-10], "float intensity": [Front_light_intensity], "color lightcolor":
[0.45,0.65,0.85]})
        ri.LightSource( "distantlight", {ri.HANDLEID:"Light_backlight", "point from":[0,-100,-100],
"point to":[0,100,100], "float intensity": [Back_light_intensity], "color lightcolor":
[0.95,0.95,0.75]})

        ri.LightSource ("ambientlight",{ri.HANDLEID: "Ambient","intensity" :[0.15], "color
lightcolor": [0.9,0.9,0.85]})

        # Foreground shadow lights
        ri.LightSource( "shadowspot", {ri.HANDLEID:fg_SpotName_01,"point from" : fg_SpotFrom_01,
"point to" : fg_SpotTo_01,"float intensity" : fg_coneAngle_01_intensity,"string shadowname"
:out_filename + fg_SpotName_01 + ".%04d" %(frame) + ".shad","float coneangle" :
fg_coneAngle_01,"float conedeltaangle" : [0.05], "color lightcolor": fg_coneAngle_01_colour, "float
```

```
beamdistribution":[2],"float samples":[128],"float width":[32]})
	ri.LightSource( "shadowspot", {ri.HANDLEID:fg_SpotName_02,"point from" : fg_SpotFrom_02,
"point to" : fg_SpotTo_02,"float intensity" : fg_coneAngle_02_intensity,"string shadowname"
:out_filename + fg_SpotName_02 + ".%04d" %(frame) + ".shad","float coneangle" :
fg_coneAngle_02,"float conedeltaangle" : [0.05], "color lightcolor": fg_coneAngle_02_colour, "float
beamdistribution":[2],"float samples":[128],"float width":[32]})

	# Magnifying glass control and movement
	mag_pos_x= 100
	mag_pos_y= 100
	mag_pos_z= 100
	mag_coneAngle=0

	if (shot == 3) :
		mag_intensity_scatter=23000
		mag_coneAngle=0.0075
		mag_pos_x= smoothstep(-8,-1.2,1,110,frame)
		mag_pos_y= smoothstep(2,5.6,1,110,frame)
		mag_pos_z= smoothstep(0,0,1,100,frame)
	if (shot == 4) :
		mag_intensity_scatter=smoothstep(20000,27500,200,300,frame)
		mag_coneAngle=smoothstep(0.0075,0.009,430,600,frame)
		mag_pos_x= smoothstep(-1.2,-1.2,300,500,frame)
		mag_pos_y= smoothstep(5.6,5.6,300,500,frame)
		mag_pos_z= smoothstep(0,0,300,500,frame)
	if (shot == 5) :
		mag_intensity_scatter=29000
		mag_coneAngle=smoothstep(0.009,0.009,430,600,frame)
		mag_pos_x= smoothstep(-1.2,-1.2,300,500,frame)
		mag_pos_y= smoothstep(5.6,5.6,300,500,frame)
		mag_pos_z= smoothstep(0,0,300,500,frame)
	if (shot == 6) :
		mag_intensity_scatter=smoothstep(20000,12000,50,400,frame)
		mag_coneAngle=smoothstep(0.01,0.011,50,400,frame)
		mag_pos_x= smoothstep(-1.0,-2.1,50,405,frame)
		mag_pos_y= smoothstep(4.75,1.4,60,390,frame)
		mag_pos_z= smoothstep(0,1.2,50,350,frame)
		if ( frame > 410):
			mag_intensity_scatter=smoothstep(20000,21000,410,500,frame)
			mag_coneAngle=smoothstep(0.011,0.011,410,500,frame)
			mag_pos_x= smoothstep(-2.1,-1.0,410,510,frame)
			mag_pos_y= smoothstep(1.4,4.15,410,500,frame)
			mag_pos_z= smoothstep(1.2,0,410,500,frame)
		if ( frame > 720):
			mag_intensity_scatter=smoothstep(21000,18000,725,825,frame)
			mag_coneAngle=smoothstep(0.011,0.011,725,825,frame)
			mag_pos_x= smoothstep(-1.0,-4.1,725,825,frame)
			mag_pos_y= smoothstep(4.15,7,725,825,frame)
			mag_pos_z= smoothstep(0,0,725,825,frame)

	if ( Light_control == 1 ):
		mag_intensity = mag_intensity_default
	if ( Light_control == 2 ):
		mag_intensity = mag_intensity_default
	if ( Light_control == 3 ):
		mag_intensity = mag_intensity_scatter

	noise_tran_x = noise(0.001,frame)
	noise_tran_y = noise(0.001,frame)
	noise_tran_z = noise(0.001,frame)

	mag_SpotTo=[mag_pos_x+noise_tran_x,mag_pos_y+noise_tran_y,mag_pos_z+noise_tran_z]

	ri.LightSource( "shadowspot", {ri.HANDLEID:mag_SpotName,"point from" : mag_SpotFrom, "point
to" : mag_SpotTo,"float intensity" : mag_intensity,"string shadowname" :out_filename + mag_SpotName
+ ".%04d" %(frame) + ".shad","float coneangle" :  mag_coneAngle,"float conedeltaangle" :
mag_coneDeltaAngle, "color lightcolor": mag_colour, "float  beamdistribution":[5],"float samples":
[128],"float width":[2]})

	# Magnifying light shadow inner
	ri.LightSource( "shadowspot", {ri.HANDLEID:shad_inner_mag_SpotName,"point from" :
mag_SpotFrom, "point to" : mag_SpotTo,"float intensity" : shad_inner_mag_intensity_default,"string
shadowname" :[""],"float coneangle" :  shad_inner_mag_coneAngle,"float conedeltaangle" :
shad_inner_mag_coneDeltaAngle, "color lightcolor": shad_inner_mag_colour, "float  beamdistribution":
[5],"float samples":[128],"float width":[2]})

	# Magnifying light shadow outter
```

```
        ri.LightSource( "shadowspot", {ri.HANDLEID:shad_outter_mag_SpotName,"point from" :
mag_SpotFrom, "point to" : mag_SpotTo,"float intensity" : shad_outter_mag_intensity_default,"string
shadowname" :[""],"float coneangle" :  shad_outter_mag_coneAngle,"float conedeltaangle" :
shad_outter_mag_coneDeltaAngle, "color lightcolor": shad_outter_mag_colour, "float
beamdistribution":[5],"float samples":[128],"float width":[2]})

"""##############################################################################################
#########
# Main loop of program were rendering is done over the specified frame range
##############################################################################################
#######"""
for frame in range(start_frame,end_frame):
        rendertype = in_rendertype
        if (rendertype == 2) :
                if ( compute_shadows == 1):
                        #to stop transparant objects messing shadows
                        rendertype =  1
                        #DistanceShadowPass(out_filename +
Fg_distance_light,Fg_distance_light_From,Fg_distance_light_To,Foreground_shadows,frame)
                        #DistanceShadowPass(out_filename +
Bg_distance_light,Bg_distance_light_From,Bg_distance_light_To,Background_shadows,frame)
                        ShadowPass(out_filename +
fg_SpotName_01,fg_SpotFrom_01,fg_SpotTo_01,fg_coneAngle_01,Foreground_shadows,frame,256)
                        ShadowPass(out_filename +
fg_SpotName_02,fg_SpotFrom_02,fg_SpotTo_02,fg_coneAngle_02,Foreground_shadows,frame,256)
                        #mag light mag_SpotName="mag_Spot"
                        #ShadowPass(out_filename +
mag_SpotName,mag_SpotFrom,mag_SpotTo,mag_coneAngle,SSS_objects_statue,frame,256)
                        #if ( compute_shadows == 1) :
                            # ShadowPass(out_filename +
SpotName_bg_01,SpotFrom_bg_01,SpotTo_bg_01,coneAngle_bg_01,Mid_ground,frame)
                            # ShadowPass(out_filename +
SpotName_bg_02,SpotFrom_bg_02,SpotTo_bg_02,coneAngle_bg_02,Background_shadows,frame)
                        point_cloud_check = 1
                        rendertype = in_rendertype
                        #create_point_cloud("statue_shot_%02d" %(shot),SSS_objects_statue,frame)
                        #create_point_cloud("blob", SSS_objects_blob_statue,frame)
        point_cloud_check = 1
        rendertype = in_rendertype
        #filename = out_filename + "%03d.rib" %(frame)
        filename = "__render"
        print "processing ",frame

        # this is the begining of the rib archive generation we can only
        # make RI calls after this function else we get a core dump
        ri.Begin(filename)

        # ArchiveRecord is used to add elements to the rib stream in this case comments
        # note the function is overloaded so we can concatinate output
        ri.ArchiveRecord(ri.COMMENT, 'File ' +filename)
        ri.ArchiveRecord(ri.COMMENT, "Created by " + getpass.getuser())
        ri.ArchiveRecord(ri.COMMENT, "Creation Date: " +time.ctime(time.time()))

        ri.PixelFilter( ri.MITCHELL ,4.0, 4.0)

        # now we add the display element using the usual elements
        # FILENAME DISPLAY Type Output format
        ri.Display( out_filename + "out_image_%02d_shot_%02d_frame_number.%03d.exr" %(renderlayer,
shot,frame), "openexr", "rgbaz", { "string filter": ["separable-catmull-rom"],"float[2] filterwidth"
: [4 ,4],"int[4] quantize" : [0, 0, 0 ,0],"float dither": [0],"float[2] exposure" :[1, 1]})
        #ri.Display( out_filename + "%03d.tif" %(frame), "framebuffer", "rgba")
        #ri.Display( out_filename + "_mask shot_%02d_%03d.tif" %(shot,frame), "file", "a")
        # Specify PAL resolution 1:1 pixel Aspect ratio
        ri.Format(res_x,res_y,1)
        ri.Clipping(0.001,7500)
        if (rendertype == 1) :
                  ri.Clipping(0.1,1000)
        #ri.Projection(ri.PERSPECTIVE,{ri.FOV:37.7777})
        ri.Projection(ri.PERSPECTIVE,{ri.FOV:37})
        #ri.ShadingInterpolation("constant")
        ri.ShadingRate([shading_rate])
        ri.PixelSamples(pixel_samples,pixel_samples)
        ri.Hider( "hidden", {"jitter" : [0]})
        #catmull-rom
        #ri.PixelFilter( ri.CUBIC ,4.0, 4.0)
        #ri.Option( "limits", {"bucketsize": [32,32]})
        #ri.Option( "limits", {"gridsize" :[32]})
```

```
        #ri.ShadingInterpolation("smooth")
        # calls camera translation to be loaded in to the scene
        cameras(ri,frame)
        focal_distance=0
        focal_length=0
        fstop=0
        if (shot == 1) :
                focal_distance = smoothstep(200,50,75,150,frame)
                focal_length = smoothstep(70,20,75,150,frame)
                fstop=smoothstep(10,10,75,150,frame)
                if(frame>250):
                        focal_distance = smoothstep(50,30,250,400,frame)
                        focal_length = smoothstep(20,2.5,250,400,frame)
                        fstop=smoothstep(10,10,250,400,frame)
                ri.GeometricApproximation("motionfactor", 3)
                ri.DepthOfField(fstop,focal_length,focal_distance)
                #Hider "hidden" "aperture" [nsides angle roundness density]
                #ri.Hider("hidden",{"aperture":[4,0,-0.2,0.2]})
        if (shot == 2) :
                focal_distance = 15
                focal_length = 2
                fstop = 40
                ri.DepthOfField(fstop,focal_length,focal_distance)
                ri.GeometricApproximation("motionfactor", 3)
                #Hider "hidden" "aperture" [nsides angle roundness density]
                #ri.Hider("hidden",{"aperture":[4,0,-0.2,0.2

        if (shot == 3) :
                focal_distance = 14
                focal_length = 4
                fstop = 25
                ri.DepthOfField(fstop,focal_length,focal_distance)
                ri.GeometricApproximation("motionfactor", 3)
                #Hider "hidden" "aperture" [nsides angle roundness density]
                #ri.Hider("hidden",{"aperture":[4,0,-0.2,0.2

        if (shot == 4) :
                focal_distance = 5
                focal_length = 4
                fstop = 35
                ri.DepthOfField(fstop,focal_length,focal_distance)
                ri.GeometricApproximation("motionfactor", 3)
                #ri.GeometricApproximation("motionfactor", 4)
                #Hider "hidden" "aperture" [nsides angle roundness density]
                #ri.Hider("hidden",{"aperture":[4,0,-0.2,0.2

        if (shot == 5) :
                focal_distance = 5.45
                focal_length = 4
                fstop = 35
                ri.DepthOfField(fstop,focal_length,focal_distance)
                ri.GeometricApproximation("motionfactor", 3)
                #ri.GeometricApproximation("motionfactor", 4)
                #Hider "hidden" "aperture" [nsides angle roundness density]
                #ri.Hider("hidden",{"aperture":[4,0,-0.2,0.2

        if (shot == 6) :
                focal_distance = smoothstep(15,10,700,950,frame)
                focal_length = 4
                fstop = 35
                ri.DepthOfField(fstop,focal_length,focal_distance)
                ri.GeometricApproximation("motionfactor", 3)
                #Hider "hidden" "aperture" [nsides angle roundness density]
                #ri.Hider("hidden",{"aperture":[4,0,-0.2,0.2

        #for occlusions environment lighting
        if (rendertype == 1) :
                ri.Attribute("visibility", {"int trace": 1})
                ri.Attribute("visibility", {"int diffuse" :1,"int specular": 1,"int transmission":
1})
                ri.Attribute("trace",  {"int maxdiffusedepth" :[2], "int maxspeculardepth" : [2],"int
displacements" : [1] , "bias" : [0.1],"int samplemotion" : [0]})
                ri.Attribute( "cull", {"hidden" : [0]})   # cull hidden surfaces
                ri.Attribute( "cull", {"backfacing" : [0]})   # cull backfacing surfaces
                ri.Attribute( "dice", {"rasterorient" : [1]}) # turn viewdependent gridding on

        #for Beauty pass
```

```
	if (rendertype == 2) :
		ri.Attribute("visibility", {"int diffuse" :1,"int specular": 1})
		#ri.Attribute("trace",  {"int maxdiffusedepth" :[1], "int maxspeculardepth" :
[1],"int displacements" : [1] , "bias" : [0.1],"int samplemotion" : [0]})
		ri.Attribute( "cull", {"hidden" : [1]})   # cull hidden surfaces
		ri.Attribute( "cull", {"backfacing" : [1]})   # cull backfacing surfaces
		ri.Attribute( "dice", {"rasterorient" : [1]}) # turn viewdependent gridding on

	#for magnifying glass
	if (rendertype == 3) :
		ri.Attribute("visibility", {"int trace": 1})
		ri.Attribute("visibility", {"int diffuse" :3,"int specular": 3,"int transmission":
1})
		ri.Attribute("trace",  {"int maxdiffusedepth" :[3], "int maxspeculardepth" : [3],"int
displacements" : [1] , "bias" : [0.1],"int samplemotion" : [0]})
		ri.Attribute( "cull", {"hidden" : [0]})   # cull hidden surfaces
		ri.Attribute( "cull", {"backfacing" : [0]})   # cull backfacing surfaces
		ri.Attribute( "dice", {"rasterorient" : [0]}) # turn viewdependent gridding on

	# now we start our world
	ri.WorldBegin()

	#Foreground statue
	if ( renderlayer == 2):
		ri.AttributeBegin()
		scene_lights(ri,2)
		if (rendertype == 1) :
			ri.Illuminate("environment_light",0)
			ri.Illuminate("occlusion_light",1)
			ri.Illuminate("Ambient",0)
			ri.Illuminate(Fg_distance_light,0)
			ri.Illuminate(Bg_distance_light,0)
			ri.Illuminate("Light_front_fill",0)
			ri.Illuminate("Light_backlight",0)
			ri.Illuminate(fg_SpotName_01,0)
			ri.Illuminate(fg_SpotName_02,0)
			ri.Illuminate(mag_SpotName,0)
			ri.Illuminate(shad_inner_mag_SpotName,0)
			ri.Illuminate(shad_outter_mag_SpotName,0)

		if (rendertype == 2) :
			ri.Illuminate("environment_light",0)
			ri.Illuminate("occlusion_light",0)
			ri.Illuminate("Ambient",0)
			ri.Illuminate(Fg_distance_light,0)
			ri.Illuminate(Bg_distance_light,0)
			ri.Illuminate("Light_front_fill",1)
			ri.Illuminate("Light_backlight",1)
			ri.Illuminate(fg_SpotName_01,1)
			ri.Illuminate(fg_SpotName_02,1)
			ri.Illuminate(mag_SpotName,1)
			ri.Illuminate(shad_inner_mag_SpotName,1)
			ri.Illuminate(shad_outter_mag_SpotName,1)

		ri.ShadingRate([shading_rate])
		#Statue(ri,frame,point_cloud_check,out_filename,rendertype,shot)
		Statue(ri,frame,point_cloud_check,out_filename,rendertype,shot)
		#Blob_Statue(ri,frame,point_cloud_check,out_filename,rendertype)

		ri.AttributeEnd()

	#Foreground objects
	if ( renderlayer == 1):
		ri.AttributeBegin()
		scene_lights(ri,1)
		if (rendertype == 1) :
			ri.Illuminate("environment_light",0)
			ri.Illuminate("occlusion_light",1)
			ri.Illuminate("Ambient",0)
			ri.Illuminate(Fg_distance_light,0)
			ri.Illuminate(Bg_distance_light,0)
			ri.Illuminate("Light_front_fill",0)
			ri.Illuminate("Light_backlight",0)
			ri.Illuminate(fg_SpotName_01,0)
			ri.Illuminate(fg_SpotName_02,0)
			ri.Illuminate(mag_SpotName,0)
			ri.Illuminate(shad_inner_mag_SpotName,0)
```

```
                ri.Illuminate(shad_outter_mag_SpotName,0)

        if (rendertype == 2) :
                ri.Illuminate("environment_light",0)
                ri.Illuminate("occlusion_light",0)
                ri.Illuminate("Ambient",0)
                ri.Illuminate(Fg_distance_light,0)
                ri.Illuminate(Bg_distance_light,0)
                ri.Illuminate("Light_front_fill",1)
                ri.Illuminate("Light_backlight",1)
                ri.Illuminate(fg_SpotName_01,1)
                ri.Illuminate(fg_SpotName_02,1)
                ri.Illuminate(mag_SpotName,1)
                ri.Illuminate(shad_inner_mag_SpotName,1)
                ri.Illuminate(shad_outter_mag_SpotName,1)

        ri.ShadingRate([shading_rate])

        Grass_forground(ri,frame,point_cloud_check,out_filename,rendertype)
        Dandilion(ri,frame,point_cloud_check,out_filename,rendertype)
        Rocks(ri,frame,point_cloud_check,out_filename,rendertype)
        Ground_foreground_2(ri,frame,point_cloud_check,out_filename,rendertype)
        Small_objects(ri,frame,point_cloud_check,out_filename,rendertype)


        #ri.ShadingRate([1])
        Fur(ri,frame,point_cloud_check,out_filename,rendertype)

        ri.ShadingRate([shading_rate*3])
        Ground_foreground_back_part(ri,frame,point_cloud_check,out_filename,rendertype)
        ri.AttributeEnd()

#Midground pass
if ( renderlayer == 2):
        ri.AttributeBegin()

        scene_lights(ri,1)
        if (rendertype == 1) :
                ri.Illuminate("environment_light",0)
                ri.Illuminate("occlusion_light",1)
                ri.Illuminate("Ambient",0)
                ri.Illuminate(Fg_distance_light,0)
                ri.Illuminate(Bg_distance_light,0)
                ri.Illuminate("Light_front_fill",0)
                ri.Illuminate("Light_backlight",0)
                ri.Illuminate(fg_SpotName_01,0)
                ri.Illuminate(fg_SpotName_02,0)
                ri.Illuminate(mag_SpotName,0)
                ri.Illuminate(shad_inner_mag_SpotName,0)
                ri.Illuminate(shad_outter_mag_SpotName,0)

        if (rendertype == 2) :
                ri.Illuminate("environment_light",0)
                ri.Illuminate("occlusion_light",0)
                ri.Illuminate("Ambient",0)
                ri.Illuminate(Fg_distance_light,0)
                ri.Illuminate(Bg_distance_light,1)
                ri.Illuminate("Light_front_fill",1)
                ri.Illuminate("Light_backlight",1)
                ri.Illuminate(fg_SpotName_01,0)
                ri.Illuminate(fg_SpotName_02,0)
                ri.Illuminate(mag_SpotName,0)
                ri.Illuminate(shad_inner_mag_SpotName,0)
                ri.Illuminate(shad_outter_mag_SpotName,0)

        ri.ShadingRate([shading_rate*3])
        Grass_midground(ri,frame,point_cloud_check,out_filename,rendertype)
        Grass_background(ri,frame,point_cloud_check,out_filename,rendertype)
        #ri.ShadingRate([1])
        #Trolly(ri,frame,point_cloud_check,out_filename,rendertype)

        ri.TransformBegin()
        ri.Translate(0,0,-200)
        #Ground_foreground_back_part(ri,frame,point_cloud_check,out_filename,rendertype)
        ri.TransformEnd()
        ri.AttributeEnd()
```

```
#Background pass
if ( renderlayer == 3):
	ri.AttributeBegin()

	scene_lights(ri,1)
	if (rendertype == 1) :
		ri.Illuminate("environment_light",0)
		ri.Illuminate("occlusion_light",1)
		ri.Illuminate("Ambient",0)
		ri.Illuminate(Fg_distance_light,0)
		ri.Illuminate(Bg_distance_light,0)
		ri.Illuminate("Light_front_fill",0)
		ri.Illuminate("Light_backlight",0)
		ri.Illuminate(fg_SpotName_01,0)
		ri.Illuminate(fg_SpotName_02,0)
		ri.Illuminate(mag_SpotName,0)
		ri.Illuminate(shad_inner_mag_SpotName,0)
		ri.Illuminate(shad_outter_mag_SpotName,0)

	if (rendertype == 2) :
		ri.Illuminate("environment_light",0)
		ri.Illuminate("occlusion_light",0)
		ri.Illuminate("Ambient",0)
		ri.Illuminate(Fg_distance_light,0)
		ri.Illuminate(Bg_distance_light,1)
		ri.Illuminate("Light_front_fill",1)
		ri.Illuminate("Light_backlight",1)
		ri.Illuminate(fg_SpotName_01,0)
		ri.Illuminate(fg_SpotName_02,0)
		ri.Illuminate(mag_SpotName,0)
		ri.Illuminate(shad_inner_mag_SpotName,0)
		ri.Illuminate(shad_outter_mag_SpotName,0)

	ri.ShadingRate([shading_rate])
	ri.GeometricApproximation("motionfactor", 1)
	Fence(ri,frame,point_cloud_check,out_filename,rendertype)
	Ivy(ri,frame,point_cloud_check,out_filename,rendertype)
	ri.GeometricApproximation("motionfactor", 3)
	ri.ShadingRate([shading_rate*2])
	#House(ri,frame,point_cloud_check,out_filename,rendertype)
	ri.ShadingRate([shading_rate*2])
	#Tree(ri,frame,point_cloud_check,out_filename,rendertype)

	ri.AttributeEnd()

#Sky/magnifying glass pass
if ( renderlayer == 4):
	ri.AttributeBegin()

	scene_lights(ri,1)
	if (rendertype == 1) :
		ri.Illuminate("environment_light",0)
		ri.Illuminate("occlusion_light",1)
		ri.Illuminate("Ambient",0)
		ri.Illuminate(Fg_distance_light,0)
		ri.Illuminate(Bg_distance_light,0)
		ri.Illuminate("Light_front_fill",0)
		ri.Illuminate("Light_backlight",0)
		ri.Illuminate(fg_SpotName_01,0)
		ri.Illuminate(fg_SpotName_02,0)
		ri.Illuminate(mag_SpotName,0)
		ri.Illuminate(shad_inner_mag_SpotName,0)
		ri.Illuminate(shad_outter_mag_SpotName,0)

	if (rendertype == 2) :
		ri.Illuminate("environment_light",0)
		ri.Illuminate("occlusion_light",0)
		ri.Illuminate("Ambient",0)
		ri.Illuminate(Fg_distance_light,0)
		ri.Illuminate(Bg_distance_light,1)
		ri.Illuminate("Light_front_fill",1)
		ri.Illuminate("Light_backlight",1)
		ri.Illuminate(fg_SpotName_01,0)
		ri.Illuminate(fg_SpotName_02,0)
		ri.Illuminate(mag_SpotName,0)
		ri.Illuminate(shad_inner_mag_SpotName,0)
		ri.Illuminate(shad_outter_mag_SpotName,0)
```

```
        ri.ShadingRate([1])

        #Grass_forground(ri,frame,point_cloud_check,out_filename,rendertype)

        #cam_y = smoothstep(45,1,200,500,frame)
        #cam_y_nf = smoothstep(45,1,200,500,frame+1)

        #SSS_objects_statue(ri,frame,point_cloud_check,out_filename,rendertype)
        if (shot == 2) :
                ri.TransformBegin()
                mag_x= smoothstep(0,0,1,300,frame)
                mag_x_nf= smoothstep(0,0,1,300,frame+1)
                mag_y= smoothstep(80,0,1,300,frame)
                mag_y_nf= smoothstep(80,0,1,300,frame+1)
                mag_z= smoothstep(90,0,1,300,frame)
                mag_z_nf= smoothstep(90,0,1,300,frame+1)
                #ri.Shutter(0,1)
                ri.MotionBegin([-0.5,1])
                ri.Translate(mag_x,mag_y,mag_z)
                ri.Translate(mag_x_nf,mag_y_nf,mag_z_nf)
                ri.MotionEnd()
                ri.TransformBegin()
                ri.Scale(3,3,3)
                ri.Translate(-20,42,20)
                ri.TransformBegin()
                ri.Rotate(12,0,0,1)
                ri.Rotate(3,0,1,0)
                Magnifying_glass(ri,frame,point_cloud_check,out_filename,rendertype)
                Lens(ri,frame,point_cloud_check,out_filename,rendertype)
                ri.TransformEnd()
                ri.TransformEnd()
                ri.TransformEnd()
                Sky_shot_02(ri,frame,point_cloud_check,out_filename,rendertype)
        else :
                Sky_shot_01(ri,frame,point_cloud_check,out_filename,rendertype)
        ri.AttributeEnd()

#Occlusion pass
if ( renderlayer == 5):
        ri.AttributeBegin()

        scene_lights(ri,1)

        ri.Illuminate("environment_light",0)
        ri.Illuminate("occlusion_light",1)
        ri.Illuminate("Ambient",0)
        ri.Illuminate(Fg_distance_light,0)
        ri.Illuminate(Bg_distance_light,0)
        ri.Illuminate("Light_front_fill",0)
        ri.Illuminate("Light_backlight",0)
        ri.Illuminate(fg_SpotName_01,0)
        ri.Illuminate(fg_SpotName_02,0)
        ri.Illuminate(mag_SpotName,0)
        ri.Illuminate(shad_inner_mag_SpotName,0)
        ri.Illuminate(shad_outter_mag_SpotName,0)

        ri.ShadingRate([shading_rate])
        Statue(ri,frame,point_cloud_check,out_filename,rendertype,shot)
        #Blob_Statue(ri,frame,point_cloud_check,out_filename,rendertype)
        Dandilion(ri,frame,point_cloud_check,out_filename,rendertype)
        #ri.Matte([1])

        #Dandilion(ri,frame,point_cloud_check,out_filename,rendertype)

        #Rocks(ri,frame,point_cloud_check,out_filename,rendertype)
        #Ground_foreground_2(ri,frame,point_cloud_check,out_filename,rendertype)
        ri.AttributeEnd()

#Environment pass
if ( renderlayer == 6):
        ri.AttributeBegin()

        scene_lights(ri,1)

        ri.Illuminate("environment_light",1)
        ri.Illuminate("occlusion_light",0)
```

```
        ri.Illuminate("Ambient",0)
        ri.Illuminate(Fg_distance_light,0)
        ri.Illuminate(Bg_distance_light,0)
        ri.Illuminate("Light_front_fill",0)
        ri.Illuminate("Light_backlight",0)
        ri.Illuminate(fg_SpotName_01,0)
        ri.Illuminate(fg_SpotName_02,0)
        ri.Illuminate(mag_SpotName,0)
        ri.Illuminate(shad_inner_mag_SpotName,0)
        ri.Illuminate(shad_outter_mag_SpotName,0)

        ri.ShadingRate([shading_rate])
        Statue(ri,frame,point_cloud_check,out_filename,rendertype,shot)
        Dandilion(ri,frame,point_cloud_check,out_filename,rendertype)
        #Rocks(ri,frame,point_cloud_check,out_filename,rendertype)
        #Ground_foreground_2(ri,frame,point_cloud_check,out_filename,rendertype)

        ri.AttributeEnd()

#Dirt pass
if ( renderlayer == 7):
        ri.AttributeBegin()

        scene_lights(ri,1)

        ri.Illuminate("environment_light",1)
        ri.Illuminate("occlusion_light",0)
        ri.Illuminate("Ambient",0)
        ri.Illuminate(Fg_distance_light,0)
        ri.Illuminate(Bg_distance_light,0)
        ri.Illuminate("Light_front_fill",0)
        ri.Illuminate("Light_backlight",0)
        ri.Illuminate(fg_SpotName_01,0)
        ri.Illuminate(fg_SpotName_02,0)
        ri.Illuminate(mag_SpotName,0)
        ri.Illuminate(shad_inner_mag_SpotName,0)
        ri.Illuminate(shad_outter_mag_SpotName,0)

        ri.ShadingRate([shading_rate])
        Statue(ri,frame,point_cloud_check,out_filename,rendertype,shot)

        ri.AttributeEnd()

#Mask pass
if ( renderlayer == 8):
        ri.AttributeBegin()

        scene_lights(ri,1)

        ri.Illuminate("environment_light",0)
        ri.Illuminate("occlusion_light",0)
        ri.Illuminate("Ambient",0)
        ri.Illuminate(Fg_distance_light,0)
        ri.Illuminate(Bg_distance_light,0)
        ri.Illuminate("Light_front_fill",0)
        ri.Illuminate("Light_backlight",0)
        ri.Illuminate(fg_SpotName_01,0)
        ri.Illuminate(fg_SpotName_02,0)
        ri.Illuminate(mag_SpotName,0)
        ri.Illuminate(shad_inner_mag_SpotName,0)
        ri.Illuminate(shad_outter_mag_SpotName,0)

        ri.ShadingRate([shading_rate])
        ri.Matte([0])
        Dandilion(ri,frame,point_cloud_check,out_filename,rendertype)

        ri.Matte([1])
        Grass_forground(ri,frame,point_cloud_check,out_filename,rendertype)
        Statue(ri,frame,point_cloud_check,out_filename,rendertype,shot)

        Rocks(ri,frame,point_cloud_check,out_filename,rendertype)
        Ground_foreground_2(ri,frame,point_cloud_check,out_filename,rendertype)
        Small_objects(ri,frame,point_cloud_check,out_filename,rendertype)
        Fur(ri,frame,point_cloud_check,out_filename,rendertype)
        Ground_foreground_back_part(ri,frame,point_cloud_check,out_filename,rendertype)

        ri.AttributeEnd()
```

```
        ri.WorldEnd()

        point_cloud_check = 0
        # and finally end the rib file
        ri.End()
```

archive_v6_r1_final.py

```
#!/usr/bin/python
# Created by Joe Gaffney
# For MSc Master project 2009
"""
Python module which contains all the functions that load in the scenes gemoetry through rib
archives.
Shaders are also set up and slso function which control geometry which are effected by shadow maps
or are
used in point cloud calculaitons. Random shader parameters are set on groups of objects of the same
type
to give varation. Also differnt gemometry is loaded for certain objects based on the current shot.
"""
# Modules loaded in
import sys
import math
import getpass
import time,random

sss_type = 1

##############################################################################################################
##############################
##############################################################################################################
##############################
# Shadow and point cloud functions
##############################################################################################################
##############################
##############################################################################################################
##############################

#Foreground_shadows
"""##########################################################################################################
#########
Function which contains a group of functions that load rib archives.
This is used to consolidate them into a group to allow easier specification of
which objects are effect/influence shadows
##############################################################################################################
#######"""
def Foreground_shadows(ri,frame,point_cloud_check,out_filename,rendertype,shot) :
        Grass_forground(ri,frame,point_cloud_check,out_filename,rendertype)
        Statue(ri,frame,point_cloud_check,out_filename,rendertype,shot)
        #Base_Statue(ri,frame,point_cloud_check,out_filename,rendertype)
        Ground_foreground_back_part(ri,frame,point_cloud_check,out_filename,rendertype)
        Ground_foreground_2(ri,frame,point_cloud_check,out_filename,rendertype)
        Rocks(ri,frame,point_cloud_check,out_filename,rendertype)
        Dandilion(ri,frame,point_cloud_check,out_filename,rendertype)
        Small_objects(ri,frame,point_cloud_check,out_filename,rendertype)
        #Fur(ri,frame,point_cloud_check,out_filename,rendertype)

#Background_shadows
"""##########################################################################################################
#########
Function which contains a group of functions that load rib archives.
This is used to consolidate them into a group to allow easier specification of
which objects are effect/influence shadows
##############################################################################################################
#######"""
def Background_shadows(ri,frame,point_cloud_check,out_filename,rendertype) :
        Grass_midground(ri,frame,point_cloud_check,out_filename,rendertype)
        Grass_background(ri,frame,point_cloud_check,out_filename,rendertype)
        House(ri,frame,point_cloud_check,out_filename,rendertype)
        Tree(ri,frame,point_cloud_check,out_filename,rendertype)
        Trolly(ri,frame,point_cloud_check,out_filename,rendertype)
        Fence(ri,frame,point_cloud_check,out_filename,rendertype)
        Ivy(ri,frame,point_cloud_check,out_filename,rendertype)

#Function to control the statues subsurface scattering geometry used in first part
```

```
#of the process of creating subsurface scattering effect.
"""#############################################################################################
#########
Loads rib archive of geometry and assigns surface and displacement shaders and applies a
displacement bound,
Also corrects orientation of the objects that are loaded in.
#############################################################################################
#######"""
def SSS_objects_statue(ri,frame,point_cloud_check,out_filename,rendertype,shot) :
        random.seed(25)
        statue_type =shot
        ri.TransformBegin()
        ri.Scale(-1.0,1.0,1.0)

#############################################################################################
##########
        # Statue

#############################################################################################
##########
        ri.TransformBegin()
        if(statue_type == 1):
                ri.ReadArchive("./Archive/forground/final_statue/shot_1_and_2/final_statue.%04d.rib"
%(1))
        if(statue_type == 2):

        ri.ReadArchive("./Archive/statue/statue_shot_three_four/first_and_second_hole_ribs.0001.rib")
        if(statue_type == 3):
                if(frame<100):

        ri.ReadArchive("./Archive/statue/statue_shot_three_four/first_and_second_hole_ribs.%04d.rib"
%(1))
                if(frame>=101):

        ri.ReadArchive("./Archive/statue/statue_shot_three_four/first_and_second_hole_ribs.%04d.rib"
%(frame-100))
        if(statue_type == 4):
                if(frame<100):

        ri.ReadArchive("./Archive/statue/statue_shot_three_four/first_and_second_hole_ribs.%04d.rib"
%(1))
                if(frame>=101):

        ri.ReadArchive("./Archive/statue/statue_shot_three_four/first_and_second_hole_ribs.%04d.rib"
%(frame-100))
        if(statue_type == 5):
                ri.ReadArchive("./Archive/statue/statue_shot_three_four/first_and_second_hole_ribs.
%04d.rib" %(1))
                ri.ReadArchive("./Archive/statue/particle_chunk_shot_five_six/particle_fluid_wax.
%04d.rib" %(frame))
        if(statue_type == 6):
                #ri.ReadArchive("./Archive/statue/statue_final_shot/3rd_and_fourth_hole_ribs.
%04d.rib" %(490))
                ri.ReadArchive("./Archive/statue/RIBS/3rd_and_fourth_hole_ribs.%04d.rib" %
(frame+400))
                #ri.ReadArchive("./Archive/statue/particle_chunk_shot_five_six/particle_fluid_wax.
%04d.rib" %(130))
        ri.TransformEnd()
        ri.TransformEnd()

#Function to control the statues blob present in shot 5 and 6 subsurface scattering geometry used in
first part
#of the process of creating subsurface scattering effect.
"""#############################################################################################
#########
Loads rib archive of geometry and assigns surface and displacement shaders and applies a
displacement bound,
Also corrects orientation of the objects that are loaded in.
#############################################################################################
#######"""
def SSS_objects_blob_statue(ri,frame,point_cloud_check,out_filename,rendertype) :
        random.seed(25)

        ri.TransformBegin()
        ri.Scale(-1.0,1.0,1.0)

#############################################################################################
```

```
##########
        # Statue

#############################################################################################################
##########
        ri.TransformBegin()
        if(frame>500):
                ri.ReadArchive("./Archive/statue/particle_chunk_shot_five_six/particle_fluid_wax.
%04d.rib" %(frame-500))
        ri.TransformEnd()
        ri.TransformEnd()

#Function to control the geometry to be used for the scene unfortunatly can not be used
#as point cloud api is currently not surported and the hack method of using a shader to
#generate the point cloud will not work on large amounts of geometry
"""#########################################################################################################
#########
Loads rib archive of geometry and assigns surface and displacement shaders and applies a
displacement bound,
Also corrects orientation of the objects that are loaded in.
#########################################################################################################
#######"""
def SSS_objects_all(ri,frame,point_cloud_check,out_filename,rendertype) :
        random.seed(25)

        ri.TransformBegin()
        ri.Scale(-1.0,1.0,1.0)
        #pulls camera back to allow render
        ri.Translate(0,0,100)
        ri.TransformBegin()
        #ri.ReadArchive("./Archive/statue_rib_archive_v1_r1.rib")
        ri.ReadArchive("./Archive/grass/foreground/grass_foreground_r1_v1_sequence_.%04d.rib" %
(frame))
        #ri.ReadArchive("./Archive/grass/midground/grass_midground_v1_r1_sequence_.%04d.rib" %
(frame))
        #ri.ReadArchive("./Archive/close_up_ground_v1_r1.rib")
        #ri.ReadArchive("./Archive/rocks_rib_archive_v1_r1.rib")
        ri.TransformEnd()
        ri.TransformEnd()

#############################################################################################################
############################
#############################################################################################################
############################
# Foreground functions
#############################################################################################################
############################
#############################################################################################################
############################

#############################################################################################################
############################
#Function to control the statue
"""#########################################################################################################
#########
Loads rib archive of geometry and assigns surface and displacement shaders and applies a
displacement bound,
Also corrects orientation of the objects that are loaded in.
#########################################################################################################
#######"""
def Statue(ri,frame,point_cloud_check,out_filename,rendertype,shot) :
        statue_type =shot
        random.seed(25)
        ri.TransformBegin()
        ri.Scale(-1.0,1.0,1.0)
        Name = "statue_shot_%02d" %(shot)
        print Name

#############################################################################################################
##########
        # Statue

#############################################################################################################
##########
        ri.TransformBegin()
        ri.AttributeBegin()
        if (rendertype == 1) :
```

```
                #ri.Surface("./shaders/occlusion/occlusion_v1_r1",{ "float  samples": [32]})
                ri.Surface("matte")
        if (rendertype == 2) :
                if ( point_cloud_check == 1 ):
                        #apple
                        if(sss_type == 1):
                                ri.Surface("./shaders/sss/render_ssdiffusion",{ "uniform string
filename": [out_filename + Name + ".%04d.ptc" %(frame)],
                                                "color albedo": [0.746, 0.741, 0.428],
                                                "color dmfp":  [6.96,  6.40,  1.50],
                                                "float ior": [1.5],
                                                "float unitlength": [1.0],
                                                "float smooth": [1],
                                                "float maxsolidangle": [1],
                                                "float Ka": [0.4],
                                                "float Kd" : [0.35],
                                                "float Ks": [0.4],
                                                "float roughness": [0.2]})
                        if(sss_type == 2):
                                ri.Surface("./shaders/sss/render_ssdiffusion",{ "uniform string
filename": [out_filename + Name + ".%04d.ptc" %(frame)],
                                                "color albedo": [0.746, 0.741, 0.428],
                                                "color dmfp":  [4.96,  4.40,  1.50],
                                                "float ior": [1.5],
                                                "float unitlength": [1.0],
                                                "float smooth": [1],
                                                "float maxsolidangle": [1],
                                                "float Ka": [0.4],
                                                "float Kd" : [0.4],
                                                "float Ks": [0.45],
                                                "float roughness": [0.2]})
                ri.Displacement("./shaders/ground/ground_displacement_v2_r1",{ "float Km":[0.004],
"float Freq":[100], "float Amplitude":[2.5], "float Layers":[8],"string space":["shader"]})
                ri.Attribute("displacementbound",{ri.COORDINATESYSTEM:["object"],"uniform float
sphere":[0.2]})
        if(statue_type == 1):
                #ri.ReadArchive("./Archive/forground/final_statue/shot_1_and_2/final_statue.%04d.rib"
%(1))

        ri.ReadArchive("./Archive/statue/statue_shot_three_four/first_and_second_hole_ribs.0001.rib")
        if(statue_type == 2):

        ri.ReadArchive("./Archive/statue/statue_shot_three_four/first_and_second_hole_ribs.0001.rib")
        if(statue_type == 3):
                if(frame<100):

        ri.ReadArchive("./Archive/statue/statue_shot_three_four/first_and_second_hole_ribs.%04d.rib"
%(1))
                if(frame>=101):

        ri.ReadArchive("./Archive/statue/statue_shot_three_four/first_and_second_hole_ribs.%04d.rib"
%(frame-100))
        if(statue_type == 4):
                if(frame<100):

        ri.ReadArchive("./Archive/statue/statue_shot_three_four/first_and_second_hole_ribs.%04d.rib"
%(1))
                if(frame>=101):

        ri.ReadArchive("./Archive/statue/statue_shot_three_four/first_and_second_hole_ribs.%04d.rib"
%(frame-100))
        if(statue_type == 5):
                ri.ReadArchive("./Archive/statue/statue_shot_three_four/first_and_second_hole_ribs.
%04d.rib" %(1))
                #ri.ReadArchive("./Archive/statue/particle_chunk_shot_five_six/particle_fluid_wax.
%04d.rib" %(frame))
        if(statue_type == 6):
                #ri.ReadArchive("./Archive/statue/statue_final_shot/3rd_and_fourth_hole_ribs.
%04d.rib" %(490))
                ri.ReadArchive("./Archive/statue/RIBS/3rd_and_fourth_hole_ribs.%04d.rib" %
(frame+400))
                #ri.ReadArchive("./Archive/statue/particle_chunk_shot_five_six/particle_fluid_wax.
%04d.rib" %(130))
        ri.AttributeEnd()
        ri.TransformEnd()

        ri.TransformEnd()
```

```
#Function to control the blob componet of the statue found in shot 5 and 6
"""#############################################################################################
#########
Loads rib archive of geometry and assigns surface and displacement shaders and applies a
displacement bound,
Also corrects orientation of the objects that are loaded in.
#############################################################################################
#######"""
def Blob_Statue(ri,frame,point_cloud_check,out_filename,rendertype) :
        random.seed(25)
        ri.TransformBegin()
        ri.Scale(-1.0,1.0,1.0)


#############################################################################################
##########
        # Statue

#############################################################################################
##########
        ri.TransformBegin()
        ri.AttributeBegin()
        if (rendertype == 1) :
                #ri.Surface("./shaders/occlusion/occlusion_v1_r1",{ "float  samples": [32]})
                ri.Surface("matte")
        if (rendertype == 2) :
                if ( point_cloud_check == 1 ):
                        #apple
                        if(sss_type == 1):
                                ri.Surface("./shaders/sss/render_ssdiffusion",{ "uniform string
filename": [out_filename + "blob" + ".%04d.ptc" %(frame)],
                                        "color albedo": [0.746, 0.741, 0.428],
                                        "color dmfp":  [8.96,  8.40,  3.50],
                                        "float ior": [1.5],
                                        "float unitlength": [1.0],
                                        "float smooth": [1],
                                        "float maxsolidangle": [1],
                                        "float Ka": [0.4],
                                        "float Kd" : [0.35],
                                        "float Ks": [0.4],
                                        "float roughness": [0.2]})
                        if(sss_type == 2):
                                ri.Surface("./shaders/sss/render_ssdiffusion",{ "uniform string
filename": [out_filename + "blob" + ".%04d.ptc" %(frame)],
                                        "color albedo": [0.746, 0.741, 0.428],
                                        "color dmfp":  [4.96,  4.40,  1.50],
                                        "float ior": [1.5],
                                        "float unitlength": [1.0],
                                        "float smooth": [1],
                                        "float maxsolidangle": [1],
                                        "float Ka": [0.4],
                                        "float Kd" : [0.4],
                                        "float Ks": [0.45],
                                        "float roughness": [0.2]})
                ri.Displacement("./shaders/ground/ground_displacement_v2_r1",{ "float Km":[0.004],
"float Freq":[30], "float Amplitude":[7], "float Layers":[8],"string space":["shader"]})
                ri.Attribute("displacementbound",{ri.COORDINATESYSTEM:["object"],"uniform float
sphere":[0.2]})
        ri.ReadArchive("./Archive/statue/particle_chunk_shot_five_six/particle_fluid_wax.%04d.rib" %
(frame))
        ri.AttributeEnd()
        ri.TransformEnd()

        ri.TransformEnd()

#Function to control magnifying glass outter rim
"""#############################################################################################
#########
Loads rib archive of geometry and assigns surface and displacement shaders and applies a
displacement bound,
Also corrects orientation of the objects that are loaded in.
#############################################################################################
#######"""
def Magnifying_glass(ri,frame,point_cloud_check,out_filename,rendertype):
        ri.TransformBegin()
        ri.Scale(-1.0,1.0,1.0)
```

```
        ri.TransformBegin()
        #ri.Scale(1.3,1.3,1.3)

##############################################################################################################
##########
        # Magnifying glass outer

##############################################################################################################
##########
        ri.TransformBegin()
        ri.AttributeBegin()

        ri.Surface("shinymetal",{"float Ka":[0.01],"float Ks":[0.2],"float Kr":[0.3],"float
roughness":[2.2],"string texturename":["./sourceimages/hdr/Meadow_non_float.tx"]})
        #ri.Surface("plastic",{"float Ks":[0.23],"float Kd":[0.6],"float Ka":[0.3],"float
roughness":[2.1],"color specularcolor":[1,1,1]})
        ri.Displacement("./shaders/ground/ground_displacement_v2_r1",{ "float Km":[0.05], "float
Freq":[0.2], "float Amplitude":[2], "float Layers":[6],"string   space":["shader"]})
        ri.Attribute("displacementbound",{ri.COORDINATESYSTEM:["object"],"uniform float sphere":
[0.1]})
        ri.ReadArchive("./Archive/new_magnifying_glass/OCoutter_v1.rib")
        ri.AttributeEnd()
        ri.TransformEnd()

        ri.TransformEnd()
        ri.TransformEnd()

#Funcition to control magnifying glass lens
"""##############################################################################################################
#########
Loads rib archive of geometry and assigns surface and displacement shaders and applies a
displacement bound,
Also corrects orientation of the objects that are loaded in.
##############################################################################################################
#######"""
def Lens(ri,frame,point_cloud_check,out_filename,rendertype):
        ri.TransformBegin()
        ri.Scale(-1.0,1.0,1.0)

##############################################################################################################
##########
        # Lens

##############################################################################################################
##########
        ri.TransformBegin()
        ri.AttributeBegin()
        #ri.Surface("aaglass")
        ri.Surface("glassrefr",{"float Kr":[1.0],"float Kt":[1.0],"float ior":[1.6],"float Ks":
[1.3],"float shinyness":[30]})
        ri.Displacement("./shaders/grass/grass_displacement_shader_v1_r3",{ "float freq_layer_one":
[10], "float freq_layer_two":[5], "float depth_one":[0.01], "float depth_two":[0.01]})
        ri.Attribute("displacementbound",{ri.COORDINATESYSTEM:["object"],"uniform float sphere":
[0.02]})
        #ri.Surface("glassbal",{"float Kd":[1.0], "float eta":[1.6],"string envname":
["./sourceimages/environment_maps/map_02.tx"]})
        #ri.Surface("./shaders/lens/lens_v1_r1",{"float Kd":[0.8],"float Ks":[0.5],"float
roughness":[10.4],"float Kenv":[0.5],"string EnvironmentMap":
["./sourceimages/environment_maps/map_02.tx"],"float Kfrac":[0.8]})
        #ri.Surface("./shaders/lens/lens_v1_r2",{"float Kd":[0.4],"float Kfrac":[1.5]})
        ri.ReadArchive("./Archive/new_magnifying_glass/OClens_join_v1.rib")
        #ri.ReadArchive("./Archive/new_magnifying_glass/OClens_small_v1.rib")
        ri.AttributeEnd()
        ri.TransformEnd()

        ri.TransformEnd()

#Function to control the foreground grass
"""##############################################################################################################
#########
Loads rib archive of geometry and assigns surface and displacement shaders and applies a
displacement bound,
Also corrects orientation of the objects that are loaded in.
##############################################################################################################
#######"""
def Grass_forground(ri,frame,point_cloud_check,out_filename,rendertype) :
        random.seed(20)
```

```
        ri.TransformBegin()
        ri.Scale(-1.0,1.0,1.0)

        ri.TransformBegin()

##############################################################################################################
##########
        # Grass foreground

##############################################################################################################
##########
        ri.TransformBegin()
        for i in range(1, 19):
                ri.AttributeBegin()
                ran_diffuse=random.uniform(0.9,1.1)
                ran_opacity=random.uniform(1,3)
                ran_tex=random.uniform(1,3)

        ran_diffuse_front=[random.uniform(1.0,1.1),random.uniform(0.8,0.9),random.uniform(0.8,0.9)]

        ran_diffuse_back=[random.uniform(1.0,1.0),random.uniform(0.9,1.1),random.uniform(1.0,1.1)]
                ran_specular=random.uniform(1.2,1.3)

        ran_spec_colour=[random.uniform(0.4,0.5),random.uniform(0.5,0.6),random.uniform(0.5,0.6)]
                if (rendertype == 1) :
                        ri.Surface("matte")
                if (rendertype == 2) :
                        ri.Surface("./shaders/new_grass/new_grass_surface_shader_v3_r1",{"float
Ks":ran_specular, "color specularcolor":ran_spec_colour, "float Kd":ran_diffuse,"float Ka":
[1.0],"color front_colour":ran_diffuse_front,"color back_colour":ran_diffuse_back,"string
texture_front":["./sourceimages/grass/grass_front_high_v1.tx"],"string  texture_back":
["./sourceimages/grass/grass_front_high_v%01d.tx" %(ran_tex)],"color opacity":[1.0,1.0,1.0],"float
roughness":[8.5], "float max_size":[5],"string  texture_opacity":
["./sourceimages/grass/grass_opacity_high_v%01d.tx" %(ran_opacity)],"float Kenv":[0.2],"string
EnvironmentMap":["./sourceimages/hdr/Meadow_non_float.tx"],"float oren_roughness":[5.0]})
                ri.Displacement("./shaders/grass/grass_displacement_shader_v1_r3",{ "float
freq_layer_one":[100], "float freq_layer_two":[40], "float depth_one":[0.025], "float depth_two":
[0.04]})
                ri.Attribute("displacementbound",{ri.COORDINATESYSTEM:["object"],"uniform float
sphere":[0.2]})
                ri.Attribute ("identifier",{"name": "grass_foreground"})

                ri.ReadArchive("./Archive/grass/new_foreground_grass/grass_sperate/OCgrass_piece_
%02d_.%04d.rib" %((i), (frame)))
                ri.AttributeEnd()
        ri.TransformEnd()

        ri.TransformEnd()

        ri.TransformEnd()

#Function to control the foreground ground
"""#############################################################################################################
#########
Loads rib archive of geometry and assigns surface and displacement shaders and applies a
displacement bound,
Also corrects orientation of the objects that are loaded in.
#############################################################################################################
#######"""
def Ground_foreground(ri,frame,point_cloud_check,out_filename,rendertype) :
        ri.TransformBegin()
        ri.Scale(-1.0,1.0,1.0)

        ri.TransformBegin()

##############################################################################################################
##########
        # Close up ground

##############################################################################################################
##########
        ri.TransformBegin()
        ri.AttributeBegin()
        if (rendertype == 1) :
                #ri.Surface("./shaders/occlusion/occlusion_v1_r1",{ "float  samples": [32]})
                ri.Surface("matte")
        if (rendertype == 2) :
```

```
                ri.Surface("./shaders/rocks/rock_surface_v3_r2",{"float Ka":[0.5],"float Kd":
[0.8],"float roughness":[10.0],"float spec_roughness":[0.2],"float Ks":[0.4],"float veining":
[200],"color diffusecolor":[1.0,1.0,1.0],"color color_multi_one":[1.0,1.0,1.0],"color
color_multi_two":[0.9,0.9,0.9],"color specularcolor":[0.7,0.7,0.7],"float Threshold":[0.7],"float
Layers_of_spots":[3],"float specksize":[0.01],"float sizes":[5],"color spattercolor":
[0.6,0.6,0.6],"float Kenv":[0.3],"string  envname":
["./sourceimages/hdr/Meadow_non_float.tx"],"string  texture_front":
["./sourceimages/Ground/ground_high_res_v1.tx"]})
                ri.Displacement("./shaders/ground/ground_displacement_v2_r1",{ "float Km":[0.2],
"float Freq":[700], "float Amplitude":[1], "float Layers":[6]})
                ri.Attribute("displacementbound",{ri.COORDINATESYSTEM:["object"],"uniform float
sphere":[0.2]})
        ri.Attribute ("identifier",{"name": "grass_01"})
        ri.ReadArchive("./Archive/forground/ground/OCground_v1_r1_.rib")
        ri.AttributeEnd()
        ri.TransformEnd()

        ri.TransformEnd()

        ri.TransformEnd()

#Function to control alternative version of the foreground ground
"""#############################################################################################
#########
Loads rib archive of geometry and assigns surface and displacement shaders and applies a
displacement bound,
Also corrects orientation of the objects that are loaded in.
#############################################################################################
#######"""
def Ground_foreground_2(ri,frame,point_cloud_check,out_filename,rendertype) :
        ri.TransformBegin()
        ri.Scale(-1.0,1.0,1.0)

        ri.TransformBegin()

#############################################################################################
##########
        # Close up ground

#############################################################################################
##########
        ri.TransformBegin()
        ri.AttributeBegin()
        if (rendertype == 1) :
                #ri.Surface("./shaders/occlusion/occlusion_v1_r1",{ "float  samples": [32]})
                ri.Surface("matte")
        if (rendertype == 2) :
                ri.Surface("./shaders/rocks/rock_surface_v2_r4",{"float Ka":[0.5],"float Kd":
[0.8],"float roughness":[10.0],"float spec_roughness":[0.2],"float Ks":[1.1],"float veining":
[200],"color diffusecolor":[1.0,1.0,1.0],"color color_multi_one":[1.0,1.0,1.0],"color
color_multi_two":[0.9,0.9,0.9],"color specularcolor":[0.8,0.7,0.6],"float Threshold":[0.7],"float
Layers_of_spots":[3],"float specksize":[0.01],"float sizes":[5],"color spattercolor":
[0.6,0.6,0.6],"float Kenv":[0.3],"string  envname":
["./sourceimages/hdr/Meadow_non_float.tx"],"string  texture_front":
["./sourceimages/Ground/ground_high_res_v1.tx"]})
        ri.Displacement("./shaders/ground/ground_displacement_v2_r1",{ "float Km":[0.2], "float
Freq":[700], "float Amplitude":[1.1], "float Layers":[10]})
        ri.Attribute("displacementbound",{ri.COORDINATESYSTEM:["object"],"uniform float sphere":
[0.2]})
        ri.Attribute ("identifier",{"name": "grass_01"})
        ri.ReadArchive("./Archive/forground/ground/OCground_v1_r1_.rib")
        ri.AttributeEnd()
        ri.TransformEnd()

        ri.TransformEnd()

        ri.TransformEnd()

#Function to control the foreground ground back part
"""#############################################################################################
#########
Loads rib archive of geometry and assigns surface and displacement shaders and applies a
displacement bound,
Also corrects orientation of the objects that are loaded in.
#############################################################################################
#######"""
def Ground_foreground_back_part(ri,frame,point_cloud_check,out_filename,rendertype) :
```

```
        ri.TransformBegin()
        ri.Scale(-1.0,1.0,1.0)

        ri.TransformBegin()

##############################################################################################################
##########
        # Close up ground

##############################################################################################################
##########
        ri.TransformBegin()
        ri.AttributeBegin()
        if (rendertype == 1) :
                #ri.Surface("./shaders/occlusion/occlusion_v1_r1",{ "float  samples": [32]})
                ri.Surface("matte")
        if (rendertype == 2) :
                ri.Surface("./shaders/rocks/rock_surface_v2_r4",{"float Ka":[0.5],"float Kd":
[0.8],"float roughness":[10.0],"float spec_roughness":[0.2],"float Ks":[1.1],"float veining":
[200],"color diffusecolor":[1.0,1.0,1.0],"color color_multi_one":[1.0,1.0,1.0],"color
color_multi_two":[0.9,0.9,0.9],"color specularcolor":[0.8,0.7,0.6],"float Threshold":[0.7],"float
Layers_of_spots":[3],"float specksize":[0.01],"float sizes":[5],"color spattercolor":
[0.6,0.6,0.6],"float Kenv":[0.3],"string  envname":
["./sourceimages/hdr/Meadow_non_float.tx"],"string  texture_front":
["./sourceimages/Ground/ground_high_res_v1.tx"]})
        ri.Displacement("./shaders/ground/ground_displacement_v2_r1",{ "float Km":[0.2], "float
Freq":[700], "float Amplitude":[1.1], "float Layers":[10]})
        ri.Attribute("displacementbound",{ri.COORDINATESYSTEM:["object"],"uniform float sphere":
[0.2]})
        ri.Attribute ("identifier",{"name": "grass_01"})
        ri.ReadArchive("./Archive/forground/ground/OCground_fg_back_part_v1.rib")
        ri.AttributeEnd()
        ri.TransformEnd()

        ri.TransformEnd()

        ri.TransformEnd()

#Function to control the foreground rocks
"""##############################################################################################################
#########
Loads rib archive of geometry and assigns surface and displacement shaders and applies a
displacement bound,
Also corrects orientation of the objects that are loaded in.
##############################################################################################################
#######"""
def Rocks(ri,frame,point_cloud_check,out_filename,rendertype) :
        random.seed(20)
        ri.TransformBegin()
        ri.Scale(-1.0,1.0,1.0)
        ri.Translate(0.0,-0.2,0.0)
        ri.TransformBegin()

##############################################################################################################
##########
        # Small rocks

##############################################################################################################
##########
        ri.TransformBegin()
        for i in range(1, 10):
                ri.AttributeBegin()
                ran_tex=random.uniform(1,3)
                ran_displace=random.uniform(0.01,0.016)
                ran_vien=random.uniform(4,5)
                ran_spot_colour_multi=random.uniform(0.7,0.9)


ran_spot_colour=[random.uniform(ran_spot_colour_multi,ran_spot_colour_multi+0.05),random.uniform(ran
_spot_colour_multi,ran_spot_colour_multi+0.05),random.uniform(ran_spot_colour_multi,ran_spot_colour_
multi+0.05)]
                ran_diffuse=[random.uniform(0.6,0.7),random.uniform(0.6,0.7),random.uniform(0.6,0.7)]
                ran_spec=random.uniform(0.2,0.7)
                ran_spot_layers=random.uniform(1,1)
                if (rendertype == 1) :
                        ri.Surface("./shaders/occlusion/occlusion_v1_r1",{ "float  samples": [32]})
                        ri.Surface("matte")
```

```
                if (rendertype == 2) :
                        ri.Surface("./shaders/rocks/rock_surface_v2_r4",{"float Ka":[0.5],"float Kd":
[1.0],"float roughness":[10.0],"float spec_roughness":[1.2],"float Ks":[0.4],"float veining":
ran_vien,"color diffusecolor":ran_diffuse,"color color_multi_one":[0.922,0.922,0.922],"color
color_multi_two":[1.0,1.0,1.0],"color specularcolor":[ran_spec,ran_spec,ran_spec],"float Threshold":
[0.7],"float Layers_of_spots":ran_spot_layers,"float specksize":[0.01],"float sizes":ran_vien,"color
spattercolor":ran_spot_colour,"float Kenv":[1.0],"string  envname":
["./sourceimages/hdr/Meadow_non_float.tx"],"string  texture_front":["./sourceimages/rocks/rock_v
%01d.tx" %(ran_tex)]})
                ri.Displacement("./shaders/ground/ground_displacement_v2_r1",{ "float Km":
[ran_displace], "float Freq":[2], "float Amplitude":[30], "float Layers":[12]})
                ri.Attribute("displacementbound",{ri.COORDINATESYSTEM:["object"],"uniform float
sphere":[0.2]})
                ri.ReadArchive("./Archive/forground/rocks/OCrocks_small_v1_r1_%03d.rib" %(i))
                ri.AttributeEnd()
        ri.TransformEnd()


##############################################################################################
##########
        # Large rocks

##############################################################################################
##########
        ri.TransformBegin()
        for i in range(1, 5):
                ri.AttributeBegin()
                ran_tex_large=random.uniform(1,4)
                ran_displace=random.uniform(0.02,0.040)
                ran_vien=random.uniform(4,5)
                ran_spot_colour_multi=random.uniform(0.8,0.9)


ran_spot_colour=[random.uniform(ran_spot_colour_multi,ran_spot_colour_multi+0.05),random.uniform(ran
_spot_colour_multi,ran_spot_colour_multi+0.05),random.uniform(ran_spot_colour_multi,ran_spot_colour_
multi+0.05)]
                ran_diffuse=[random.uniform(0.8,0.9),random.uniform(0.8,0.9),random.uniform(0.8,0.9)]
                ran_spec=random.uniform(0.5,0.6)
                ran_spot_layers=random.uniform(1,1)
                if (rendertype == 1) :
                        #ri.Surface("./shaders/occlusion/occlusion_v1_r1",{ "float  samples": [32]})
                        ri.Surface("matte")
                if (rendertype == 2) :
                        ri.Surface("./shaders/rocks/rock_surface_v2_r4",{"float Ka":[0.5],"float Kd":
[1.0],"float roughness":[10.0],"float spec_roughness":[1.2],"float Ks":[0.22],"float veining":
ran_vien,"color diffusecolor":ran_diffuse,"color color_multi_one":[0.922,0.922,0.922],"color
color_multi_two":[1.0,1.0,1.0],"color specularcolor":[ran_spec,ran_spec,ran_spec],"float Threshold":
[0.7],"float Layers_of_spots":ran_spot_layers,"float specksize":[0.01],"float sizes":ran_vien,"color
spattercolor":ran_spot_colour,"float Kenv":[0.3],"string  texture_front":
["./sourceimages/rocks/rock_v%01d.tx" %(ran_tex_large)],"string  envname":
["./sourceimages/hdr/Meadow_non_float.tx"]})
                ri.Displacement("./shaders/ground/ground_displacement_v2_r1",{ "float Km":
[ran_displace], "float Freq":[2], "float Amplitude":[30], "float Layers":[15]})
                ri.Attribute("displacementbound",{ri.COORDINATESYSTEM:["object"],"uniform float
sphere":[0.2]})
                ri.ReadArchive("./Archive/forground/rocks/OCrocks_large_v1_r1_%03d.rib" %(i))
                ri.AttributeEnd()
        ri.TransformEnd()

        random.seed(5)


##############################################################################################
##########
        # Medium rocks

##############################################################################################
##########
        ri.TransformBegin()
        for i in range(1, 7):
                ri.AttributeBegin()
                ran_tex=random.uniform(1,2)
                ran_displace=random.uniform(0.02,0.03)
                ran_vien=random.uniform(4,5)
                ran_spot_colour_multi=random.uniform(0.5,0.9)
```

```
ran_spot_colour=[random.uniform(ran_spot_colour_multi,ran_spot_colour_multi+0.05),random.uniform(ran
_spot_colour_multi,ran_spot_colour_multi+0.05),random.uniform(ran_spot_colour_multi,ran_spot_colour_
multi+0.05)]
                ran_diffuse=[random.uniform(0.8,0.9),random.uniform(0.8,0.9),random.uniform(0.8,0.9)]
                ran_spec=random.uniform(0.4,0.5)
                ran_spot_layers=random.uniform(1,1)
                if (rendertype == 1) :
                        #ri.Surface("./shaders/occlusion/occlusion_v1_r1",{ "float  samples": [32]})
                        ri.Surface("matte")
                if (rendertype == 2) :
                        ri.Surface("./shaders/rocks/rock_surface_v2_r4",{"float Ka":[0.5],"float Kd":
[1.0],"float roughness":[10.0],"float spec_roughness":[2.2],"float Ks":[0.45],"float veining":
ran_vien,"color diffusecolor":ran_diffuse,"color color_multi_one":[0.922,0.922,0.922],"color
color_multi_two":[1.0,1.0,1.0],"color specularcolor":[ran_spec,ran_spec,ran_spec],"float Threshold":
[0.7],"float Layers_of_spots":ran_spot_layers,"float specksize":[0.01],"float sizes":ran_vien,"color
spattercolor":ran_spot_colour,"float Kenv":[1.0],"string  envname":
["./sourceimages/hdr/Meadow_non_float.tx"],"string  texture_front":["./sourceimages/rocks/rock_v
%01d.tx" %(ran_tex)]})
                ri.Displacement("./shaders/ground/ground_displacement_v2_r1",{ "float Km":
[ran_displace], "float Freq":[2], "float Amplitude":[20], "float Layers":[14]})
                ri.Attribute("displacementbound",{ri.COORDINATESYSTEM:["object"],"uniform float
sphere":[0.2]})
                ri.ReadArchive("./Archive/forground/rocks/OCrocks_medium_v1_r1_%03d.rib" %(i))
                ri.AttributeEnd()
        ri.TransformEnd()
        ri.TransformEnd()

        ri.TransformEnd()

#Function to control Dandilion and stem and leaves
"""#############################################################################################
#########
Loads rib archive of geometry and assigns surface and displacement shaders and applies a
displacement bound,
Also corrects orientation of the objects that are loaded in.
#############################################################################################
#######"""
def Dandilion(ri,frame,point_cloud_check,out_filename,rendertype) :
        ri.TransformBegin()
        ri.Scale(-1.0,1.0,1.0)

#############################################################################################
##########
        # dandilion

#############################################################################################
##########
        ri.TransformBegin()
        ri.Translate(0.0,0.0,0.0)
        ri.AttributeBegin()
        ran_displace=random.uniform(0.03,0.08)
        ran_diffuse=random.uniform(0.7,0.9)
        ran_diffuse_front=[random.uniform(1.0,1.0),random.uniform(1.0,1.0),random.uniform(1.0,1.0)]
        ran_diffuse_back=[random.uniform(1.0,1.0),random.uniform(0.9,1.0),random.uniform(1.0,1.0)]
        ran_specular=random.uniform(0.2,0.3)
        ran_spec_colour=[random.uniform(0.2,0.3),random.uniform(0.2,0.3),random.uniform(0.4,0.5)]
        if (rendertype == 1) :
                #ri.Surface("./shaders/occlusion/occlusion_v1_r1",{ "float  samples": [32]})
                ri.Surface("matte")
        if (rendertype == 2) :
                ri.Surface("./shaders/new_grass/new_grass_surface_shader_v3_r1",{"float
Ks":ran_specular, "color specularcolor":ran_spec_colour, "float Kd":ran_diffuse,"float Ka":
[1.0],"color front_colour":ran_diffuse_front,"color back_colour":ran_diffuse_back,"string
texture_front":["./sourceimages/leaves/leaves_high_res_v3.tx"],"string  texture_back":
["./sourceimages/leaves/leaves_high_res_v3.tx"],"color opacity":[1.0,1.0,1.0],"float roughness":
[0.5], "float max_size":[5],"string  texture_opacity":
["./sourceimages/grass/grass_opacity_high_v1.tx"],"float Kenv":[0.4],"string EnvironmentMap":
[""],"float oren_roughness":[5.0]})
        ri.Displacement("./shaders/ground/ground_displacement_v2_r1",{ "float Km":[ran_displace],
"float Freq":[0.5], "float Amplitude":[2.5], "float Layers":[6]})
        ri.Attribute("displacementbound",{ri.COORDINATESYSTEM:["object"],"uniform float sphere":
[0.02]})
        ri.ReadArchive("./Archive/forground/dandilion/OCstem_v1_.%04d.rib" %(frame))
        ri.AttributeEnd()
        ri.TransformBegin()
        ri.Rotate(3,1,0,0)
```

```
        ri.Rotate(3,0,1,0)
        ri.AttributeBegin()
        ran_displace=random.uniform(0.03,0.08)
        ran_diffuse=random.uniform(0.7,0.9)
        ran_diffuse_front=[random.uniform(1.0,1.0),random.uniform(1.0,1.0),random.uniform(1.0,1.0)]
        ran_diffuse_back=[random.uniform(1.0,1.0),random.uniform(0.9,1.0),random.uniform(1.0,1.0)]
        ran_specular=random.uniform(0.2,0.3)
        ran_spec_colour=[random.uniform(0.2,0.3),random.uniform(0.2,0.3),random.uniform(0.4,0.5)]
        if (rendertype == 1) :
                #ri.Surface("./shaders/occlusion/occlusion_v1_r1",{ "float  samples": [32]})
                ri.Surface("matte")
        if (rendertype == 2) :
                ri.Surface("./shaders/new_grass/new_grass_surface_shader_v3_r1",{"float
Ks":ran_specular, "color specularcolor":ran_spec_colour, "float Kd":ran_diffuse,"float Ka":
[1.0],"color front_colour":ran_diffuse_front,"color back_colour":ran_diffuse_back,"string
texture_front":["./sourceimages/leaves/leaves_high_res_v3.tx"],"string  texture_back":
["./sourceimages/leaves/leaves_high_res_v3.tx"],"color opacity":[1.0,1.0,1.0],"float roughness":
[0.5], "float max_size":[5],"string  texture_opacity":
["./sourceimages/grass/grass_opacity_high_v1.tx"],"float Kenv":[0.4],"string EnvironmentMap":
[""],"float oren_roughness":[5.0]})
        ri.Displacement("./shaders/ground/ground_displacement_v2_r1",{ "float Km":[ran_displace],
"float Freq":[0.5], "float Amplitude":[2.5], "float Layers":[6]})
        ri.Attribute("displacementbound",{ri.COORDINATESYSTEM:["object"],"uniform float sphere":
[0.02]})
        ri.ReadArchive("./Archive/forground/dandilion/OCbase_v1.rib")
        ri.AttributeEnd()
        ri.TransformEnd()
        ri.AttributeBegin()
        ran_displace=random.uniform(0.03,0.08)
        ran_diffuse=random.uniform(0.6,0.65)
        ran_diffuse_front=[random.uniform(1.0,1.0),random.uniform(0.9,0.9),random.uniform(1.0,1.0)]
        ran_diffuse_back=[random.uniform(1.0,1.0),random.uniform(0.9,0.9),random.uniform(1.0,1.0)]
        ran_specular=random.uniform(0.0,0.02)
        ran_spec_colour=[random.uniform(0.2,0.3),random.uniform(0.2,0.3),random.uniform(0.3,0.4)]
        if (rendertype == 1) :
                #ri.Surface("./shaders/occlusion/occlusion_v1_r1",{ "float  samples": [32]})
                ri.Surface("matte")
        if (rendertype == 2) :
                ri.Surface("./shaders/new_grass/new_grass_surface_shader_v3_r1",{"float Ks":[0.5],
"color specularcolor":[0.2,0.2,0.2], "float Kd":ran_diffuse,"float Ka":[1.0],"color
front_colour":ran_diffuse_front,"color back_colour":ran_diffuse_back,"string  texture_front":
["./sourceimages/leaves/petal_high_res_v1.tx"],"string  texture_back":
["./sourceimages/leaves/petal_high_res_v1.tx"],"color opacity":[1.0,1.0,1.0],"float roughness":
[0.2], "float max_size":[5],"string  texture_opacity":[""],"float Kenv":[0.2],"string
EnvironmentMap":["./sourceimages/hdr/Meadow_non_float.tx"],"float oren_roughness":[5.0]})
        ri.Displacement("./shaders/ground/ground_displacement_v2_r1",{ "float Km":[ran_displace],
"float Freq":[0.5], "float Amplitude":[2.5], "float Layers":[6]})
        ri.Attribute("displacementbound",{ri.COORDINATESYSTEM:["object"],"uniform float sphere":
[0.02]})
        ri.ReadArchive("./Archive/forground/dandilion/OCflower_v1_.%04d.rib" %(frame))
        ri.AttributeEnd()
        ri.TransformEnd()

        ri.TransformBegin()
        ri.Translate(-14.0,-0.2,0.0)
        ri.AttributeBegin()
        ran_displace=random.uniform(0.03,0.08)
        ran_diffuse=random.uniform(0.7,0.9)
        ran_diffuse_front=[random.uniform(1.0,1.0),random.uniform(1.0,1.0),random.uniform(1.0,1.0)]
        ran_diffuse_back=[random.uniform(1.0,1.0),random.uniform(0.9,1.0),random.uniform(1.0,1.0)]
        ran_specular=random.uniform(0.2,0.3)
        ran_spec_colour=[random.uniform(0.2,0.3),random.uniform(0.2,0.3),random.uniform(0.4,0.5)]
        if (rendertype == 1) :
                #ri.Surface("./shaders/occlusion/occlusion_v1_r1",{ "float  samples": [32]})
                ri.Surface("matte")
        if (rendertype == 2) :
                ri.Surface("./shaders/new_grass/new_grass_surface_shader_v3_r1",{"float
Ks":ran_specular, "color specularcolor":ran_spec_colour, "float Kd":ran_diffuse,"float Ka":
[1.0],"color front_colour":ran_diffuse_front,"color back_colour":ran_diffuse_back,"string
texture_front":["./sourceimages/leaves/leaves_high_res_v3.tx"],"string  texture_back":
["./sourceimages/leaves/leaves_high_res_v3.tx"],"color opacity":[1.0,1.0,1.0],"float roughness":
[0.5], "float max_size":[5],"string  texture_opacity":
["./sourceimages/grass/grass_opacity_high_v1.tx"],"float Kenv":[0.4],"string EnvironmentMap":
[""],"float oren_roughness":[5.0]})
        ri.Displacement("./shaders/ground/ground_displacement_v2_r1",{ "float Km":[ran_displace],
"float Freq":[0.5], "float Amplitude":[2.5], "float Layers":[6]})
        ri.Attribute("displacementbound",{ri.COORDINATESYSTEM:["object"],"uniform float sphere":
```

```
[0.02]})
        ri.ReadArchive("./Archive/forground/dandilion/OCbase_v1.rib")
        ri.AttributeEnd()

        ri.TransformEnd()

        ri.TransformEnd()

#Function to control small objects found on the ground in the animation such as small stones and
sticks
"""#############################################################################################
#########
Loads rib archive of geometry and assigns surface and displacement shaders and applies a
displacement bound,
Also corrects orientation of the objects that are loaded in.
#############################################################################################
#######"""
def Small_objects(ri,frame,point_cloud_check,out_filename,rendertype) :
        ri.TransformBegin()
        ri.Scale(-1.0,1.0,1.0)
        ri.TransformBegin()
        ri.Translate(0.0,0.0,0.0)

#############################################################################################
##########
        # Small stones

#############################################################################################
##########
        ri.TransformBegin()
        for i in range(1, 6):
                ri.AttributeBegin()
                ran_displace=random.uniform(0.005,0.001)
                ran_vien=random.uniform(4,20)
                ran_spot_colour_multi=random.uniform(0.5,0.9)


ran_spot_colour=[random.uniform(ran_spot_colour_multi,ran_spot_colour_multi+0.05),random.uniform(ran
_spot_colour_multi,ran_spot_colour_multi+0.05),random.uniform(ran_spot_colour_multi,ran_spot_colour_
multi+0.05)]
                ran_diffuse=[random.uniform(0.5,0.6),random.uniform(0.5,0.6),random.uniform(0.6,0.6)]
                ran_spec=random.uniform(0.1,0.2)
                ran_spot_layers=random.uniform(1,2)
                if (rendertype == 1) :
                        #ri.Surface("./shaders/occlusion/occlusion_v1_r1",{ "float  samples": [32]})
                        ri.Surface("matte")
                if (rendertype == 2) :
                        ri.Surface("./shaders/rocks/rock_surface_v2_r4",{"float Ka":[0.5],"float Kd":
[0.6],"float roughness":[10.0],"float spec_roughness":[2.2],"float Ks":[0.3],"float veining":
ran_vien,"color diffusecolor":ran_diffuse,"color color_multi_one":[0.922,0.922,0.922],"color
color_multi_two":[1.0,1.0,1.0],"color specularcolor":[ran_spec,ran_spec,ran_spec],"float Threshold":
[0.7],"float Layers_of_spots":ran_spot_layers,"float specksize":[0.01],"float sizes":ran_vien,"color
spattercolor":ran_spot_colour,"float Kenv":[0.3],"string  envname":[""]})
                ri.Displacement("./shaders/ground/ground_displacement_v2_r1",{ "float Km":
[ran_displace], "float Freq":[2], "float Amplitude":[30], "float Layers":[8]})
                ri.Attribute("displacementbound",{ri.COORDINATESYSTEM:["object"],"uniform float
sphere":[0.2]})
                ri.ReadArchive("./Archive/forground/small_objects/OCstones_%03d.rib" %(i))
                ri.AttributeEnd()
        ri.TransformEnd()


#############################################################################################
##########
        # Small sticks

#############################################################################################
##########
        ri.TransformBegin()
        for i in range(1, 5):
                ri.AttributeBegin()
                ran_displace=random.uniform(0.005,0.001)
                ran_vien=random.uniform(4,20)
                ran_spot_colour_multi=random.uniform(0.4,0.5)


ran_spot_colour=[random.uniform(ran_spot_colour_multi,ran_spot_colour_multi+0.05),random.uniform(ran
```

```
_spot_colour_multi,ran_spot_colour_multi+0.05),random.uniform(ran_spot_colour_multi,ran_spot_colour_
multi+0.05)]
                ran_diffuse=[random.uniform(0.5,0.6),random.uniform(0.5,0.6),random.uniform(0.6,0.6)]
                ran_spec=random.uniform(0.1,0.22)
                ran_spot_layers=random.uniform(1,2)
                if (rendertype == 1) :
                        #ri.Surface("./shaders/occlusion/occlusion_v1_r1",{ "float  samples": [32]})
                        ri.Surface("matte")
                if (rendertype == 2) :
                        ri.Surface("./shaders/new_grass/new_grass_surface_shader_v3_r1",{"float Ks":
[0.2], "float Kd":[0.5],"float Ka":[1.0],"color front_colour":[0.5,0.5,0.5],"color back_colour":
[0.5,0.5,0.5],"string  texture_front":["./sourceimages/fence/fence_v1_r1_.tx"],"string
texture_back":["./sourceimages/fence/fence_v1_r1_.tx"],"color specularcolor":[0.4,0.6,1.0],"float
roughness":[0.3]})
                ri.Displacement("./shaders/ground/ground_displacement_v2_r1",{ "float Km":
[ran_displace], "float Freq":[2], "float Amplitude":[30], "float Layers":[8]})
                ri.Attribute("displacementbound",{ri.COORDINATESYSTEM:["object"],"uniform float
sphere":[0.02]})
                ri.ReadArchive("./Archive/forground/small_objects/OCsticks_%03d.rib" %(i))
                ri.AttributeEnd()
        ri.TransformEnd()



##############################################################################################
##########
        # Plants

##############################################################################################
##########
        ri.TransformBegin()
        for i in range(1, 5):
                ri.AttributeBegin()
                ran_displace=random.uniform(0.03,0.08)
                ran_diffuse=random.uniform(0.6,0.7)

        ran_diffuse_front=[random.uniform(1.0,1.0),random.uniform(1.0,1.0),random.uniform(1.0,1.0)]

        ran_diffuse_back=[random.uniform(1.0,1.0),random.uniform(0.9,1.0),random.uniform(1.0,1.0)]
                ran_specular=random.uniform(0.31,0.41)

        ran_spec_colour=[random.uniform(0.3,0.4),random.uniform(0.3,0.4),random.uniform(0.4,0.5)]
                if (rendertype == 1) :
                        #ri.Surface("./shaders/occlusion/occlusion_v1_r1",{ "float  samples": [32]})
                        ri.Surface("matte")
                if (rendertype == 2) :
                        ri.Surface("./shaders/new_grass/new_grass_surface_shader_v3_r1",{"float
Ks":ran_specular, "color specularcolor":ran_spec_colour, "float Kd":ran_diffuse,"float Ka":
[1.0],"color front_colour":ran_diffuse_front,"color back_colour":ran_diffuse_back,"string
texture_front":["./sourceimages/leaves/leaves_high_res_v3.tx"],"string  texture_back":
["./sourceimages/leaves/leaves_high_res_v3.tx"],"color opacity":[1.0,1.0,1.0],"float roughness":
[0.5], "float max_size":[5],"string  texture_opacity":
["./sourceimages/grass/grass_opacity_high_v3.tx"],"float Kenv":[0.4],"string EnvironmentMap":
[""],"float oren_roughness":[5.0]})
                ri.Displacement("./shaders/ground/ground_displacement_v2_r1",{ "float Km":
[ran_displace], "float Freq":[0.5], "float Amplitude":[2.5], "float Layers":[6]})
                ri.Attribute("displacementbound",{ri.COORDINATESYSTEM:["object"],"uniform float
sphere":[0.02]})
                ri.ReadArchive("./Archive/forground/plants/OCplant_v%03d.rib" %(i))
                ri.AttributeEnd()
        ri.TransformEnd()

        ri.TransformEnd()
        ri.TransformEnd()

#Function to control the fur found on the ground in the animation
"""##############################################################################################
#########
Loads rib archive of geometry and assigns surface and displacement shaders and applies a
displacement bound,
Also corrects orientation of the objects that are loaded in.
##############################################################################################
#######"""
def Fur(ri,frame,point_cloud_check,out_filename,rendertype) :
        ri.TransformBegin()
        ri.Scale(-1.0,1.0,1.0)
```

```
##############################################################################################################
##########
        # Small rocks

##############################################################################################################
##########
        ri.TransformBegin()
        for i in range(1, 16):
                ri.AttributeBegin()
                ran_diffuse=random.uniform(0.7,0.8)


ran_diffuse_front=[random.uniform(0.47,0.53),random.uniform(0.4,0.52),random.uniform(0.4,0.5)]

        ran_diffuse_back=[random.uniform(0.4,0.5),random.uniform(0.5,0.55),random.uniform(0.4,0.5)]
                ran_specular=random.uniform(0.2,0.3)

        ran_spec_colour=[random.uniform(0.4,0.5),random.uniform(0.5,0.6),random.uniform(0.5,0.6)]
                if (rendertype == 1) :
                        #ri.Surface("./shaders/occlusion/occlusion_v1_r1",{ "float  samples": [32]})
                        ri.Surface("matte")
                if (rendertype == 2) :
                        ri.Surface("./shaders/new_grass/new_grass_surface_shader_v3_r1",{"float
Ks":ran_specular, "color specularcolor":ran_spec_colour, "float Kd":ran_diffuse,"float Ka":
[1.0],"color front_colour":ran_diffuse_front,"color back_colour":ran_diffuse_back,"string
texture_front":["./sourceimages/fence/fence_v1_r1_.tx"],"string  texture_back":
["./sourceimages/fence/fence_v1_r1_.tx"],"color opacity":[1.0,1.0,1.0],"float roughness":[0.5],
"float max_size":[5],"string  texture_opacity":[""],"float Kenv":[0.5],"string EnvironmentMap":
[""],"float oren_roughness":[10.0]})
                #ri.Displacement("./shaders/grass/grass_displacement_shader_v1_r3",{ "float
freq_layer_one":[100], "float freq_layer_two":[60], "float depth_one":[0.04], "float depth_two":
[0.06]})
                #ri.Attribute("displacementbound",{ri.COORDINATESYSTEM:["object"],"uniform float
sphere":[0.02]})
                ri.ReadArchive("./Archive/forground/small_objects/OCfur_%03d.rib" %(i))
                ri.AttributeEnd()
        ri.TransformEnd()

        ri.TransformEnd()

##############################################################################################################
##############################
##############################################################################################################
##############################
# Midground functions
##############################################################################################################
##############################
##############################################################################################################
##############################

#Function to control the midground grass
"""#############################################################################################################
#########
Loads rib archive of geometry and assigns surface and displacement shaders and applies a
displacement bound,
Also corrects orientation of the objects that are loaded in.
##############################################################################################################
#######"""
def Grass_midground(ri,frame,point_cloud_check,out_filename,rendertype) :
        random.seed(25)
        ri.TransformBegin()
        ri.Scale(-1.0,1.0,1.0)

        ri.TransformBegin()

##############################################################################################################
##########
        # Grass midground

##############################################################################################################
##########
        ri.TransformBegin()
        ri.AttributeBegin()
        ran_diffuse=random.uniform(0.9,1.1)
        ran_opacity=random.uniform(1,3)
        ran_tex=random.uniform(1,1)
        ran_diffuse_front=[random.uniform(1.0,1.1),random.uniform(0.8,0.9),random.uniform(0.8,0.9)]
```

```
        ran_diffuse_back=[random.uniform(1.0,1.0),random.uniform(0.9,1.1),random.uniform(1.0,1.1)]
        ran_specular=random.uniform(1.2,1.3)
        ran_spec_colour=[random.uniform(0.4,0.5),random.uniform(0.5,0.6),random.uniform(0.5,0.6)]
        if (rendertype == 1) :
                #ri.Surface("./shaders/occlusion/occlusion_v1_r1",{ "float  samples": [32]})
                ri.Surface("matte")
        if (rendertype == 2) :
                ri.Surface("./shaders/new_grass/new_grass_surface_shader_v3_r1",{"float
Ks":ran_specular, "color specularcolor":ran_spec_colour, "float Kd":ran_diffuse,"float Ka":
[1.0],"color front_colour":ran_diffuse_front,"color back_colour":ran_diffuse_back,"string
texture_front":["./sourceimages/grass/grass_front_high_v1.tx"],"string  texture_back":
["./sourceimages/grass/grass_front_high_v%01d.tx" %(ran_tex)],"color opacity":[1.0,1.0,1.0],"float
roughness":[8.5], "float max_size":[5],"string  texture_opacity":
["./sourceimages/grass/grass_opacity_high_v%01d.tx" %(ran_opacity)],"float Kenv":[0.2],"string
EnvironmentMap":["./sourceimages/hdr/Meadow_non_float.tx"],"float oren_roughness":[5.0]})
        ri.ReadArchive("./Archive/grass/Still_midground/OCgrass_midground_v1_r1_sequence_.%04d.rib"
%(1))
        ri.AttributeEnd()
        ri.TransformEnd()

        ri.TransformEnd()

        ri.TransformBegin()
        ri.Translate(0.5,-30.0,30.0)

##########################################################################################################
##########
        # Grass midground

##########################################################################################################
##########
        ri.TransformBegin()
        ri.AttributeBegin()
        ran_diffuse=random.uniform(0.9,1.1)
        ran_opacity=random.uniform(1,3)
        ran_tex=random.uniform(1,1)
        ran_diffuse_front=[random.uniform(1.0,1.1),random.uniform(0.8,0.9),random.uniform(0.8,0.9)]
        ran_diffuse_back=[random.uniform(1.0,1.0),random.uniform(0.9,1.1),random.uniform(1.0,1.1)]
        ran_specular=random.uniform(1.2,1.3)
        ran_spec_colour=[random.uniform(0.4,0.5),random.uniform(0.5,0.6),random.uniform(0.5,0.6)]
        if (rendertype == 1) :
                #ri.Surface("./shaders/occlusion/occlusion_v1_r1",{ "float  samples": [32]})
                ri.Surface("matte")
        if (rendertype == 2) :
                ri.Surface("./shaders/new_grass/new_grass_surface_shader_v3_r1",{"float
Ks":ran_specular, "color specularcolor":ran_spec_colour, "float Kd":ran_diffuse,"float Ka":
[1.0],"color front_colour":ran_diffuse_front,"color back_colour":ran_diffuse_back,"string
texture_front":["./sourceimages/grass/grass_front_high_v1.tx"],"string  texture_back":
["./sourceimages/grass/grass_front_high_v%01d.tx" %(ran_tex)],"color opacity":[1.0,1.0,1.0],"float
roughness":[8.5], "float max_size":[5],"string  texture_opacity":
["./sourceimages/grass/grass_opacity_high_v%01d.tx" %(ran_opacity)],"float Kenv":[0.2],"string
EnvironmentMap":["./sourceimages/hdr/Meadow_non_float.tx"],"float oren_roughness":[5.0]})
        ri.ReadArchive("./Archive/grass/Still_midground/OCgrass_midground_v1_r1_sequence_.%04d.rib"
%(2))
        ri.AttributeEnd()
        ri.TransformEnd()

        ri.TransformEnd()

        ri.TransformEnd()

#Function to control the trolly not used in the final animaiton
"""######################################################################################################
#########
Loads rib archive of geometry and assigns surface and displacement shaders and applies a
displacement bound,
Also corrects orientation of the objects that are loaded in.
######################################################################################################
#######"""
def Trolly(ri,frame,point_cloud_check,out_filename,rendertype) :
        random.seed(25)
        ri.TransformBegin()
        ri.Scale(-1.0,1.0,1.0)

        ri.TransformBegin()
        ri.Translate(50.0,30.0,120.0)
```

```
##########################################################################################################
##########
        # Fence

##########################################################################################################
##########
        ri.TransformBegin()
        ri.AttributeBegin()
        if (rendertype == 1) :
                #ri.Surface("./shaders/occlusion/occlusion_v1_r1",{ "float  samples": [32]})
                ri.Surface("matte")
        if (rendertype == 2) :
                ri.Surface("plastic")
        ri.ReadArchive("./Archive/environment/OCtrolly_v1_r1_.rib")
        ri.AttributeEnd()
        ri.TransformEnd()

        ri.TransformEnd()


        ri.TransformEnd()

##########################################################################################################
############################
##########################################################################################################
############################
# Background functions
##########################################################################################################
############################
##########################################################################################################
############################

#Function to control the background grass
"""######################################################################################################
#########
Loads rib archive of geometry and assigns surface and displacement shaders and applies a
displacement bound,
Also corrects orientation of the objects that are loaded in.
######################################################################################################
#######"""
def Grass_background(ri,frame,point_cloud_check,out_filename,rendertype) :
        random.seed(25)
        ri.TransformBegin()
        ri.Scale(-1.0,1.0,1.0)

        ri.TransformBegin()
        ri.Translate(50,10.0,30.0)

##########################################################################################################
##########
        # Grass foreground

##########################################################################################################
##########
        ri.TransformBegin()
        ri.AttributeBegin()
        ran_diffuse=random.uniform(0.9,1.1)
        ran_opacity=random.uniform(1,3)
        ran_tex=random.uniform(1,3)
        ran_diffuse_front=[random.uniform(1.0,1.1),random.uniform(0.8,0.9),random.uniform(0.8,0.9)]
        ran_diffuse_back=[random.uniform(1.0,1.0),random.uniform(0.9,1.1),random.uniform(1.0,1.1)]
        ran_specular=random.uniform(1.2,1.3)
        ran_spec_colour=[random.uniform(0.4,0.5),random.uniform(0.5,0.6),random.uniform(0.5,0.6)]
        if (rendertype == 1) :
                #ri.Surface("./shaders/occlusion/occlusion_v1_r1",{ "float  samples": [32]})
                ri.Surface("matte")
        if (rendertype == 2) :
                ri.Surface("./shaders/new_grass/new_grass_surface_shader_v3_r1",{"float
Ks":ran_specular, "color specularcolor":ran_spec_colour, "float Kd":ran_diffuse,"float Ka":
[1.0],"color front_colour":ran_diffuse_front,"color back_colour":ran_diffuse_back,"string
texture_front":["./sourceimages/grass/grass_front_high_v1.tx"],"string  texture_back":
["./sourceimages/grass/grass_front_high_v%01d.tx" %(ran_tex)],"color opacity":[1.0,1.0,1.0],"float
roughness":[8.5], "float max_size":[5],"string  texture_opacity":
["./sourceimages/grass/grass_opacity_high_v%01d.tx" %(ran_opacity)],"float Kenv":[0.2],"string
EnvironmentMap":["./sourceimages/hdr/Meadow_non_float.tx"],"float oren_roughness":[5.0]})
        ri.ReadArchive("./Archive/grass/Still_background/OCgrass_back_ground_v1_r1_sequence_.
%04d.rib" %(1))
```

```
        ri.AttributeEnd()
        ri.TransformEnd()

        ri.TransformEnd()

        ri.TransformBegin()
        ri.Translate(-50,-10.0,30.0)

##############################################################################################################
##########
        # Grass foreground

##############################################################################################################
##########
        ri.TransformBegin()

        ri.AttributeBegin()
        ran_diffuse=random.uniform(0.9,1.1)
        ran_opacity=random.uniform(1,3)
        ran_tex=random.uniform(1,3)
        ran_diffuse_front=[random.uniform(1.0,1.1),random.uniform(0.8,0.9),random.uniform(0.8,0.9)]
        ran_diffuse_back=[random.uniform(1.0,1.0),random.uniform(0.9,1.1),random.uniform(1.0,1.1)]
        ran_specular=random.uniform(1.2,1.3)
        ran_spec_colour=[random.uniform(0.4,0.5),random.uniform(0.5,0.6),random.uniform(0.5,0.6)]
        if (rendertype == 1) :
                #ri.Surface("./shaders/occlusion/occlusion_v1_r1",{ "float  samples": [32]})
                ri.Surface("matte")
        if (rendertype == 2) :
                ri.Surface("./shaders/new_grass/new_grass_surface_shader_v3_r1",{"float
Ks":ran_specular, "color specularcolor":ran_spec_colour, "float Kd":ran_diffuse,"float Ka":
[1.0],"color front_colour":ran_diffuse_front,"color back_colour":ran_diffuse_back,"string
texture_front":["./sourceimages/grass/grass_front_high_v1.tx"],"string  texture_back":
["./sourceimages/grass/grass_front_high_v%01d.tx" %(ran_tex)],"color opacity":[1.0,1.0,1.0],"float
roughness":[8.5], "float max_size":[5],"string  texture_opacity":
["./sourceimages/grass/grass_opacity_high_v%01d.tx" %(ran_opacity)],"float Kenv":[0.2],"string
EnvironmentMap":["./sourceimages/hdr/Meadow_non_float.tx"],"float oren_roughness":[5.0]})
        ri.ReadArchive("./Archive/grass/Still_background/OCgrass_back_ground_v1_r1_sequence_.
%04d.rib" %(2))
        ri.AttributeEnd()
        ri.TransformEnd()

        ri.TransformEnd()

        ri.TransformEnd()

#Function to control the tree not used in the final animation
"""##############################################################################################
#########
Loads rib archive of geometry and assigns surface and displacement shaders and applies a
displacement bound,
Also corrects orientation of the objects that are loaded in.
##############################################################################################
#######"""
def Tree(ri,frame,point_cloud_check,out_filename,rendertype) :
        random.seed(25)
        ri.TransformBegin()
        ri.Scale(-1.0,1.0,1.0)

        ri.TransformBegin()
        ri.Translate(200.0,0.0,250.0)

##############################################################################################################
##########
        # Tree

##############################################################################################################
##########
        ri.TransformBegin()
        ri.AttributeBegin()
        if (rendertype == 1) :
                #ri.Surface("./shaders/occlusion/occlusion_v1_r1",{ "float  samples": [32]})
                ri.Surface("matte")
        if (rendertype == 2) :
                        ri.Surface("./shaders/house/multi_surface_v1_r1", { "float Ks": [0.8], "float
Kd": [0.95],"float Ka":[1.0], "float roughness":[1.4],
                                                "color diffusecolor": [0.6,0.8,0.7],
                                                "color specularcolor":[0.5,0.7,0.7],
```

```
                                        "string  texture_one":
["./sourceimages/fence/fence_v1_r1_.tx"],
                                        "string  texture_two":
["./sourceimages/fence/fence_v1_r1_.tx"],
                                        "float Kenv": [0.4],
                                        "string EnvironmentMap":
["./sourceimages/environment_maps/map_02.tx"],})
                        #ri.Displacement("./shaders/house/multi_displacement_v1_r1", { "float Km":
[0.5], "string texname": ["./sourceimages/fence/fence_v1_r1_.tx"]})
        ri.ReadArchive("./Archive/environment/trees/leaves_type_01_v1_r1_.%04d.rib" %(1))
        if (rendertype == 1) :
                #ri.Surface("./shaders/occlusion/occlusion_v1_r1",{ "float  samples": [32]})
                ri.Surface("matte")
        if (rendertype == 2) :
                ri.Surface("./shaders/ground/ground_surface_v1_r3",{"color diffusecolor":
[0.01,0.01,0.01], "float Layers_of_spots":[3],"color spattercolor": [0.02,0.01,0.01]})
                ri.Displacement("./shaders/ground/ground_displacement_v2_r1",{ "float Km":[0.5],
"float Freq":[5], "float Amplitude":[10], "float Layers":[4]})
                ri.Attribute("displacementbound",{ri.COORDINATESYSTEM:["object"],"uniform float
sphere":[0.2]})
        ri.ReadArchive("./Archive/environment/trees/trunk_type_01_v1_r1_single_frame.rib")
        ri.AttributeEnd()
        ri.TransformEnd()

        ri.TransformEnd()

        ri.TransformEnd()

#Function to control the ivy on the fence
"""#############################################################################################
#########
Loads rib archive of geometry and assigns surface and displacement shaders and applies a
displacement bound,
Also corrects orientation of the objects that are loaded in.
#############################################################################################
#######"""
def Ivy(ri,frame,point_cloud_check,out_filename,rendertype) :
        random.seed(25)
        ri.TransformBegin()
        ri.Scale(-1.0,1.0,1.0)
        ri.TransformBegin()
        ri.Translate(-50.0,0.0,80.0)
        #ri.Rotate(5,1,0,0)

#############################################################################################
#########
        # Fence

#############################################################################################
#########
        ri.TransformBegin()
        for i in range(1, 5):
                ri.AttributeBegin()
                if (rendertype == 1) :
                        #ri.Surface("./shaders/occlusion/occlusion_v1_r1",{ "float  samples": [32]})
                        ri.Surface("matte")
                if (rendertype == 2) :
                        ri.Surface("./shaders/new_grass/new_grass_surface_shader_v3_r1",{"float Ks":
[0.5], "float Kd":[1.0],"float Ka":[1.0],"color front_colour":[0.4,0.55,0.5],"color back_colour":
[0.3,0.5,0.8],"string  texture_front":["./sourceimages/grass/Grass_front.tx"],"string
texture_back":["./sourceimages/grass/Grass_back.tx"],"color specularcolor":[0.4,0.6,1.0],"float
roughness":[0.3]})
                #ri.Displacement("./shaders/grass/grass_displacement_shader_v1_r3",{ "float
freq_layer_one":[150], "float freq_layer_two":[20], "float depth_one":[0.03], "float depth_two":
[0.05]})
                #ri.Attribute("displacementbound",{ri.COORDINATESYSTEM:["object"],"uniform float
sphere":[0.02]})
                ri.ReadArchive("./Archive/Ivy/OCIvy_sticks_v%02d.rib" %(i))
                ri.AttributeEnd()
        ri.TransformEnd()

        ri.TransformBegin()
        for i in range(1, 5):
                ri.AttributeBegin()
                if (rendertype == 1) :
                        #ri.Surface("./shaders/occlusion/occlusion_v1_r1",{ "float  samples": [32]})
                        ri.Surface("matte")
```

```
                if (rendertype == 2) :
                        ri.Surface("./shaders/new_grass/new_grass_surface_shader_v3_r1",{"float Ks":
[0.5], "float Kd":[1.0],"float Ka":[1.0],"color front_colour":[0.4,0.55,0.5],"color back_colour":
[0.3,0.5,0.8],"string  texture_front":["./sourceimages/grass/Grass_front.tx"],"string
texture_back":["./sourceimages/grass/Grass_back.tx"],"color specularcolor":[0.4,0.6,1.0],"float
roughness":[0.3]})
                #ri.Displacement("./shaders/grass/grass_displacement_shader_v1_r3",{ "float
freq_layer_one":[150], "float freq_layer_two":[20], "float depth_one":[0.03], "float depth_two":
[0.05]})
                #ri.Attribute("displacementbound",{ri.COORDINATESYSTEM:["object"],"uniform float
sphere":[0.02]})
                ri.ReadArchive("./Archive/Ivy/OCIvy_leaves_v%02d.rib" %(i))
                ri.AttributeEnd()
        ri.TransformEnd()


        ri.TransformEnd()


        ri.TransformEnd()

#Function to control the fence in the background
"""#############################################################################################
#########
Loads rib archive of geometry and assigns surface and displacement shaders and applies a
displacement bound,
Also corrects orientation of the objects that are loaded in.
#############################################################################################
#######"""
def Fence(ri,frame,point_cloud_check,out_filename,rendertype) :
        random.seed(25)
        ri.TransformBegin()
        ri.Scale(-1.0,1.0,1.0)
        ri.TransformBegin()
        ri.Translate(-50.0,0.0,80.0)
        #ri.Rotate(5,1,0,0)

#############################################################################################
##########
        # Fence

#############################################################################################
##########
        ri.TransformBegin()
        for i in range(1, 101):
                ri.AttributeBegin()
                ran_displace=random.uniform(1.5,2.0)

        ran_diffuse=[random.uniform(0.9,1.1),random.uniform(1.05,1.1),random.uniform(1.05,1.1)]
                ran_spec=random.uniform(0.5,0.8)
                if (rendertype == 1) :
                        #ri.Surface("./shaders/occlusion/occlusion_v1_r1",{ "float  samples": [32]})
                        ri.Surface("matte")
                if (rendertype == 2) :
                        #ri.Surface("wood",{"Ks":[0.1],"point c0":c0,"point c1":c1,"float
grain":random.randint(2,20)})
                        ri.Surface("./shaders/house/multi_surface_v1_r1", { "float Ks": [1.0], "float
Kd": [1.0],"float Ka":[1.0], "float roughness":[2.4],
                                                "color diffusecolor": ran_diffuse,
                                                "color specularcolor":[ran_spec,ran_spec,ran_spec],
                                                "string  texture_one":
["./sourceimages/fence/fence_v1_r1_.tx"],
                                                "string  texture_two":
["./sourceimages/fence/fence_v1_r1_.tx"],
                                                "float Kenv": [0.4],
                                                "string EnvironmentMap":
["./sourceimages/environment_maps/map_02.tx"],})
                ri.Displacement("./shaders/house/multi_displacement_v1_r1", { "float Km":
ran_displace, "string texname": ["./sourceimages/fence/fence_v1_r1_.tx"]})
                ri.Attribute("displacementbound",{ri.COORDINATESYSTEM:["object"],"uniform float
sphere":[0.05]})
                ri.Attribute ("identifier",{"name": "grass_background"})
                if ( i < 87 ):
                        ri.ReadArchive("./Archive/fence/OCfence_v1_r1_%03d.rib" %(i))
                if ( i > 88 ):
                        ri.ReadArchive("./Archive/fence/OCfence_v1_r1_%03d.rib" %(i))
                ri.AttributeEnd()
```

```
        ri.TransformEnd()

        ri.TransformEnd()


        ri.TransformEnd()

#Function to control the background house and all its componets bricks, tiles etc
"""#############################################################################################
#########
Loads rib archive of geometry and assigns surface and displacement shaders and applies a
displacement bound,
Also corrects orientation of the objects that are loaded in.
#############################################################################################
#######"""
def House(ri,frame,point_cloud_check,out_filename,rendertype) :
        ri.TransformBegin()
        ri.Scale(-1.0,1.0,1.0)

        ri.TransformBegin()
        ri.Translate(290.0,-225.0,1950.0)
        #ri.Scale(2.5,2.5,2.5)

#############################################################################################
##########
        # House base

#############################################################################################
##########
        ri.TransformBegin()
        ri.AttributeBegin()
        ri.Surface("./shaders/house/multi_surface_v1_r1", { "float Ks": [0.9], "float Kd":
[1.0],"float Ka":[1], "float roughness":[2.0],
                            "color diffusecolor": [1.2,1.2,1.2],
                            "color specularcolor":[1,1,1],
                            "string  texture_one": ["./sourceimages/building/house_paint.tx"],
                            "string  texture_two": ["./sourceimages/building/house_paint.tx"],
                            "float Kenv": [0.1],
                            "string EnvironmentMap":
["./sourceimages/environment_maps/map_02.tx"],})
        ri.Attribute ("identifier",{"name": "grass_background"})
        ri.Displacement("./shaders/house/multi_displacement_v1_r1", { "float Km": [1.6], "string
texname": ["./sourceimages/building/house_paint.tx"]})
        ri.Attribute("displacementbound",{ri.COORDINATESYSTEM:["object"],"uniform float sphere":
[0.05]})
        ri.ReadArchive("./Archive/house/OChouse_base_v1_r1_single_frame_.rib")
        ri.AttributeEnd()
        ri.TransformEnd()


#############################################################################################
##########
        # House under roof

#############################################################################################
##########
        ri.TransformBegin()
        ri.AttributeBegin()
        ri.Surface("./shaders/house/multi_surface_v1_r1", { "float Ks": [0.5], "float Kd":
[1.0],"float Ka":[1], "float roughness":[40.0],
                            "color diffusecolor": [0.5,0.5,0.5],
                            "color specularcolor":[1,1,1],
                            "string  texture_one": ["./sourceimages/fence/fence_v1_r1_.tx"],
                            "string  texture_two": ["./sourceimages/fence/fence_v1_r1_.tx"],
                            "float Kenv": [0.4],
                            "string EnvironmentMap":
["./sourceimages/environment_maps/map_02.tx"],})
        ri.Attribute ("identifier",{"name": "grass_background"})
        ri.ReadArchive("./Archive/house/OChouse_beams_under_roof_v1_r1_single_frame_.rib")
        ri.AttributeEnd()
        ri.TransformEnd()


#############################################################################################
##########
        # House breast
```

```
##########################################################################################################
##########
        ri.TransformBegin()
        ri.AttributeBegin()
        ri.Surface("./shaders/house/multi_surface_v1_r1", { "float Ks": [0.5], "float Kd":
[1.0],"float Ka":[1], "float roughness":[10.0],
                                "color diffusecolor": [1,1,1],
                                "color specularcolor":[1,1,1],
                                "string  texture_one": ["./sourceimages/building/house_paint.tx"],
                                "string  texture_two": ["./sourceimages/building/house_paint.tx"],
                                "float Kenv": [0.3],
                                "string EnvironmentMap":
["./sourceimages/environment_maps/map_02.tx"],})
        ri.Attribute ("identifier",{"name": "grass_background"})
        ri.ReadArchive("./Archive/house/OChouse_breast_v1_r1_single_frame_.rib")
        ri.AttributeEnd()
        ri.TransformEnd()


##########################################################################################################
##########
        # House bricks

##########################################################################################################
##########
        ri.TransformBegin()
        ri.AttributeBegin()
        ri.Surface("./shaders/house/multi_surface_v1_r1", { "float Ks": [0.5], "float Kd":
[1.0],"float Ka":[1], "float roughness":[2.2],
                                "color diffusecolor": [1,1,1],
                                "color specularcolor":[1,1,1],
                                "string  texture_one": ["./sourceimages/fence/fence_v1_r1_.tx"],
                                "string  texture_two": ["./sourceimages/fence/fence_v1_r1_.tx"],
                                "float Kenv": [0.3],
                                "string EnvironmentMap":
["./sourceimages/environment_maps/map_02.tx"],})
        ri.Displacement("./shaders/house/multi_displacement_v1_r1", { "float Km": [0.6], "string
texname": ["./sourceimages/environment_maps/map_02.tx"]})
        ri.Attribute("displacementbound",{ri.COORDINATESYSTEM:["object"],"uniform float sphere":
[0.05]})
        ri.ReadArchive("./Archive/house/OChouse_bricks_v1_r1_single_frame_.rib")
        ri.AttributeEnd()
        ri.TransformEnd()


##########################################################################################################
##########
        # House flashing

##########################################################################################################
##########
        ri.TransformBegin()
        ri.AttributeBegin()
        ri.Color([1,1,1])
        ri.Surface("./shaders/house/multi_surface_v1_r1", { "float Ks": [0.5], "float Kd":
[1.0],"float Ka":[1], "float roughness":[140.0],
                                "color diffusecolor": [0.7,0.7,0.7],
                                "color specularcolor":[1,1,1],
                                "string  texture_one": ["./sourceimages/fence/fence_v1_r1_.tx"],
                                "string  texture_two": ["./sourceimages/fence/fence_v1_r1_.tx"],
                                "float Kenv": [0.8],
                                "string EnvironmentMap":
["./sourceimages/environment_maps/map_02.tx"],})
        #ri.Displacement("./shaders/house/multi_displacement_v1_r1", { "float Km": [1.6], "string
texname": ["./sourceimages/building/tiles.tx"]})
        #ri.Attribute("displacementbound",{ri.COORDINATESYSTEM:["object"],"uniform float sphere":
[0.05]})
        ri.ReadArchive("./Archive/house/OChouse_flashing_v1_r1_single_frame_.rib")
        ri.AttributeEnd()
        ri.TransformEnd()


##########################################################################################################
##########
        # House glass

##########################################################################################################
```

```
##########
        ri.TransformBegin()
        ri.AttributeBegin()
        """
        ri.Surface("./shaders/house/multi_surface_v1_r1", { "float Ks": [1.0], "float Kd":
[0.3],"float Ka":[1], "float roughness":[0.5],
                            "color diffusecolor": [1,1,1],
                            "color specularcolor":[1,1,1],
                            "string  texture_one": ["./sourceimages/fence/fence_v1_r1_.tx"],
                            "string  texture_two": ["./sourceimages/fence/fence_v1_r1_.tx"],
                            "float Kenv": [0.9],
                            "string EnvironmentMap":
["./sourceimages/environment_maps/map_02.tx"],})"""
        ri.Surface("glass", { "float Ks": [0.4], "float Kd": [0.1],"float Ka":[0.0], "float Kr":
[0.4],"color specularcolor":[1.0,1.0,1.0], "float roughness":[1.25],"string envname":
["./sourceimages/environment_maps/map_02.tx"]  })
        ri.Attribute ("identifier",{"name": "grass_background"})
        ri.ReadArchive("./Archive/house/OChouse_glass_v1_r1_single_frame_.rib")
        ri.AttributeEnd()
        ri.TransformEnd()


##############################################################################################
##########
        # House tiles

##############################################################################################
##########
        ri.TransformBegin()
        ri.AttributeBegin()
        ri.Surface("./shaders/house/multi_surface_v1_r1", { "float Ks": [0.5], "float Kd":
[0.8],"float Ka":[1], "float roughness":[3.2],
                            "color diffusecolor": [0.4,0.4,0.4],
                            "color specularcolor":[1,1,1],
                            "string  texture_one": ["./sourceimages/fence/fence_v1_r1_.tx"],
                            "string  texture_two": ["./sourceimages/fence/fence_v1_r1_.tx"],
                            "float Kenv": [0.45],
                            "string EnvironmentMap": ["./sourceimages/environment_maps/map_02.tx"]})
        ri.Displacement("./shaders/house/multi_displacement_v1_r1", { "float Km": [0.6], "string
texname": ["./sourceimages/building/tiles.tx"]})
        ri.Attribute("displacementbound",{ri.COORDINATESYSTEM:["object"],"uniform float sphere":
[0.05]})
        ri.ReadArchive("./Archive/house/OChouse_tiles_v1_r1_single_frame_.rib")
        ri.AttributeEnd()
        ri.TransformEnd()


##############################################################################################
##########
        # House top tiles

##############################################################################################
##########
        ri.TransformBegin()
        ri.AttributeBegin()
        ri.Surface("./shaders/house/multi_surface_v1_r1", { "float Ks": [0.7], "float Kd":
[1.0],"float Ka":[1], "float roughness":[10.0],
                            "color diffusecolor": [1,1,1],
                            "color specularcolor":[1,1,1],
                            "string  texture_one": ["./sourceimages/fence/fence_v1_r1_.tx"],
                            "string  texture_two": ["./sourceimages/fence/fence_v1_r1_.tx"],
                            "float Kenv": [0.3],
                            "string EnvironmentMap":
["./sourceimages/environment_maps/map_02.tx"],})
        ri.Attribute ("identifier",{"name": "grass_background"})
        ri.ReadArchive("./Archive/house/OChouse_top_tiles_v1_r1_single_frame_.rib")
        ri.AttributeEnd()
        ri.TransformEnd()


##############################################################################################
##########
        # House window frames

##############################################################################################
##########
        ri.TransformBegin()
```

```
        ri.AttributeBegin()
        ri.Surface("./shaders/house/multi_surface_v1_r1", { "float Ks": [0.6], "float Kd":
[1.0],"float Ka":[1], "float roughness":[10.0],
                                "color diffusecolor": [1,1,1],
                                "color specularcolor":[1,1,1],
                                "string  texture_one": ["./sourceimages/building/window_frames.tx"],
                                "string  texture_two": ["./sourceimages/building/window_frames.tx"],
                                "float Kenv": [0.3],
                                "string EnvironmentMap":
["./sourceimages/environment_maps/map_02.tx"],})
        ri.Attribute ("identifier",{"name": "grass_background"})
        ri.ReadArchive("./Archive/house/OChouse_window_frames_v1_r1_single_frame_.rib")
        ri.AttributeEnd()
        ri.TransformEnd()


##########################################################################################################
##########
        # Gutter

##########################################################################################################
##########
        ri.TransformBegin()
        ri.AttributeBegin()
        ri.Surface("./shaders/house/multi_surface_v1_r1", { "float Ks": [0.9], "float Kd":
[1.0],"float Ka":[1], "float roughness":[2.0],
                                "color diffusecolor": [1.1,1.1,1.1],
                                "color specularcolor":[1,1,1],
                                "string  texture_one": ["./sourceimages/building/window_frames.tx"],
                                "string  texture_two": ["./sourceimages/building/window_frames.tx"],
                                "float Kenv": [0.3],
                                "string EnvironmentMap":
["./sourceimages/environment_maps/map_02.tx"],})
        ri.Displacement("./shaders/house/multi_displacement_v1_r1", { "float Km": [1.6], "string
texname": ["./sourceimages/building/house_paint.tx"]})
        ri.Attribute("displacementbound",{ri.COORDINATESYSTEM:["object"],"uniform float sphere":
[0.05]})
        ri.ReadArchive("./Archive/house/OChouse_gutter_v1_r1_single_frame_.rib")
        ri.AttributeEnd()
        ri.TransformEnd()

        ri.TransformEnd()

        ri.TransformEnd()

##########################################################################################################
##############################
##########################################################################################################
##############################
# Far Background Sky functions
##########################################################################################################
##############################
##########################################################################################################
##############################

#Function to control the sky in shot one, three, four, five, six
"""#########################################################################################################
#########
Loads rib archive of geometry and assigns surface and displacement shaders and applies a
displacement bound,
Also corrects orientation of the objects that are loaded in.
#########################################################################################################
#######"""
def Sky_shot_01(ri,frame,point_cloud_check,out_filename,rendertype) :
        ri.TransformBegin()
        ri.Scale(-1.0,1.0,1.0)

##########################################################################################################
##########
        # Environment Sky shot 01

##########################################################################################################
##########
        ri.TransformBegin()
        ri.Translate(180,400,250)
        ri.AttributeBegin()
        ri.Color([1,1,1])
```

```
        ri.Surface("./shaders/sky/clouds_v1_r2",{"string texture_one":
["./sourceimages/sky/shot_01_sky.tx"]})
        ri.Attribute ("identifier",{"name": "grass_background"})
        ri.ReadArchive("./Archive/environment/OCsky_shot_01_v1_r1_single_frame_.rib")
        ri.AttributeEnd()
        ri.TransformEnd()

        ri.TransformEnd()

#Function to control the sky in shot two
"""#############################################################################################
#########
Loads rib archive of geometry and assigns surface and displacement shaders and applies a
displacement bound,
Also corrects orientation of the objects that are loaded in.
#############################################################################################
#######"""
def Sky_shot_02(ri,frame,point_cloud_check,out_filename,rendertype) :
        ri.TransformBegin()
        ri.Scale(-1.0,1.0,1.0)


#############################################################################################
##########
        # Environment Sky shot 01

#############################################################################################
##########
        ri.TransformBegin()
        ri.Translate(-20,20,20)

        ri.AttributeBegin()
        ri.Color([1,1,1])
        ri.Surface("./shaders/sky/clouds_v1_r2",{"string texture_one":
["./sourceimages/sky/shot_02_sky.tx"]})
        ri.Attribute ("identifier",{"name": "grass_background"})
        ri.ReadArchive("./Archive/environment/OCsky_shot_02_v1_r1_single_frame_.rib")
        ri.AttributeEnd()
        ri.TransformEnd()

        ri.TransformEnd()
```

Shaders

surface rock_surface_v2_r4

```
/*!
Shader was developed to suit the need for the rocks and the ground in the piece the shaders needed
to be able to create a wide to be able to have variation as it shades prominent and different
objects.
A high degree of physical accuracy of the properties of the ground and rocks had to be reproduced or
made to look convincing to the viewer.
*/

/// from RManNotes http://accad.osu.edu/~smay/RManNotes/index.html
/// from http://www.fundza.com/rman_shaders/subsurf_texture/index.html
color envmapping(normal norm;
                 vector incident;
                 string mapname;
                 string coordname;
                 float  kenv)
{
color envcolor = 0;
if(mapname != "")
    {
    normal    n = normalize(norm);
    vector    i = normalize(incident);
    normal    nf = faceforward(n, i);

    vector R = reflect(i, nf);
    R = transform(coordname, R);
    envcolor = environment(mapname, R) * kenv;
    }
```

```
return envcolor;
}

/* Code from Jon Macey
 * Oren and Nayar's generalization of Lambert's reflection model.
 * The roughness parameter gives the standard deviation of angle
 * orientations of the presumed surface grooves.  When roughness=0,
 * the model is identical to Lambertian reflection.
 * Modified from the advanced renderman companion notes
 */
color LocIllumOrenNayar (normal N;  vector V;  float roughness)
{
    /* Surface roughness coefficients for Oren/Nayar's formula */
    float sigma2 = roughness * roughness;
    float A = 1 - 0.5 * sigma2 / (sigma2 + 0.33);
    float B = 0.45 * sigma2 / (sigma2 + 0.09);
    /* Useful precomputed quantities */
    float  theta_r = acos (V . N);        /* Angle between V and N */
    vector V_perp_N = normalize(V-N*(V.N)); /* Part of V perpendicular to N */

    /* Accumulate incoming radiance from lights in C */
    color  C = 0;
    extern point P;
    illuminance (P, N, PI/2)
       {
       /* Must declare extern L & Cl because we're in a function */
              vector LN = normalize(L);
              float cos_theta_i = LN . N;
              float cos_phi_diff = V_perp_N . normalize(LN - N*cos_theta_i);
              float theta_i = acos (cos_theta_i);
              float alpha = max (theta_i, theta_r);
              float beta = min (theta_i, theta_r);
              C += Cl * cos_theta_i *
              (A + B * max(0,cos_phi_diff) * sin(alpha) * tan(beta));
       }
    return C;
}

/// Parameters of shader

surface rock_surface_v2_r4(
float Ka= 0.5;
float Kd= 1.0;
float roughness= 10.0;
float spec_roughness = 2.0;
float Ks= 0.5;
float veining = 40.0;
color diffusecolor = ( 0.7,0.7,0.6);
color color_multi_one = ( 0.922,0.922,0.922);
color color_multi_two = ( 1.0,1.0,1.0);
color specularcolor = ( 0.1,0.1,0.1);
float Threshold = 0.7;
float Layers_of_spots = 1;
float specksize = 0.01;
float sizes = 5;
color spattercolor = color (1,1,1);
float Kenv = 0.3;
string  envname = "";
string envspace = "world";
string  texture_front = "";)
{
/// vector calulation
normal Nf = faceforward(normalize(N),I);
vector V = -normalize(I);
color surface_color, layer_color, surface_opac;
color layer_opac, surface_spec;
color texture_front_colour =1;
/// Load textures if available
// texture data
if(texture_front != "")
{
    texture_front_colour = texture(texture_front);
}

/// pattern setting
surface_color = (float noise(s * 2, t * 2) + color noise(s * 1, t * 1) )* diffusecolor;
```

```
/// for marble viening
/// The RenderMan marble shader was looked into modified and parts were used to create a marble
veining on the rock although subtle effect in the end it does add procedural variation.

point PP;
float width, cutoff, fade, f, turb, maxfreq = 16;

PP = transform("shader", P) * veining;

width = (max(sqrt(area(PP)), 1e-7));
//width = PP/2;
cutoff = clamp(0.5 / width, 0, maxfreq);

turb = 0;
for (f = 1; f < 0.5 * cutoff; f *= 2)
{
  turb += abs((noise(PP * f) * 2 - 1)) / f;
}

fade = clamp(2 * (cutoff - f) / cutoff, 0, 1);
turb += fade * abs((noise(PP * f) * 2 - 1)) / f;

turb *= 0.5;

layer_color = (spline(turb,
                      color (0.9, 0.9, 0.9),
                      color (0.8, 0.8, 0.8),
                      color (0.5, 0.5, 0.5),
                      color (0.4, 0.4, 0.4),
                      color (0.6, 0.6, 0.6),
                      color (0.3, 0.3, 0.3),
                      color (0.2, 0.2, 0.2),
                      color (0.1, 0.1, 0.1)))*color_multi_one;
layer_opac = layer_color;
surface_color = ((surface_color) * (1 - (layer_opac)) + (layer_color) * (layer_opac));

/// spots
/*!
The RenderMan spatter shader and the theory behind it was looked into modified and parts were used
to create a repeating spots of colour to the surface to add procedural variation. Also a
modification offer the specular component of the shader was achieved with his effect to be used as a
multiplier during the specular component calculation stage*/
varying float spot,size,scalefac,threshold = Threshold;
float layer,temp_layer, max_size = sizes;

surface_spec = color (0.0,0.0,0.0);//specularcolor;

for (layer = 1; layer <= Layers_of_spots; layer += 1)
{
/// add variation of colours to the spots
       scalefac = 1/specksize;
       for (size=1; size<=max_size; size +=1)
       {
              spot = noise(transform("shader",P)*scalefac);
              if (spot > threshold)
              {
                    surface_color += ((color (max(0.7,1.0),max(0.7,1.0),max(0.7,1.0))) *
(spattercolor * (layer *0.5)) * 1.0);
                    if ( layer >= 2 )
                    {
                       surface_spec += ((color (max(0.1,0.3),max(0.1,0.3),max(0.1,0.3)))) * float
random();
                    }
                    break;
                    //}
              }
              scalefac /= 2;
       }
max_size /= 2;
threshold *= 0.8;
scalefac /= 2;
}
surface_color *= spattercolor;

/// environment lighting
color  envcolor = color(0.5,0.5,0.5);
if (envname != "") {
```

```
       envcolor= envmapping(N, I, envname, envspace, Kenv);
}

/// Out colour and opacity
surface_color *= texture_front_colour;
Oi=Os;

Ci = ((Kd * (LocIllumOrenNayar(Nf,V,roughness)*surface_color)) + (specularcolor *
specular(Nf,V,spec_roughness) * envcolor * Ks)* Oi);
}
```

displacement ground_displacement_v2_r1

```
/*! ground displacement shader
A simulation of turbulence or fractal noise was used to create the bumpy and protruding surfaces of
rocks and a great amount of detail using displacement can be achieve on relatively low resolution
geometry.
This be achieved by using the noise() within a loop. On each iteration of the loop the value
returned from noise() is added to the result of the previous iteration. Successfully higher
frequencies but smaller amplitudes are used for iteration. The visual result is richer because the
shading can appear to mimic natural surfaces ie. large bumps have small bumps which in turn have
even smaller[http://www.fundza.com/rman_shaders/displacement/index.html]
 inflenced by http://www.fundza.com/rman_shaders/displacement/index.html */
displacement ground_displacement_v2_r1(
float Km = 0.02;
float Freq = 5;
float Amplitude = 30;
float Layers = 8;
string   space = "object";)
{

/// initialise variables
float   proc_displace = 0;
normal  n = normalize(N);
point   p = transform(space, P);
float   j, f = Freq, amplitude = Amplitude;

/// loop that calculates the noise displacement in layers
for(j = 0; j < Layers; j += 1)
{
    proc_displace += (noise(p * f) -0.5) * amplitude;
    f *= 2;
    amplitude *= 0.5;
}

/// calulates the out displacement
P = P - n * proc_displace * Km;
N = calculatenormal(P);
}
```

/// libary of shading models

```
#define SQR(A) ((A)*(A))



// from RManNotes http://accad.osu.edu/~smay/RManNotes/index.html
/// LocIllumOrenNayar(Nf,V,oren_roughness)
/// color environment_refl = color environment(EnvironmentMap,Rworld);
// from http://www.fundza.com/rman_shaders/subsurf_texture/index.html
// for environment lighting
color envmapping(normal norm;
                 vector incident;
                 string mapname;
                 string coordname;
                 float  kenv)
{
color envcolor = 0;
if(mapname != "")
    {
    normal    n = normalize(norm);
    vector    i = normalize(incident);
    normal    nf = faceforward(n, i);

    vector R = reflect(i, nf);
    R = transform(coordname, R);
```

```
    envcolor = environment(mapname, R) * kenv;
    }
return envcolor;
}

/* Code from Jon Macey
 * Oren and Nayar's generalization of Lambert's reflection model.
 * The roughness parameter gives the standard deviation of angle
 * orientations of the presumed surface grooves.  When roughness=0,
 * the model is identical to Lambertian reflection.
 * Modified from the advanced renderman companion notes
 */
color LocIllumOrenNayar (normal N;  vector V;  float roughness)
{
    /* Surface roughness coefficients for Oren/Nayar's formula */
    float sigma2 = roughness * roughness;
    float A = 1 - 0.5 * sigma2 / (sigma2 + 0.33);
    float B = 0.45 * sigma2 / (sigma2 + 0.09);
    /* Useful precomputed quantities */
    float  theta_r = acos (V . N);        /* Angle between V and N */
    vector V_perp_N = normalize(V-N*(V.N)); /* Part of V perpendicular to N */

    /* Accumulate incoming radiance from lights in C */
    color  C = 0;
    extern point P;
    illuminance (P, N, PI/2)
        {
        /* Must declare extern L & Cl because we're in a function */
               vector LN = normalize(L);
               float cos_theta_i = LN . N;
               float cos_phi_diff = V_perp_N . normalize(LN - N*cos_theta_i);
               float theta_i = acos (cos_theta_i);
               float alpha = max (theta_i, theta_r);
               float beta = min (theta_i, theta_r);
               C += Cl * cos_theta_i *
               (A + B * max(0,cos_phi_diff) * sin(alpha) * tan(beta));
        }
    return C;
}

//-------------------------------------------------------------------------------------------------------
----//
//-------------------------------------------------------------------------------------------------------
----//
//     SCRIPT: CookTorrance.sl
//     AUTHOR: Scott Eaton
//     DATE:          July 3, 2007
//
//     DESCRIPTION: A simple implementation of the Cook-Torrance
//  shading model describe in:
//  A Reflectance Model for Computer Graphics
//  R. L. Cook, K. E. Torrance, ACM Transactions on Graphics 1982
//
//-------------------------------------------------------------------------------------------------------
----//
//-------------------------------------------------------------------------------------------------------
----//

/// cook torrance shading model calulations
color cook_torrance (normal Nn;  vector Vn;  float roughness; float gaussConstant; float IOR)
{
        //normal Nn = normalize(N);
         //vector Vn = normalize(-I);
         float F, Ktransmit;
         float m = roughness;
         fresnel( normalize(I), Nn, 1/IOR, F, Ktransmit);

        color cook = 0;
         float NdotV = Nn.Vn;

         illuminance( P, Nn, PI/2 ){
                 //half angle vector
                 vector Ln = normalize(L);
                 vector H = normalize(Vn+Ln);

                 float NdotH = Nn.H;
                 float NdotL = Nn.Ln;
```

```
                float VdotH = Vn.H;

                float D;
                float alpha = acos(NdotH);

                //microfacet distribution
                D = gaussConstant*exp(-(alpha*alpha)/(m*m));

                //geometric attenuation factor
                float G = min(1, min((2*NdotH*NdotV/VdotH), (2*NdotH*NdotL/VdotH)));

                //sum contributions
                cook += Cl*(F*D*G)/(PI*NdotV);
        }
        cook = cook/PI;
return cook;
}

/*Velvet illuminance loop
The Stephen H. Westin velvet
illuminance loop.*/

color velvet (normal Nf;  vector V;  float roughness; color sheen)
{
color shiny = 0;
vector H;
vector ln;
float cosine, sine;

illuminance (P, Nf, 1.57079632679489661923)
{
    ln = normalize(L);
    cosine = max (-Nf.V,0);
    shiny += pow (cosine, 1.0/roughness) / (ln.Nf) * Cl * sheen;
    cosine = max (Nf.V, 0);
    sine = sqrt (1.0-SQR(cosine));
    shiny += pow(sine, 10.0)*ln.Nf * Cl*sheen;
}
return shiny;
}
```

surface new_grass_surface_shader_v3_r1

```
#include "shading_models_v1_r1.sl"
surface new_grass_surface_shader_v3_r1(
/// Contol multiplers
float Ks=1.0, Kd=1.0, Ka=1.0;

/// for opacity calulations
string _OPACITY_ = " ";
float hole_size = 0.01;
float max_size = 10.0;
float control_threshold = 0.8;
float threshold_adjust = -0.05;
color opacity = color (1.0,1.0,1.0);
color hole_opacity = color (0.0,0.0,0.0);
color color_holes_value = color (0.8,0.8,0.8);
string  texture_opacity = "";

/// for diffuse/pattern calulations
float oren_roughness = 5.0;
color   front_colour = color (0.8,0.9,0.7);
color  back_colour  = color (0.7,0.9,0.8);
string  texture_front = "";
string  texture_back = "";

/// for specular
color specularcolor = color(1.0,1.0,1.0);
float roughness = 0.2;

/// for environment reflection calulations
string _ENVIRONMENT_ = "";
float Kenv = 0.3;
string EnvironmentMap = "";
string envspace = "world";
)
{
```

```
/// Defines and sets vector, normal and point values used
normal Nf;
vector V;

Nf = faceforward( normalize(N), I );
V = -normalize(I);

/// OPACITY GENERATION
/*!Influenced by  the RenderMan spatter shader which ships with RenderMan as a standard shader.
Noise is created in shader space using the 3d point position P. Controlling its
influence by a scale factor and checking if the value of the current influence noise on a point is
greater than a threshold if its is  it is applied an opacity value 0.0 else
it is kept its default value of 1.0 opaque. There is also
a similar check with an offset to the threshold which modify the colour value of the point. This
gives the effect of having a hole on the surface with a surrounding colour gradation.*/

color out_opacity;
varying float hole,size,scalefac,threshold = control_threshold;
varying color opacity_holes;
varying color color_holes;
scalefac = 1/hole_size;

// sets main opacity
opacity_holes = color(1.0,1.0,1.0) * opacity;

// color holes
color_holes = color(0.0,0.0,0.0);

// sets holes opacity
for (size=1; size<=max_size; size +=1)
{
      hole = noise(transform("shader",P)*scalefac);
      if (hole > threshold)
      {
       opacity_holes = hole_opacity;
       break;
      }
      if (hole > (threshold+threshold_adjust))
      {
       color_holes = color_holes_value;
       break;
      }
scalefac /= 2;
}

opacity_holes = color(1.0,1.0,1.0) * opacity;
if(texture_opacity != "")
{
    opacity_holes = texture(texture_opacity);
}
// set opacity
out_opacity = opacity_holes;

////////////////////////////////////////////////////////////////////////////////////////////////
////////////////
/// Pattern diffuse calculations
////////////////////////////////////////////////////////////////////////////////////////////////
////////////////
/// Defines colour and texture values used in the diffuse calulations
color front_face_diffuse, back_face_diffuse, out_diffuse = 1;
color texture_front_colour, texture_back_colour = 1;

/// Load textures if available
// texture data
if(texture_front != "")
{
    texture_front_colour = texture(texture_front);
}

if(texture_back != "")
{
    texture_back_colour = texture(texture_back);
}

/// Calulates the diffuse for both sides
//front_face_diffuse = diffuse(Nf) * ((texture_front_colour += color_holes)) * front_colour;
front_face_diffuse = LocIllumOrenNayar(Nf,V,oren_roughness) * texture_front_colour * front_colour;
```

```
//front_face_diffuse = diffuse(Nf) * texture_front_colour * front_colour;

//back_face_diffuse = diffuse(-Nf) * ((texture_back_colour += color_holes)) * back_colour;
back_face_diffuse = LocIllumOrenNayar(-Nf,V,oren_roughness) * texture_back_colour * back_colour;
//back_face_diffuse = diffuse(-Nf) * texture_back_colour * back_colour;
/// MODIFY ALTERNATE SIDE
/// ////////////////////////////////////////////////////////////////////////////////////////////
// front side
if (Nf.N>0)
{
        // Final colour applied
        //out_diffuse = mix(front_face_diffuse,back_face_diffuse,0.7);
        out_diffuse = front_face_diffuse + (back_face_diffuse * 0.2);
}else
// back side
{
         // Final colour applied
        out_diffuse = (front_face_diffuse * 0.9) + (back_face_diffuse * 0.3);
        //out_diffuse = mix(back_face_diffuse,front_face_diffuse,0.7);
        //out_diffuse = back_face_diffuse;// + (front_face_diffuse * 0.5);
}

////////////////////////////////////////////////////////////////////////////////////////////////
////////////////
/// Environement map
////////////////////////////////////////////////////////////////////////////////////////////////
////////////////
color out_environment = color (1.0,1.0,1.0);
if(EnvironmentMap != "")
{
       out_environment = envmapping(N, I, EnvironmentMap, envspace, Kenv);
}

////////////////////////////////////////////////////////////////////////////////////////////////
////////////////
/// Out colour and opacity
////////////////////////////////////////////////////////////////////////////////////////////////
////////////////
 Oi = out_opacity * opacity;//out_opacity;
 //Oi = opacity;
 //Ci = front_face_diffuse;
 //Ci = texture_front_colour * Oi;
 Ci = Oi * (((Kd* out_diffuse) + (Ka*ambient())) +(specularcolor * out_environment * Ks *
specular(Nf,V,roughness)));
}
```

surface multi_surface_v1_r1

```
#include "shading_models_v1_r1.sl"

surface multi_surface_v1_r1( float Ks=.5, Kd=.5, Ka=1, roughness=.1;
                             color diffusecolor=1;
                             color specularcolor=1;
                             color opacitycolor=1;
                             string  texture_one = "";
                             string  texture_two = "";
                             string  texture_opacity = "";
                             float Kenv = 0.3;
                             string EnvironmentMap = "";
                             string envspace = "world";)
{
normal Nf;
vector V;

Nf = faceforward( normalize(N), I );
V = -normalize(I);

color texture_one_colour, texture_opacity_colour, texture_two_colour, out_environment =
color(1.0,1.0,1.0);

// texture data
if(texture_one != "")
{
    texture_one_colour = texture(texture_one);
}

if(texture_two != "")
```

```
{
    texture_two_colour = texture(texture_one);
}

color out_diffuse = ((texture_one_colour * texture_two_colour) * diffusecolor);

/// Opacity

if(texture_opacity != "")
{
    texture_opacity_colour = texture(texture_opacity);
}

/// ENVIRONMENT LIGHTING

if(EnvironmentMap != "")
{
    out_environment = envmapping(N, I, EnvironmentMap, envspace, Kenv);
}

Oi = 1;//texture_opacity_colour * opacitycolor;
Ci = ( ((Ka*ambient() + Kd*diffuse(Nf) * out_diffuse)) +
              ((specularcolor * out_environment) * Ks * specular(Nf,V,roughness)) ) * Oi;
}
```