Ville Saario

Visual Effects in SideFX Houdini

Bachelor of Business Administration

Business Information Technology

Spring 2019

KAMK • University of Applied Sciences

Tiivistelmä

Tekijä(t): Saario Ville

Työn nimi: Visuaaliset tehosteet SideFX Houdini -ohjelmassa

Tutkintonimike: tradenomi, tietojenkäsittelyn koulutusohjelma

Asiasanat: Houdini, visuaaliset tehosteet, vfx, simulointi

Opinnäytetyössä tutustutaan Houdini 3D-ohjelmaan ja tuotetaan lyhyt animaatio käyttäen sen perusominaisuuksia. Ohjelma ei ollut aikaisemmin tuttu joten työhön kuului ohjelman opettelu. Aluksi työssä käydään lyhyesti läpi mitä tarkoitetaan visuaalisilla tehosteilla. Tämän jälkeen tutustutaan päällisin puolin ohjelmistoon ja syihin miksi juuri se valittiin opeteltavaksi. Lopuksi tuotettu animaatio puretaan vaihe kerrallaan auki. Animaation sisältö valittiin, jotta samalla kertaa voitaisin käyttää useita ohjelman eri ominaisuuksia. Animaatio koostuu neljästä pääosasta: neste-, fysiikka-, partikkeli-, ja savusimulaatiosta.

Nestesimulaatiota varten luodaan lähteet nesteelle ja mallinnetaan kristalli sekä animoidut viivat joita pitkin neste ohjataan. Fysiikkasimulaatiossa kristalli murretaan erikokoisiksi palasiksi kahta eri tapaa käyttäen. Kristallinsirpaleille määritellään nopeus, jotta kristalli räjähtää kappaleiksi. Sirpaleista synnytetään partikkeleita, joita simulaatiossa liikutetaan erilaisin keinoin. Savusimulaatiossa jäljellä olevat isommat kristallin palaset vähitellen katoavat erittäen savua. Työ renderöidään PNG-kuvatiedostoiksi käyttäen Houdinin omaa Mantra-renderöijää ja muutetaan videoksi käyttäen ilmaista Blender-ohjelmaa.

Abstract

Author(s): Saario Ville

Title of the Publication: Visual Effects in SideFX Houdini

Degree Title: Bachelor of Business Administration, Business Information Technology

Keywords: Houdini, visual effects, vfx, simulation

The objective of this Bachelor's thesis was to learn the 3D software Houdini and produce a short animation using its basic functions. The program was not previously familiar. First is a small introduction to the term Visual Effects. This is followed by an overview of Houdini itself and the reasons for choosing this program for the thesis. Finally, the animation project is broken down part by part. The contents of the animation were chosen so a wide variety of the program's features could be used. The animation consists of four main parts: fluid, physics, particle, and smoke simulations.

For the fluid simulation, sources were created for emitting the fluid. A crystal was also modeled as well as animated lines for guiding the fluid. In the physics simulation, the crystal was fractured into different sized shards using two different methods. The shards were given velocity so the crystal explodes apart. Particles were birthed from the smaller shards that were then given additional motion in the simulation. In the smoke simulation, the remaining large shards were eroded away while generating smoke. The project was rendered into PNG-files using Mantra, Houdini's own renderer and edited into a video file with the free program Blender.

Table of contents

1 Introduction 1

2 What are Visual Effects? 2

3 What is Houdini? 3

3.1 Why Houdini? 3

4 Learning and resources 5

5 Project Breakdown 6

5.1 Foundation 6

5.2 Constructing the crystal 8

5.3 Fluid simulation 9

5.3.1 Sources 9

5.3.2 Target 9

5.3.3 Simulation 10

5.3.4 Caching and surfacing 13

5.3.5 Exploding liquid simulation 13

5.4 Breaking geometry for physics simulation 14

5.4.1 Using Boolean shatter 16

5.4.2 Using Voronoi fracturing 16

5.5 Crystal shatter simulation 17

5.6 Disintegration 18

5.6.1 Small shards 18

5.6.2 Big shards 20

5.7 Particles 20

5.7.1 Wedges 22

5.8 Pyro smoke simulation 23

5.9 Building the final scene 24

5.10 Rendering and lighting 25

6 Closing thoughts 26

Sources 27

Terminology

Voxel: A portmanteau of the words volume and pixel. A value in a three-dimensional grid.

VDB: An abbreviation of “Volumetric, Dynamic grid that shares several characteristics with B+trees.” Simply put, a way to represent volumetric data.

SOP/DOP/POP/ROP/CHOP: Various names for different operators in Houdini depending on their context. SOP stands for Surface Operator, DOP for Dynamics Operator, POP for Particle Operator, ROP for Render Operator, and CHOP for Channel Operator. There are others, but they are not used in this thesis.

VEX: A coding/scripting language inside Houdini.

Keyframe: A keyframe is a frame in an animation that defines the start or end of a transition.

Substep: In simulations a single frame can be further broken down into substeps to increase the accuracy of the simulation at the cost of increased simulation time.

Normal: A vector on the surface of 3D geometry that determines how light behaves on the surface.

1 Introduction

Ever since first seeing computer generated cutscenes in games in the 1990s, I've been fascinated with computer graphics. Not limited to the physical realm, anything one can imagine can be realized.

The aim of this thesis is to learn the basics of the program Houdini and produce a simple animation. The idea was to do a sequence of effects that would use a wide variety of the program's features. The final result evolved over time from the initial concept. There was no previous experience with Houdini before settling on learning it for the thesis. As such, it was important to keep the scope of the project in mind and avoid "feature creep" as imagination tended to far outpace competence. Sometimes a seemingly simple idea turned out to be more complex than anticipated and had to be discarded.

2 What are Visual Effects?

Visual effects, commonly abbreviated as VFX, is an umbrella term for enhancing the imagery of a film or television series with the aid of computers. While special effects are physical and happen in the real world, visual effects are added afterwards digitally. Everything from adding or removing background objects and scenery with chroma keying and matte paintings, performance capturing real people, to creating entirely artificial characters in the computer can be regarded as being visual effects. [1.]

In the early years of computer graphics, only those with access to monstrously expensive hardware, usually film studios and the like, would have enough computing power to produce them. With abundant processing power now available even in consumer level computers, combined with freely available learning material thanks to the Internet, anyone with an interest can begin learning. The ease of getting into computer graphics is further helped by non-commercial versions of professional software and even completely free programs.

3 What is Houdini?

Houdini is a 3D graphics software package developed by the Toronto based Side Effects Software. Version 1.0 of Houdini was released in 1996, but it has been in development well before that [2]. The versions used in this thesis were 16.5 and 17.

The main workflow in Houdini is based around nodes and connecting them together (figure 1). There are simply too many nodes to reasonably list in a short introduction. The node-based approach makes the workflow non-destructive, meaning at any point one can go back to an earlier node to change its settings and have the changes propagate through the network.

Houdini has primarily been used in films and TV. Movies with effects from Houdini are too numerous to list, but some examples are movies in Marvel's Cinematic Universe and Disney films such as Frozen and Moana [3].

In addition to movies, Houdini is seeing increasing use in the games industry. Recently released titles that have used Houdini in some part of their production include Anthem from BioWare, Far Cry 5 from Ubisoft, and Marvel's Spider-Man from Insomniac Games [4]. The game development toolset inside Houdini has special nodes designed for use in that field.

3.1 Why Houdini?

There were two primary reasons why Houdini was chosen as the program to learn. Firstly, it incorporated a lot of features in one program that elsewhere were their own separate pieces of software. Features like particles, physics, fluid, and fire simulations.

Secondly and very simply, the learning edition of Houdini was free. The Apprentice Edition is a fully featured, non-commercial version of Houdini with some limitations: The only available renderer is Houdini's own Mantra, the renders are limited to 1280x720 and are watermarked. [5.]

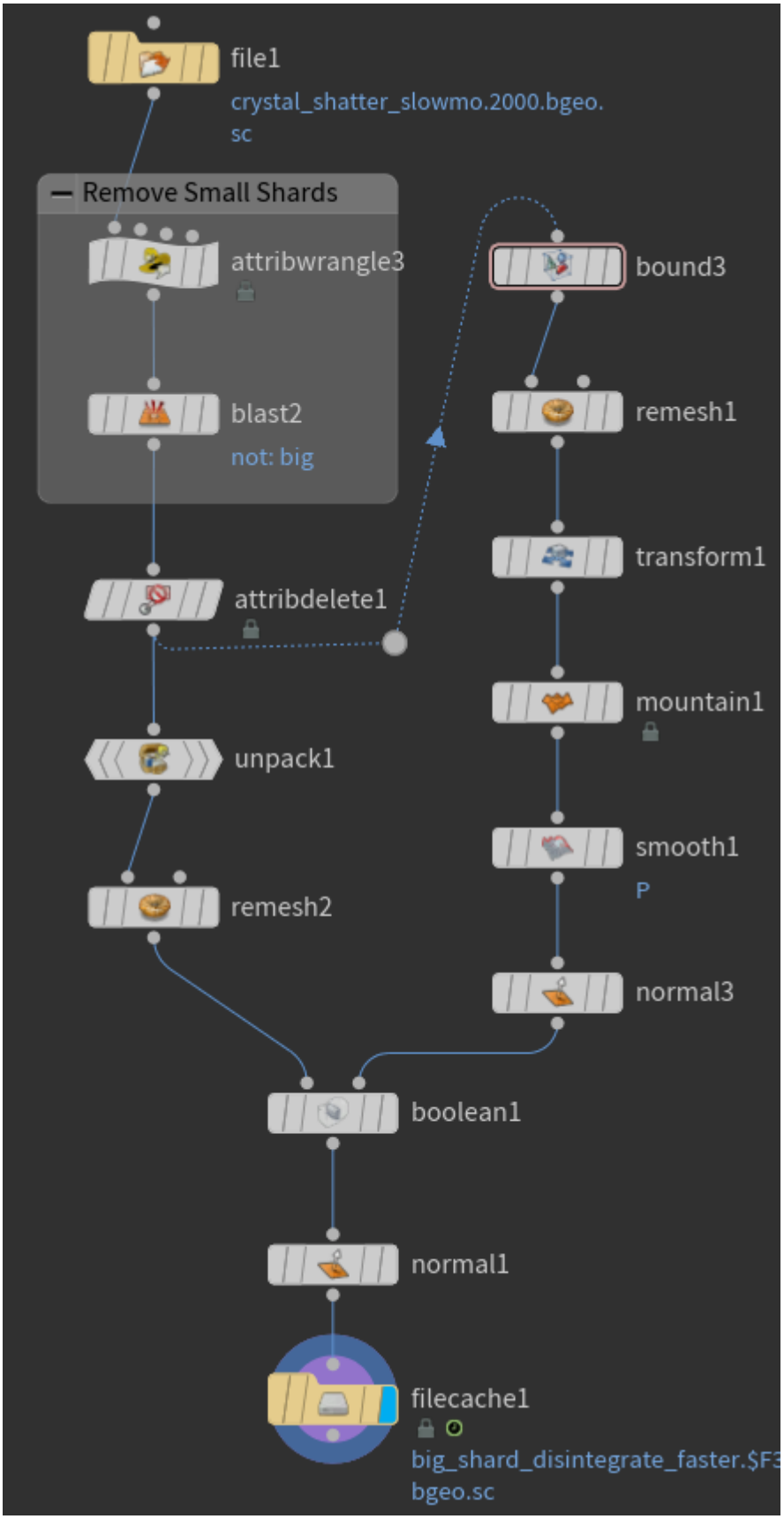

Figure 1 An example of a network in Houdini

4 Learning and resources

The learning curve for Houdini felt rather steep. Coming from programs like 3ds Max, Houdini's node-based workflow was very different. The most commonly used nodes quickly became familiar though and the upsides of using nodes to work became apparent, even if the full list of all available nodes was long and somewhat intimidating.

Attributes are an absolutely integral part of working with Houdini and it took a while before they began to make sense. Attributes can exist on many levels of the geometry: on vertices, points, polygons (called primitives in Houdini), and as a detail attribute. The same object can have attributes on every level of its geometry. One important discovery was realizing attributes could be set on geometry going into a simulation. By using attribute names the simulation recognizes, for example v for velocity, the behavior of the simulation could be affected without introducing any additional forces. The same works with attributes used in rendering, such as opac for opacity.

The primary resources for tutorials were Pluralsight, YouTube, and Vimeo. Pluralsight had the most consistently high-quality tutorials, but it is a subscription service. YouTube and Vimeo had free tutorials, but their quality could of course vary wildly. The tutorial's age had to also be taken into account as some older ones referred to features no longer used in the newer versions. When searching for tutorials, a rather wide net was as rarely did they deal with the exact problem at hand. Usually it was helpful to watch a tutorial if it seemed even vaguely related to a problem, as there could be some piece of information there that would lead towards a solution.

5 Project Breakdown

Here, each step of the project is broken down and the workflow explained. The project consists of four main parts: fluids, physics, particles, and smoke. Each simulation requires some preparation beforehand, whether it's creating collision geometry or guides for objects to follow.

5.1 Foundation

Construction of the project began with a simple four-sided tube object (figure 2). The features of this tube control the number of sources for the fluid simulation and the location of the crystal. The number of rows was locked as that would disrupt the point order required by other nodes. The polygons of the tube are unnecessary, and they're removed using an Add node leaving only the points behind.

An Attribute Wrangle is a node that can execute snippets of VEX code. Here it is used to calculate the distance between two points, one on the bottom of the tube and the other on the top, and store it as an attribute called 'srad'. The distance is calculated by measuring the length of a vector pointing from one point to the other.

The top and bottom points are separated with Blast nodes. The vertical position of the top points is slightly randomized with another Attribute Wrangle by adding a small random value to the points' y-coordinates. From the Attribute Wrangle the stream splits temporarily into three. In one stream the top points are fed into another Add node, this time to create a single polygon. The polygon goes into a Primitive node and scaled down, so all the points are on top of each other. A Fuse node fuses the four points into one. This is the anchor point for the crystal. In another stream the top points are grouped into a group called 'top' using a Group node. The third stream goes into a Merge node to combine the top and bottom points back into a single object.

An Add node is once again used, now to create lines between the points. The number for the number of points to skip between each new line is fetched from the original Tube object. This way the node remains functional even if the number of sides is changed in the tube.

The resulting four lines go through a Resample node to add multiple new points so the lines can deform smoothly. The Resample node also adds a ‘curveu’ attribute. This an attribute on the points along each line going from 0 at one end to 1 in the other.

With a Group Transfer node, the ‘top’ group is transferred to the four points on the top of each line. If the ‘top’ group had been present before the resample, the group would have gained unwanted points.

A Soft Transform node is used to create spirals out of the straight lines. Here the ‘srad’ attribute is used to control the Soft Radius parameter so that the node continues to perform its function correctly even if the shape of the original tube is changed.

The spiraling lines go into an Attribute VOP which is a visual counterpart to the Attribute Wrangle. In the end the Attribute VOP executes VEX code like the Wrangle node, but unlike the Wrangle the VOP uses a node-based workflow and creates the code based on the nodes. Which node to use depends on the task, some are simpler to accomplish in a Wrangle, some in a VOP. Here a VOP was simpler.

Inside the VOP a noise field is created with Anti-Aliased Noise, this noise is added to each point’s position to slightly offset each one. Certain parameters in the noise node are promoted so they can be adjusted outside the VOP. The ‘curveu’ attribute is used to drive a ramp with which the amount of noise can be controlled along the length of each curve. The ramp is shaped so that the noise is removed at each end of the lines to keep them in place. The offset of the noise is animated with an expression tied to the current frame to make the lines move. Finally, the lines are converted into NURBS curves to further smooth them.

Here the procedural nature breaks down when each line is separated to its own object using Delete nodes. This somewhat inelegant solution had to be used so the fluid simulation would function as needed.

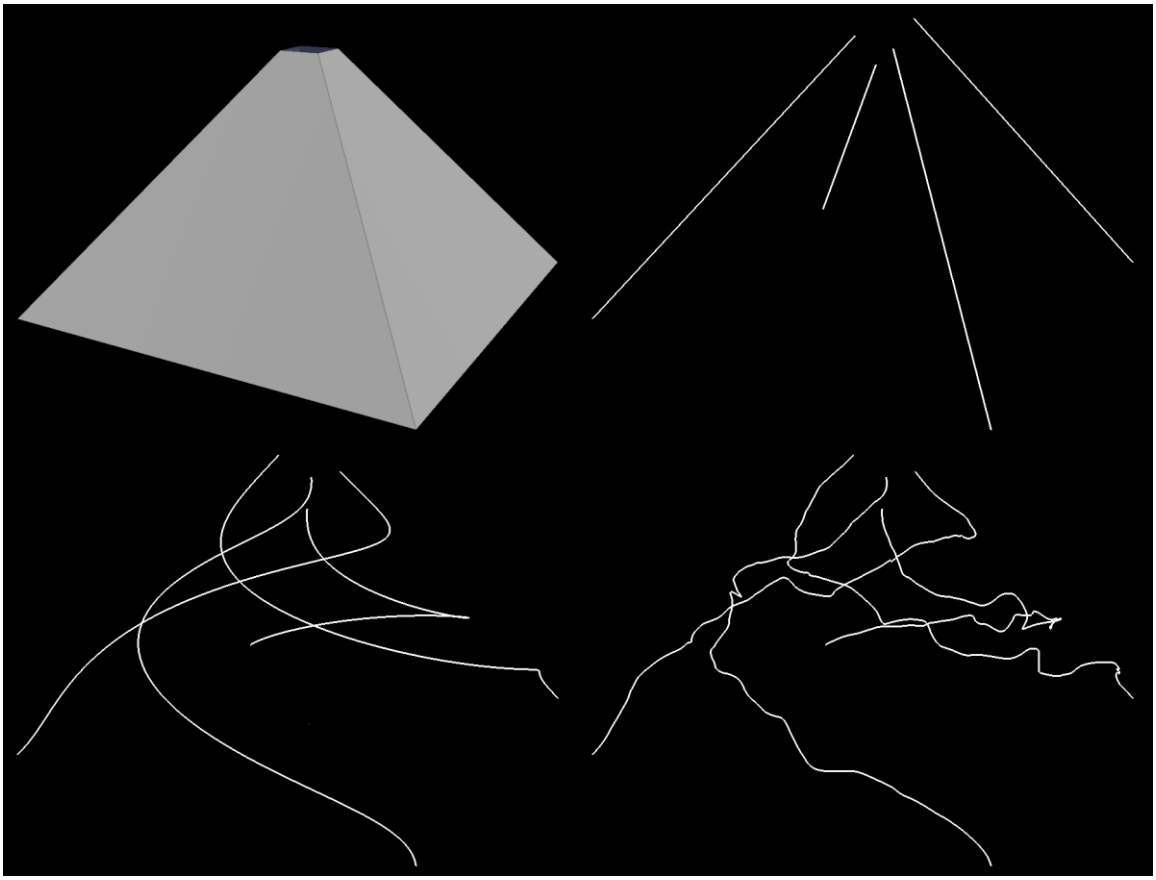

Figure 2 Tube object is turned into noisy lines from top left to bottom right

5.2 Constructing the crystal

The crystal begins as a sphere set to polygon mesh, that way the top and bottom points remain as points 0 and 1 for the next step regardless of the number of rows and columns. A Soft Transform node is used on points 0 and 1 to achieve the general shape. The regular polygon shape of the sphere is broken up with two Remesh nodes, the first with a higher Target Edge Length to break up the shape and the second to add more polygons for detail. A Mountain node is used to add noise to surfaces. Here it works well to break the shape up even more and add natural looking peaks and irregularities to the crystal.

To place the crystal in the correct position in space an Object Merge node brings in the single point created along with the lines. The crystal is moved to the correct position with a Copy to Points node. A File Cache node saves the crystal to disk.

The resulting object doesn't yet look like a crystal. A Normal node with a low Cusp Angle is added to create a faceted look. The Cusp Angle determines the maximum angle between polygons before they stop appearing as a single smooth surface and instead gain a visible edge. Progress of the crystal construction can be seen below (figure 3).

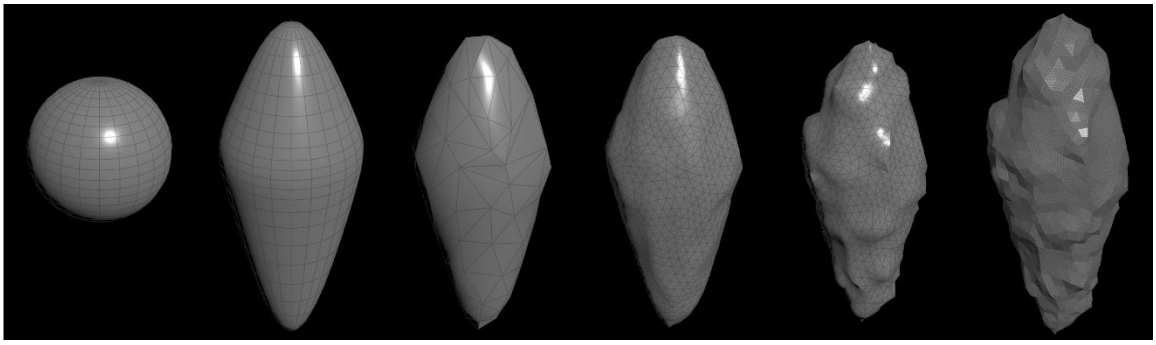

Figure 3: Stages of the crystal construction from left to right

5.3 Fluid simulation

There are two fluid simulations in the project, the results of the first leading into the second. The construction of both, and everything related to them, is detailed below.

5.3.1 Sources

The source where the fluid particles are emitted from is a sphere shaped to resemble a water droplet with a Soft Transform node. Locations for the sources are brought in with an Object Merge from the Lines network, and the sphere is duplicated to each point with a Copy to Points node. A null node added to the end once again helps with pointing to the correct position from other networks.

5.3.2 Target

To make the target for the fluid simulation to flow into, a sphere with the correct dimensions is needed. Instead of roughly estimating the correct size and location of the target sphere, its dimensions can be fetched from the crystal object itself using the bbox() function in the x, y and z

sizes. For the location the centroid() function is used to find the crystal's center. The functions can be entered into the parameters of the sphere in place of numbers.

The resulting sphere is turned into an SDF, or a Signed Distance Field volume using a VDB from Polygons node. In an SDF each voxel stores the distance to the closest surface and whether it's on the inside or the outside of the surface. Positive values are on the outside, negative values are on the inside, at the surface the voxels have a value of 0.

To get vectors from the volume a VDB Analysis node is set to calculate the gradient, or the change in the volume's values. The vectors can be given a custom name and they're named 'force' as it is a standard attribute name recognized by the Dynamics networks.

Next, a Volume Wrangle and some VEX code shape the vectors. The code is commented in the picture below (figure 4). The vectors are shaped to spiral inwards to keep the fluid particles in motion and also attempt to avoid a cavity inside the fluid by forcing the particles towards the center.

```
@force = normalize(-@force); //Reverse force vectors to point inwards

vector up = set(0, 1, 0); //Set an upwards pointing vector

float blend = chf("Blend"); //Spare parameter

float surface = volumesample(1, 0, @P); //Fetch the value of the SDF

if (surface < 0) { //Only apply to area inside the target sphere

    vector tangent = normalize(cross(up, @force)); //Flatten vectors to follow the sphere
    @force = tangent * blend + (1 - blend) * @force; //Blend vectors into spirals
    @force *= chf("Force_Multiplier"); //Multiplier on force

}
```

Figure 4: Vex code for shaping vectors

5.3.3 Simulation

To get started with simulations in Houdini it is often helpful to use the shelf tools to set up the basic nodes and networks. With the source geometry selected, it was turned into a particle fluid emitter. This creates a Dynamics network with the necessary nodes and adds a Fluid Source node to the source geometry. These nodes could be created manually but using the shelf tool creates the correct connections between them and links required parameters.

To keep everything clean, the created nodes are moved into their own sub-network. Even if nodes are moved to new networks all linked parameters are automatically updated.

The lines that were created at the very beginning of the project are needed now, so they are imported with an Object Merge and connected to one of the available inputs on the Dynamics network.

Simulations in Houdini follow a similar structure: they need a source, a container for the simulated objects and a solver. The shelf tool had already created the basic nodes for a fluid simulation, so only their settings needed to be tweaked.

The first settings that need to be changed are on the FLIP Object and the solver itself. Particle Separation on the FLIP Object determines the “resolution” of the simulation by controlling how close particles can get to each other. The lower the value, the more particles and a more detailed simulation.

The default base unit in Houdini is 1 meter, so a particle separation of 0.1 would mean 10 centimeters between each particle. This of course doesn't catch much of the finer details in a smaller scale simulation but might result in far too many particles in larger simulation. It’s a trade-off between detail and simulation time, and storage space as simulations with tens or hundreds of millions of particles can require hundreds of gigabytes of hard disk space.

For this project the particle separation was set to 0.035 for the final simulation. This gave a maximum particle count of around one million. But for initial look development the particle separation was increased to 0.2 so the simulation could run almost in real time and there were no lengthy wait times between changing other settings.

Another very important setting is on the FLIP solver node under Volume Motion and Volume Limits. This determines the size of the simulated volume and is best minimized to cover only the space that is needed.

To have the particles follow the lines that were created earlier and collect at the target location, several other nodes were needed. Because FLIP simulations deal with particles, nodes intended for regular particle simulations can be used with fluid simulations.

With the POP Curve Force node, the lines are brought into the simulation. The node works best with single lines, so as there are four lines four nodes are needed. Later another four were added for additional control. All the nodes need to do the same thing, so all their important parameters

were linked so only one had to be edited. To achieve the desired look for the movement of the particles along the curves, values on the first two curve force node were animated with keyframes, the values would be copied to the rest of the nodes. These nodes are all merged together and wired into the Particle Velocity input of the solver. A picture of the simulation at this stage can be seen in figure 5.

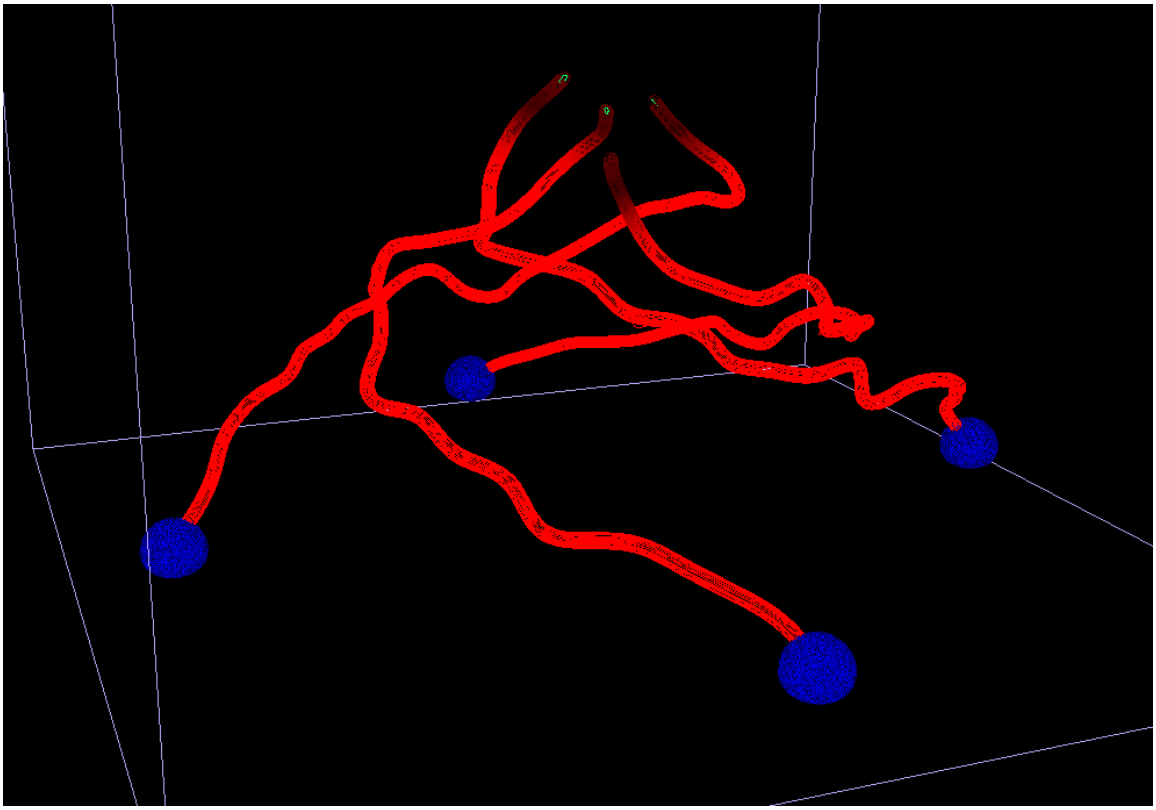

Figure 5 Fluid simulation with fluid sources in dark blue, curve forces in red, and the bounds of the simulation in light blue

Two nodes are needed to get the target for the fluid working: A Field Force and a SOP Vector Field. The vectors made earlier were named ‘force’, so they are automatically recognized by the node. The dimensions and divisions size of the Vector Field are linked to the dimensions and particle separation of the FLIP simulation. Apart from scaling the force up slightly, the Field Force node does not require any more set up. Gravity is not needed in this simulation so the automatically created Gravity Force is deleted.

Once a desired look was achieved the simulation can be cached, but first the source node was configured to stop emitting new particles at frame 300 with ‘$F < 300’ in its activation text box. $F is a global variable in Houdini that refers to the current frame number.

5.3.4 Caching and surfacing

Outside the Dynamics network its contents are fetched with a DOP I/O node and wired into a File Cache. The File Cache can be attached directly to a Dynamics network, but with an I/O node the Dynamics network can be filtered to only output what is needed. The FLIP fluid preset selects the correct fields for import. On the File Cache node, the frame range is set correctly and 'Load from disk' is checked so the node reads the files from disk if possible and the simulation doesn't need to run anymore.

The fluid simulation comes out of the DOP Network as particles and volumes. To look like water when rendered it needs to be turned into polygons. To do this the File Cache is wired into a Particle Fluid Surface node. Only the particle separation setting was changed to match that of the fluid simulation. A second File Cache after the surfacing node writes the surface files to disk.

5.3.5 Exploding liquid simulation

After the first simulation has been cached, its last frame can be used as the starting point for the explosion simulation. Because there are almost a million points all the unnecessary attributes are stripped from them leaving only velocity. For look development a temporary Attribute Wrangle is added that removes a certain percentage of the points to make the simulation faster. For the final simulation the node was disabled.

Houdini 17 introduced a new fluid solver called POP Fluid that wires into a normal Particle Solver. Both the POP Fluid and the FLIP fluid were tried for this part, but a satisfactory look could not be achieved with the POP Fluid. Due to time constraints the more familiar FLIP fluid was used instead.

This DOP Network was built from scratch. The geometry file from the previous simulation already had all the required fields, so they only needed to be piped into the solver. Also, because only the existing points will be used, no source is needed for the simulation.

This simulation will also collide with objects, so they need to be imported. The crystal is imported with an Object Merge and wired into the DOP Network. Because the collisions are against static, or non-dynamic objects, a Static Solver is created. The crystal is brought in as a Static Object. A mistake was made when setting up collisions which was only discovered after the simulation had

been run and the final renders done. FLIP, like other particles, uses Rigid Body Dynamics for collisions, not Bullet. The default RBD collision was just good enough that the error wasn't noticed. It was only discovered after the sequence had already been simulated and rendered.

The fluid also has to collide with the ground, so a Ground Plane is created. The Ground Plane does not require any additional settings, it is just an infinitely large plane for objects to collide against. Both the Static Object and the Ground Plane are merged together and wired into the Static Solver. The Static Solver also has no settings to change. The FLIP Solver and Static Solver are also merged together and wired into a Drag Force which adds air resistance to the simulation. The final node is a Gravity Force.

Most of the look development for this simulation happens on the FLIP Solver. On the FLIP Object only the particle separation is set to that of the previous simulation and a very small amount of viscosity added. To get the viscosity working it also has to be enabled on the solver. On the same page of settings, Slip on Collision is enabled with a value of 0.9. This means upon collision the fluid particle is allowed to slip along the colliding surface with 90% of its initial velocity instead of sticking to the surface or bouncing off.

A Life attribute was added to the particles outside the simulation and to take advantage of it the Age Particles and Reap Particles settings are enabled so the particles will be removed after reaching the end of their lifespan. Once the desired look was achieved the simulation was once again cached to disk and surfaced just as it was with the previous fluid simulation.

5.4 Breaking geometry for physics simulation

For the crystal shattering simulation, the crystal geometry must first be broken into pieces (figure 6). Breaking the geometry before the simulation gives more control over the look. The two different methods used are detailed below.

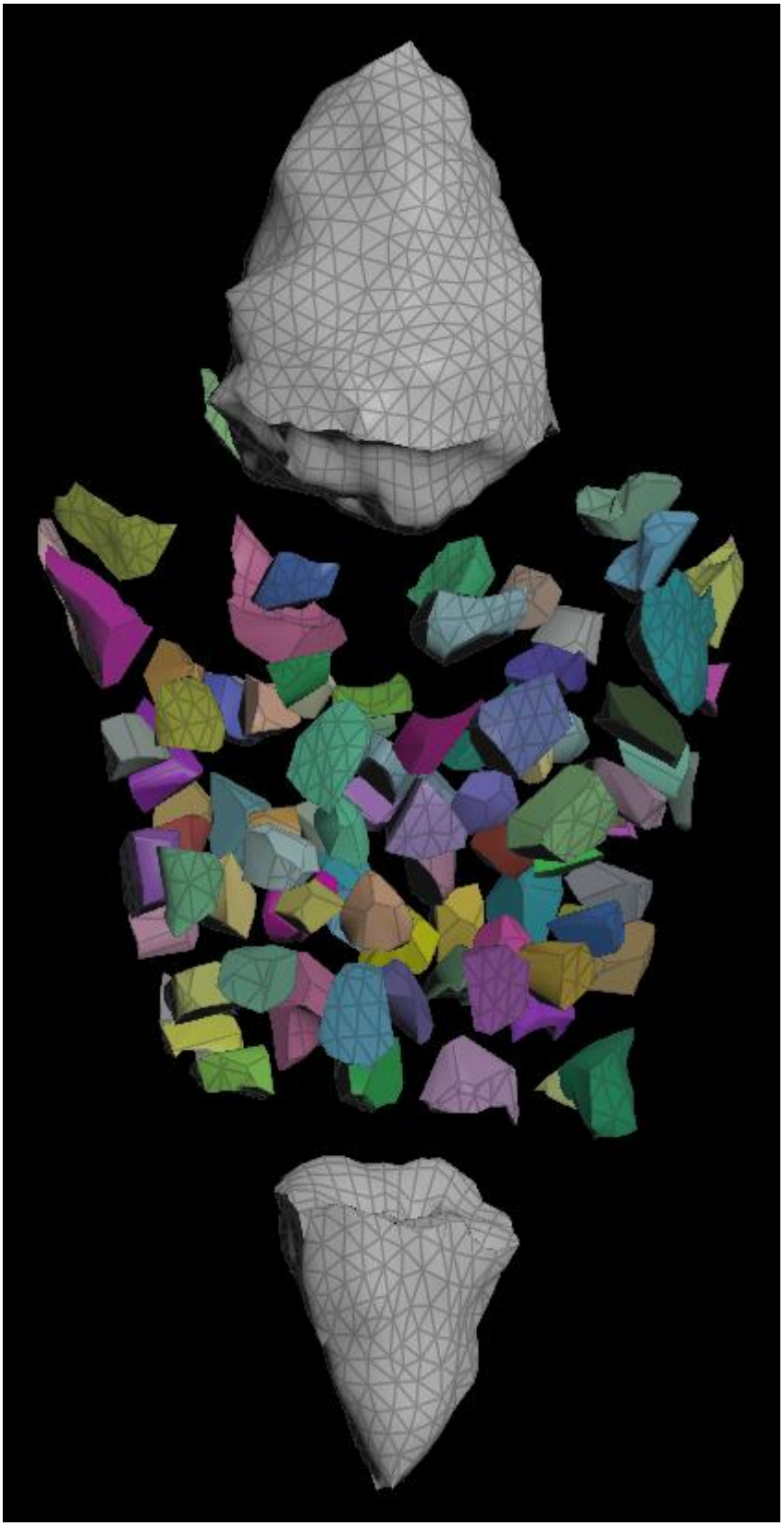

Figure 6 The crystal shattered into pieces before simulation

5.4.1 Using Boolean shatter

For cutting the geometry, a Boolean node is used. In addition to combining and subtracting geometry, it also has a shatter function. In this mode the geometry of one input is cut along the input of the second. A copy of the crystal was wired into the first input.

The cutting geometry was originally created separately while testing the Attribute Wrangle node. In the test the points on a line were transformed into a parabola by mapping their y coordinates to x^2. Then the points on a grid were made into a paraboloid with y = x^2 + z^2. This shape looked suitable and for added control the curvature could be controlled with an overall multiplier.

Using a Transform node, offsetting along the y-axis, rotating by 180 degrees along the x-axis, and merging back with the original there were now two mirrored grids. These were further deformed with a Mountain node. The grids were moved to the correct position by getting the centroid of the crystal with a second Transform node.

The grids were wired into a Boolean node and the node mode changed to Shatter. The Boolean node automatically creates a group called ‘apieces’ and this will be helpful in separating the pieces later.

5.4.2 Using Voronoi fracturing

In a new network the shattered crystal is brought in with an Object Merge. Using two Blast nodes and the ‘apieces’ group the middle can be separated from the ends. One Blast deletes everything belonging to the group and the other everything not belonging to the group.

With the middle of the crystal isolated, its volume is filled with points with a Points from Volume node. The Point Separation value determines the number of pieces the middle will be fractured into. A value of 0.25 resulted in 92 pieces. Jitter Scale introduces random noise to the point grid, making it less orderly.

The middle of the crystal was wired into the first input of a Voronoi Fracture node as the geometry to be fractured. The points created previously were wired into the second input. No additional settings needed to be changed on the fracture node.

5.5 Crystal shatter simulation

The now fractured middle piece of the crystal is merged back with the ends and packed with an Assemble node. Packed geometry cannot be edited but its memory footprint is reduced and is the default workflow when doing physics simulations with non-deforming geometry.

To have the crystal explode, velocities are added to each piece with an Attribute Wrangle. The geometry is already packed, so the attributes can be added to the points at the centers of each piece of the crystal. By wiring the original crystal geometry from the Object Merge into the second input of the Attribute Wrangle the center of its bounding box can be referenced with the function getbbox_center(1), the number 1 referring to the second input of the node. This position was subtracted from the centers of each shard to create vectors pointing outward. These vectors are then assigned to the velocity attribute v. Angular velocity is the attribute w and is randomly added to each piece to give them some rotation.

Three additional controls are created. One for dampening the y-axis value of the velocity vectors so the motion of the shards is mostly sideways rather than up or down. The last two are overall multipliers for velocity and angular velocity. This way the important values can easily be tweaked without looking for them in the code.

The shattered and packed crystal is fed into a Dynamics network. On the network node itself the Scale Time parameter is lowered to 0.1 resulting in the simulation playing at 10% of normal speed.

Inside the DOP Network, the crystal is brought in with an RBD Packed Object node and connected to a Rigid Body Solver. For collisions, only a Ground Plane is needed along with a Static Solver. Both solvers are merged and connected to a Drag Force to simulate air resistance. Ignore Mass is turned on so the mass of the crystal isn't taken into account and the node can dampen velocities faster. The last node in this network is a Gravity Force. Both force nodes are set to activate at frame 200 by entering the expression '$F > 200' into each Activation parameter.

This time the results of the simulation are fetched directly from the DOP Network without a separate import node. Unnecessary attributes are deleted with an Attribute Delete node leaving only velocity, angular velocity, and the name attribute unique to each piece.

Because the Time Scale was set to a low value, 2000 frames are written to disk with a File Cache node. The simulation did not take long due to its simplicity. A "normal" speed version of the simulation was created from these files with another File Cache set to write 1 frame out of every 10.

5.6 Disintegration

The effect of the crystal pieces being eroded away is done with a Boolean operation, the same node was used previously when the crystal was shattered into three pieces. The basic procedure of the erosion effect is the same for the small and the large pieces, but their implementations differ. Both are explained below.

5.6.1 Small shards

The shattering simulation is loaded in with a File node. The top and bottom pieces need to be removed as they will be disintegrated in a separate network. Because the geometry is still packed entire pieces can be deleted by deleting the point representing that piece. The two large pieces are the two last points. The pieces could be deleted with a Blast node, but if the number of pieces was later changed the node would need to be edited each time to delete the correct points.

An Attribute Wrangle and the function removepoint() can be used to delete the correct points no matter what their numbers happen to be. The attribute @numpt is a built-in attribute that refers to the total number of points in the geometry coming into the wrangle node. Point numbers begin from 0 and the number of points is an integer value starting from 1 so @numpt-1 and @numpt-2 tell the function to remove the correct points.

With only the middle pieces remaining the geometry is unpacked using an Unpack node so it can be edited. Each shard needs to be worked on individually and because the shards still have the name attribute from the Voronoi fracture, a For-Each Named Primitive loop is used.

Within the loop, a single point is placed on a random spot on the piece's surface with a Scatter node. Onto this point a sphere is copied with a Copy to Points node. The sphere's size is animated by keyframing the Uniform Scale parameter so that it starts from 0 and grows to its final size over time. The shape of the sphere is changed with a Mountain node. The height of the noise is fetched from the scale value on the sphere so that the noise doesn't turn the sphere inside out when it is very small. The offset for the noise is taken from the iteration attribute on the metadata node of the loop. Because this attribute is different each time the loop is run, each Mountain node has a differently shaped noise. The sphere is then turned into a VDB volume with a VDB from Polygons node.

This part of the network, including the Scatter, Copy to Points, and the sphere, is duplicated so each piece has two randomly placed spheres. The two sphere volumes are merged with a VDB Combine, then turned back into polygons with a Convert VDB. The piece being worked on is also first turned into a VDB and back into polygons with the same nodes.

Both the piece and the spheres are fed into a Boolean node set to Subtract. The node is set to output a group called ‘binsidea’ which stores the polygons from the spheres that intersect the piece (figure 7).

The reason everything was turned into VDBs and back to polygons is because the Boolean operation resulted in artifacts when working with the original geometry. Using VDBs slowed the network down considerably, but it was the only way to get clean results. Because the ‘binsidea’ group was necessary for later parts of the project a Boolean node had to be used, otherwise a similar effect could have been achieved with a VDB Combine. After unnecessary attributes are cleaned from the output of the For-Each loop, the results are cached to disk.

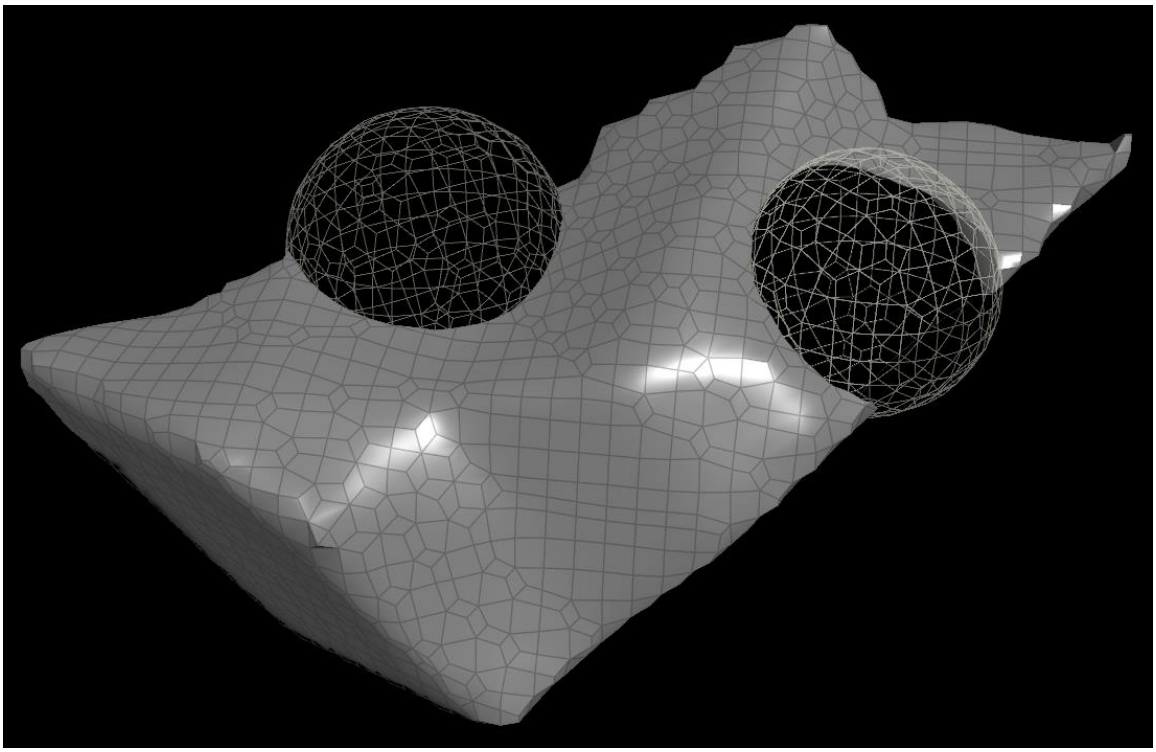

Figure 7 A single shard being disintegrated, the spheres grow over time

5.6.2 Big shards

The big shards aren't moving while they're disintegrated so only a single frame where they had settled onto the ground is needed from the simulation. This time the middle part of the crystal isn't needed. Using an Attribute Wrangle, the big shards are added to a group called 'big' using the function setpointgroup() and the @numpt attribute. A Blast node deletes everything that isn't part of the group.

To create the cutting geometry for the Boolean operation the packed geometry of the shards is piped into a Bound node, which creates a box geometry from the bounding box of the input. The created box only has 6 polygons which isn't enough for the Boolean. A Remesh node vastly increases the polygon count. The box is moved above the shards with a Transform node. The y-value for the scale is keyframe animated to make the box grow in height until it completely covers the shards. The box's pivot is still based on the shards so transformations can behave unexpectedly. In the Pivot Translate on the Transform node the variables $CEX, $CEY, and $CEZ are entered into the corresponding parameters. This moves the pivot to the center of the box.

Next, the shape of the box is altered with a Mountain node. The values for the noise offset are a result of trial and error to find values that result in an error free Boolean operation. A Smooth node does what its name implies and smooths the harsh edges from the Mountain. Finally, a Normal node adds normals to the geometry.

The big shards are unpacked and re-meshed with Unpack and Remesh, respectively. Similarly to the small shards, the big shards are fed into a Boolean set to subtract with the box geometry in the second input. The 'binsidea' group is also enabled. Lastly, the sequence is cached to disk.

5.7 Particles

The cached frames of the disintegrating small shards are loaded with a File node with two nodes attached under it. First, a Collision Source node creates an SDF volume from the shards for collisions in the particle simulation. Most of the settings on the node can be turned off as they're not needed, nor would they work with the geometry used. Interpolation and velocity calculations require consistent point numbers. The shard geometry is created from a volume and so has different topology on each frame. The only option that needs to be changed is the Voxel Size of the

resulting volume. The collision volume is then cached to disk so it does not need to be created every time the simulation is run.

In the second branch, the 'binsidea' group created with the Boolean is used with a Blast node, deleting everything not in the group. This leaves only the surfaces that are being eaten away and they will be used as the source for birthing particles. This geometry is connected to a DOP Network for particle simulation. Also wired into the DOP Network is the position of the crystal, brought in as a point with an Object Merge. On the network node itself, substeps are increased to 3

Similarly to other simulations, particle simulations also require a solver, a source, and a simulation object. No settings need to be changed on the POP Object and the only change to the POP Solver is increasing the maximum substeps. The POP Source is responsible for creating the particles and setting their initial attributes.

On the POP Source node, the Geometry Source is set correctly to read from the DOP Network's first input. The input geometry changes shape and size so the setting for Scale Point Count by Area is turned on, this way the density of particles emitted remains roughly the same despite the size of the emission area changing.

On the Birth tab are two ways particles can be generated. Impulse activation generates particles every time the node is calculated, which could be multiple times per frame if the substep count is increased. Constant activation generates the desired number of particles spread over however many frames constitutes one second determined by the framerate of the scene. Here the framerate is 24, so the particle birth count is divided over those 24 frames.

Although the original crystal shattering was simulated at 10% of normal speed, it only plays back slowly, velocities and forces involved are not scaled down. Therefore, the inherited velocity has to be lowered to 0.025 of the original. Otherwise the particles would shoot out far faster than the shards are moving.

POP Fluid was tried again here to give the emitted particles a more fluidlike look. The POP Fluid node applies constraints between nearby particles, trying to keep them close. Particle Separation controls how close the particles can get to each other. Constraint Stiffness and Constraint iterations determine how closely and how accurately the separation is enforced. A tiny bit of viscosity was added on top. Here the node worked well to produce long tendrils and thin sheets of particles.

To give the particles additional motion, two more nodes are used. Pop Wind introduces random motion to the particles with an animated noise. The amplitude of the noise is lowered somewhat and the swirl size increased, giving the particles subtle, large scale motion. A POP Attract normally pulls particles towards a location. The location is set to the crystal's location, which was connected to the second input of the simulation network. By setting the force scale to a negative number, the node instead pushes particles away. The force scale is also keyframed to increase the force pushing the particles away tenfold later in the simulation.

The collision volume was created outside the simulation, but it has to be brought into the simulation for collisions to work. This is again done with the Static Object and Static Solver nodes. Because the collision geometry is animated, the Use Deforming Geometry option has to be turned on. Otherwise the node will only evaluate the geometry on the first frame. Particles use RBD, or Rigid Body Dynamics for collisions, so all the relevant settings are under the RBD tab of the node. The collision mode is changed to Volume Sample, the Division Method to By Size, and the Division Size is linked to the Voxel Size value on the Collision Source node outside the simulation. The Proxy Volume path is pointed at the cached volume data written to disk earlier.

Outside of the DOP Network, an Attribute Delete node is added to remove unnecessary attributes from the particles before a ROP Geometry Output node writes them to disk. The reason this node was used instead of the regular File Cache is because the particle simulation was wedged.

5.7.1 Wedges

The Wedge node is found in the Output context along with other render operations. Its function is to execute a render operation multiple times while changing certain parameters each time. Here it was used to run the particle simulation multiple times. The Seed value from the POP Source was set to change for each pass, otherwise each simulation would be identical.

The Wedge node defines a prefix, which by default is _wedge. To use the prefix, the variable $WEDGE is added to the file path in the render operator used. The name of the wedge parameter used is seed, so the file name particles_small_shards$WEDGE becomes particles_small_shards_seed_0, followed by _seed_1 and so on.

Use of the wedge was not strictly necessary as there were not enough particles to run out of memory, rather it was used simply to become familiar with it.

5.8 Pyro smoke simulation

The smoke simulation for the big shards begins the same way as the particle simulation for the small shards. Under the File node that loads in the geometry are the same two nodes and for the same purposes: Collision Source for collisions and Blast to isolate parts of the geometry used as sources.

A pyro simulation requires more nodes to get started, so the initial setup was done by selecting Billowy Smoke from the Pyro FX shelf. The first node created is a Pyro Source which scatters points onto the incoming surfaces and creates attributes relevant to a smoke simulation on the points, namely density and temperature. The second node is Volume Rasterize Attributes and it turns the points from the previous node into volumes with the same name as the attributes on those points.

The shelf tool also creates a DOP Network with many of the required nodes already created. A pyro simulation has a similar node structure to other simulations. The Smoke Object defines the voxel size and the initial size of the simulation container. The Gas Resize Fluid Dynamic node dynamically changes the size of the container and it's set to track the source geometry. The maximum bounds are also turned off so the container can grow as large as it needs to. A Volume Source node brings in the volumes created earlier, it is automatically set up by the shelf tool so nothing needs to be changed on it. The collisions however need to be added manually, so a second Volume Source is wired in with a Merge node. The Initialization is changed to Collision and pointed at the volume created by the Collision Source node. This is a decidedly different way to do collisions than the usual static object.

The behavior of the Pyro simulation is handled on the Pyro Solver. First, the temperature diffusion, how fast the temperature differences in the gas even out, and cooling rate, how fast the gas cools down, were lowered. Buoyancy lift was also lowered, this controls how fast the gas rises.

Only smoke is needed in this simulation, so the setting for enabling combustion could be turned off. The look of the smoke is controlled on the Shape tab. When working with the different controls, it helped to turn off every other setting entirely, going through them one at a time.

Once the simulation had been set up, it was imported back to the geometry level with a DOP I/O node. The same node can also write the selected fields to disk once the correct DOP network and object are selected.

5.9 Building the final scene

Once everything had been simulated and written to disk, a new scene file is created. Technically everything could have been done in the same file, but it was much cleaner to move to a new one and load just the files from disk.

The original intent was to have all the sequences play one after another, time shifted into correct positions. This idea was discarded as it was unnecessarily complicated. Instead, a camera was created for each part and rendered separately.

For the first shot, the camera tracks the tendril of liquid. To achieve this, the lines used for guiding the liquid in the simulation were imported but without the noise applied to them. A sphere is then created and made to follow one of the lines. The camera was then made to look at the sphere. Both of these, path follow and look at, use CHOPs or Channel Operators. CHOPs are a part of Houdini that was not studied much for this project. For the features mentioned earlier, the shelf tools were used. The position along the curve the sphere is at is a value from 0 to 1 on a Follow Path node and this value was keyframed to roughly match the speed at which the liquid advances. The Lookat node determines the object the camera is looking at and required no additional setup after it was created by the shelf tool. For subsequent shots the camera positions were either static or had their movements manually animated.

During the formation, the crystal is only visible when it's covered by the liquid. To achieve this, volumes from the fluid simulation were used to find the parts of the crystal that were inside the liquid. This was done with an Attribute Wrangle, the function volumesample(), and a simple if-statement. The value of the fluid volume at the sampled position was stored into a temporary variable. If the value was a negative number, that part of the crystal was inside the liquid. Two attributes are then written to the point: opac and emitint. Both are attributes that are recognized by the renderer, opacity with opac and emission intensity with emitint. Outside the liquid those values are 0, making that part of the crystal invisible.

The shattering crystal switching between normal speed and slow motion is done with Switch and Retime nodes. As its name implies, the Switch node switches between the nodes wired into it. The switching can be done manually, by keyframing, or with expressions. In this case it was easiest with expressions. The Retime node is used to shift the cached animations to correct positions in time. The same combination of Switch and Retime nodes is used in three networks to sync three parts of the animation together.

Out of the simulation, particles simply vanish at the end of their life. They have the age and life attributes on them and those can be used to make their disappearance smoother. With an Attribute VOP a particle's age is divided by its life creating a value going from 0 to 1. The result of the division is wired into two Ramp Parameters, one for color and one for a float value. The output of the color ramp is connected to the Cd attribute, which is the default attribute name for color in Houdini. An extra parameter is created and multiplied with the output of the float ramp and exported to the attribute pscale. Pscale is another default Houdini attribute, this one for controlling point scale. With both ramps shaped, the particles now change color and size as they age, eventually shrinking to 0 size at the end of their life.

5.10 Rendering and lighting

The entire animation takes place inside a dome. The dome itself is lit by a single point light set to only illuminate the dome. Lighting for the rest of the animation is done with an environment light and an environment map from the website HDRIHaven [6]. The dome was excluded from casting shadows from the environment light.

The renderer used by Houdini is called Mantra. For rendering the animation not many settings were changed. To keep render times low, render quality could also not be increased much. The main driver of quality in the render are the pixel samples. Pixel sampling of 3x3 means a minimum of 9 rays are sent into the scene per pixel. Higher values were tested, but they resulted in much higher render times, so the setting was kept at 3x3.

Motion blur had to be turned off for the segments of the crystal shattering. Geometry motion blur relies on point velocities to calculate the blurring and since the shard topology, and subsequently the point numbers, changes each frame no accurate velocities can be calculated.

The animation was rendered out as numbered sequences of PNG image files. No additional image planes were used as no post-processing was planned. After all the sequences were rendered, they were compiled into a video file using Blender. Blender is a completely free 3D modeling software which also includes a video editor.

6 Closing thoughts

The original intent of this thesis was to learn the basics of Houdini. This goal was reached, but in doing so it showed how only the surface was scratched and just how much more there is to learn. The project did instill a desire to learn more and new ideas for future projects came up while working on this.

But what of the project done for this thesis? The final product is quite simple, and different from the initially imagined version. This is, of course, to be expected. It was a learning process after all. Even the simplest things seemed insurmountable in the beginning, but towards the end there was a conscious effort to stop adding new things and concentrate on finishing.

In hindsight it might have been better to create multiple self-contained effects rather than one where each part led to the next. This might have resulted in more varied effects and given more freedom in them.

Whether Visual Effects will be the primary focus going forward remains to be seen, but it will be studied further. 3D character design is another field that will be studied alongside VFX. And while character design might be more weighted towards game development, VFX has a home in film and TV as well. This might hopefully widen the scope of potential work opportunities in the future.

Sources

1 Nuts Computer Graphics, Special effects and visual effects: what is the difference?

https://www.nutscomputergraphics.com/en/special-effects-and-visual-effects-what-is-the-difference/

Reference 2.5.2019

2 fxguide, Side Effects Software – 25 years on

https://www.fxguide.com/featured/side-effects-software-25-years-on/

Referenced 27.4.2019.

3 The Globe and Mail, At SideFX, 'we know that we'll be in every movie up for the visual-effect award at the Oscars'

https://www.theglobeandmail.com/arts/film/at-sidefx-we-know-that-well-be-in-every-movie-up-for-the-visual-effect-award-at-theoscars/article37796101/

Referenced 29.4.2019.

4 Houdini Games Reel 2019

https://vimeo.com/323879609

Referenced 29.4.2019.

5 Houdini Apprentice

https://www.sidefx.com/products/houdini-apprentice/

Referenced 25.4.2019.

6 HDRI Haven, Noon Grass environment map.

https://hdrihaven.com/hdri/?c=skies&h=noon_grass

Referenced 18.1.2019.