A simple particle simulation in Houdini

Nikitha Ann George

Faculty of Media and Communication
Bournemouth University

This dissertation is submitted for the degree of
Masters in Computer Animation and Visual effects

Bournemouth University August 2016

Abstract

Houdini gives the simple user different methods for creating visually interesting environmental effects . In this program, three different types of objects simulation are shown: a collection of **soft objects simulation, hard objects and very hard objects**. This paper presents an effective method to simulate the ball diffusion process in real time to produce realistic movements. A group of objects that interact with an object, to enhance the particle motion effects, a force is required . This force might not be physically accurate, but it proves effective for producing animations.

Table of contents

List of figures

Chapter 1

Introduction

Particles are essentially single vertices or points that can have dynamics and forces applied to them generating movements that produce a desired outcome. Each individual particle represents a location in 3D space, but simulations can also calculate additional information per-particle such as speed and the age of the particle, which can be applied in shading to produce a more dynamic result. A single moving particle doesn't offer any really compelling benefits but as particle are simple, thousands or even millions of them can be used in a simulation producing effects that would be difficult, if not impossible to generate any other way. Modo provides a robust and flexible node based particle simulation platform that can be used to simulate a huge range of phenomena. Fire effects, smoke and fluid simulations are all possible in this manner. And while the underlying framework relies on nodes, for most basic simulations, nodes will auto connect when added to the scene requiring little, if any intervention from the user. **For users that require advanced setups do need to use the schematic view, items are easily added using drag and drop from the Items list to the Schematic Viewport**. All nodes have outputs for their major functions and inputs if applicable. Users can connect more nodes; it is better to have a left to right fashion so as to have a clean representation for the flow of the simulation evaluation. The program has been arranged so as to take inputs on the left and outputs on the right.

All programs are created in Houdini.

Submission_01 A group of particles that seem like they splash throughout, and finally, spreads on the ground. It then moves through a path which is imported from a curve.

Submission_02 Objects that are more elastic, like rubber, thrown onto a wall results in all balls to bounce quickly and fall on the ground because of gravity.

Submission_03 Some objects that seem to be sticky are thrown on the wall and its spread on the ground like fluids. This allows easy control of the animation, it includes basic

solid–fluid interactions. The most attractive visual characteristic of liquids is its flow of the objects and its dynamic diffusion process, as it is drops on the floor, it spreads. A third movement shows some semi solid objects thrown on a wall. On hitting the wall, the objects bounces slowly. This construction proves to be more tightly wound.

Chapter 2

Simulation

2.1 Overview

Particle simulations can contain a number of parameters in its evaluations, such as **forces, modifiers and operators**, but a basic type of simulation require only two important features, an **emitter** and a **Particle Simulator**.

An *emitter* is the source for the generation of the *particles* that define the location of **emission**, which also specify the number of particles that can be emitted and their initial speed and direction. Users can *control the emission point and initial direction* by the position of the actual emitter item in the 3D scene, combined with the attribute settings in the **Properties** panel.

The *particle simulator* defines the *global options for the overall simulation*, such as controlling the particles *lifespan* (if desired) and assigning the *Gravitational forces*. It is also responsible for the storage of the cached values for later rendering. Users can add any number of particle emitters to a single *particle simulation* node. When adding any of the emitter types, first a *particle simulation* is added automatically. In complex setups, it may be desirable to use multiple *particle simulators*. On simulation, all the nodes are active and run simultaneously. The addition of subsequent emitters will be connected automatically to the most recently selected *particle simulation* node.

2.2 Rendering

While particle are visible as points in the 3D viewports, they cannot be directly rendered. In order to actually see particles in a render, users will need to **assign them as point sources to an appropriate effect**, such as *Replicators , Blobs, Volumes* and *Sprites*. The effect chosen

will depend greatly on the desired result. **Replicators** can be used to assign geometry to the particles for effects like *bullets firing and shell casings flying*. **Blobs** can be applied to particles with **Dynamic Fluids** for liquid type effects. Fire and smoke are perfect candidates for **Volumes**, especially when combined with **shader tree textures** controlling the color and density. **Sprites** are useful for image based particles effects, such as adding **falling ashes after an explosion**.

Each effect type has a `particle source` option where the `particle simulation` node can be assigned. It is important that the *simulation must be cached using the appropriate simulation control for the render to actually see the particle*.

Chapter 3

Technical Background

This animation is based on particle simulation. Bringing the balls to the ground by applying gravity through equations is the main theme of this project. For doing this project I've gone through several nodes in Houdini and it is noted below.

3.1 POPNET

`popnet.exe` is a `popnet` belonging to *PHAROS* from **Pharos Systems International Non-System**. Processes like `popnet.exe` originate from software that has been installed on the system. Since most applications store data in the system's registry, it is likely that over the time your registry suffers fragmentation and accumulates invalid entries which can affect the PC's performance.

3.2 POP SOLVER

`POP Solver` is a solver designed to update particles for each time step. `POP micro solvers` control the particles and accumulate forces on the basis of the inputs attached. It integrates using the forces,

`target (v), Air Resist` : The pair describes a local drag frame for each particles.

`force` : Specifies a magnitude of force affecting each particle.

`target (w), Spin Resist` : This pair describes a spin-drag frame. The target w is in world space.

`torque` : Specifies the world-space torque to apply to the particle.

3.3 STATIC SOLVER

The `Static Solver` DOP is a solver that particularly does not do anything to its objects at each time step. Although the behavior of the object is the same as if no solver was attached to the object, it can be necessary to use this solver when using a `switch solver` or `blend solver`. When using those solvers, there is no way to turn off the solving other than using a `Static Solver`.

3.4 hou.Node

In Houdini each node has a unique path that defines its location in the tree of nodes. The node path hierarchy is similar to the hierarchy of folders and files in a file system. Some nodes, called networks, may contain other nodes inside them, much like a file folder would, while other nodes may not. For example, an `object node` instance and a `SOP sub network node` instance may contain `SOP nodes`, but a `box SOP` instance may not.(http://www.sidefx.com/docs/houdini11.1/hom/hou/Node).

3.5 POPVOP - DYNAMIC NODES

(http://www.sidefx.com/docs/houdini/nodes/dop/popvop)

The `POP VOP` operation runs `CVEX` over a set of particles. The `CVEX shader` can be defined as a `.vfl file`, a `SHOP`, or by building a `CVEX VOP network` inside this node. The latter is the most straightforward approach. Binding to the `CVEX script` controls the attributes that are passed to the specific parameters of the script. The default *AutoBind* will use the name of the attributes to determine which parameter they must **override**. If that parameter is marked as *exported* in the `CVEX script`, the attribute will be written to and/or created. Contrarily, the attribute is read to get the parameter's *values*.

3.6 POP ADVECT by FILAMENTS DYNAMICS NODE

This node is used as part of a **crowd simulation** to modify the transforms of agent leg bones to avoid intersection with terrain and prevent the feet from sliding.

3.7 POP ADVECT by VOLUMES DYNAMICS NODE

A POP node that uses velocity volumes to move particles. The Advect by Volumes POP is designed to make it easy to control the horizontal movement of a *particle system by a fluid simulation*. Often the fluid simulation will be simulated as a separate pass and the velocity fields read off the disk. However, the particles can be live-linked to an existing simulation. This operator modifies the *force, velocity, and position attributes*.

3.8 POP ATTRACT DYNAMICS NODE

A POP node that attracts particles to positions and geometry. The PPOP attract node applies a force to particles to steer them towards a target location. This operator modifies the **force attribute**. It attracts particles to positions and geometry and also attracts nodes that applies a force to particles to steer them towards a target location.

3.9 POP AWAKEN DYNAMICS NODE

A POP node that resets the stopped attribute on particles, that is, waking them up. Particles with the stopped attribute set to 1 are considered asleep and consequently, stop moving. This nodes provides tools to modify this, either from existing *volume fields or by the particles* themselves. This operator modifies the **stopped attribute**.

3.10 POP AXIS FORCE DYNAMICS NODE

The POP Axis Force node applies a force around a specified axis. It is either a **line (Sphere)** or **circle (Torus)**. For forces along a curve, see the POP Curve Force. This operator modifies the force, **target (v)**, and **air resist attributes**.

3.11 POP COLLISION BEHAVIOR DYNAMICS NODE

A POP node that reacts to collisions. The POP Collision Behavior node performs operations based on collisions set in the hit attributes. For simple RBD collisions, we use the controls on the *Collision Behavior tab* of the POP Solver. Typically, the POP Collision Detect node is used to do both the **collision detection and resolution**. However, this node can be useful when responding to DOP collisions created by the *Add Hit Attributes* option

of the POP Solver. This does not handle dynamic collision response, such as bouncing. Instead, static or RBD objects should be added to the system. This operator modifies the P, Cd, stopped, stuck, sliding, pospath, posuv, and posprim attributes.

3.12 POP Collision Detect dynamics node

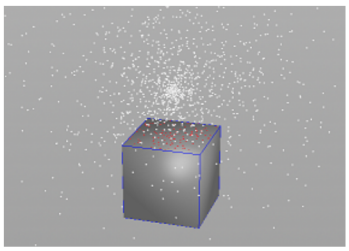

Fig. 3.1 Output of a POP Collision Detect dynamics node

A POP node that detects and reacts to collisions. The POP Collision Detect node finds collisions between *particles and geometry*. It stores the resulting collision information in a set of hit attributes.

3.13 POP ATTRIBUTE FROM VOLUME DYNAMICS NODE

A POP node that copies volume values into a particle attribute. This can then, be used to detect when a particle enters or leaves other objects in the scene. We can use the **Volume tools** to create a fog volume for the external objects, and then use this node to create a float attribute marking if the particle is inside or not. The resulting attribute can be then used in VEXpressions to modulate other POPs. This operator modifies the *attribute specified as a parameter*.

3.14 POP COLLISION IGNORE DYNAMICS NODE

A POP node marks particles to ignore implicit collisions. This node marks particles to not engage in implicit collisions performed by the POP Solver. This allows you to *turn off bouncing on a subset of particles*.

3.15 POP Color dynamics node

A POP node that colors particles and its color node colors each particle. This operator modifies the *Alpha and Cd* attributes.

3.16 POP CURVE FORCE DYNAMICS NODE

A POP node that creates forces generated from a curve. The POP Curve Force node applies a force to particles to steer them *along, toward, or around a curve*. This operator modifies the *force attribute*. This node creates forces which can cause the particles to get pushed away from the curve and outside the *Max Influence Radius*. For curves with sharp turns or complex shapes, the *Suction Force* will need to be balanced with the other forces to keep the particles near the curve and under the influence of the *Curve Forces*.

3.17 POP DRAG DYNAMICS NODE

This is a dynamics node. A POP node that applies and defines a drag to particles. This operator modifies the *target v and air resist attributes*.

3.18 POP FAN CONE

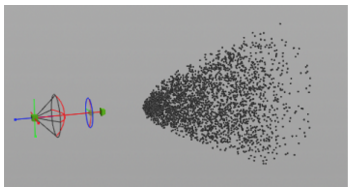

Fig. 3.2 Output of a POP FAN CONE

A dynamics node POP node that applies a conical fan wind to particles. This operator modifies the *target v and air resist attributes*.

3.19 POP FIREWORKS DYNAMICS NODE

A POP node that creates a simple fireworks system. Other than being useful for a fireworks simulation, this provides a quick way to see something pretty, and can be useful as a source for testing a particle chain.

3.20 POP FLOAT BY VOLUMES DYNAMICS NODE

A POP node that floats particles on the surface of a liquid simulation. The Float by Volumes POP is designed to make it easy to float oriented particles on the surface of a liquid simulation for debris or other effects. Because the particles are oriented, geometry instanced onto the particles with the Instance POP will appear to float properly in the liquid. This DOP applies several operators to the particle system,

- Floating underwater particles to the surface, similar to the Force POP.

- Advection of the particles with the velocity field of the liquid simulation to make the particles flow with the liquid, similar to the `Advect By Volumes POP`.
- Align the specified particle axis with the surface normal of the liquid to make particles float upright, similar to the Look At POP.
- Spin the particles by the vorticity of the velocity field, similar to the Spin by Volumes POP.
- Often the liquid simulation will be simulated as a separate pass and the velocity fields read off disk. However, the particles can be live-linked to an existing simulation.

3.21 POP Flock dynamics node

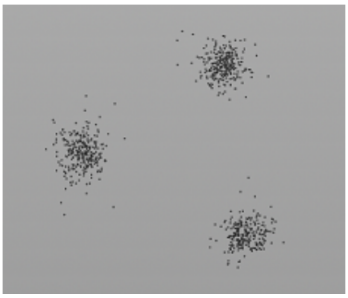

Fig. 3.3 Output of a `POP Flock dynamics node`

A `POP` node that applies a flocking algorithm to particles. It causes particles to flock together. This operator modifies the *force* attribute.

3.22 POP Force dynamics node

A `POP` node that applies forces to particles. It applies a force to particles by updating their force attribute.

3.23 POP GRAINS DYNAMICS NODE

Fig. 3.4 Output of a `POP Force dynamics node`

A `POP` node that applies sand grain interaction to particles. It treats the particles as small spheres and applies interaction between them. This allows for *interpenetration prevention, cohesion, and even explicit constraints* to be enforced. Unlike `POP Interact`, which uses forces to prevent particles from penetrating, the `POP Grains` uses a `Position Based Dynamics` approach to directly move the particles apart. This approach allows for a more stable enforcement of the constraints without the sort of explosions that very high forces would induce. This operator directly modifies the P attribute to move the points and the v attribute to reflect how they moved. Particles under control of POP Grains have the `ispbd` attribute set to 1. This causes them to not perform normal

movement update in the POP Solver, as the actual motion update is done in this node.

3.24 POP GROUP DYNAMICS NODE

A POP node that groups particles. It creates a point group to provide a way to refer to a group of particles later by name.

3.25 POP INSTANCE DYNAMICS NODE

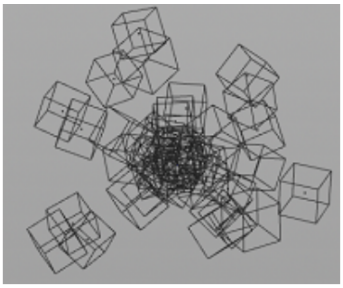

Fig. 3.5 Output of a POP INSTANCE DYNAMICS NODE

A POP node that sets up the instance path for particles. It sets the instance path for particles, allowing them to be rendered with point instancing. You can also use the Instance SOP to apply the instancing in SOPs. The guide geometry for this node is created by the POP Object, which is where one can disable it or change its default color. The *orient attribute* is used to control the orientation of the instance. This operator modifies the *instance path* and *pscale* attributes.

3.26 POP Interact dynamics node

A POP node that applies forces between particles. It applies a forces to particles based on their closest neighbors. This can be used to keep particles apart or to make them match velocities. This operator modifies the *force* attribute.

3.27 POP Kill dynamics node

A POP node that kills particles. The POP Kill node marks particles for deletion. They will then be deleted in the final stage of the POP Solver. This means they will not be visible in the next frame, but they will be available to later nodes , for example, birthing particles. This operator modifies the *dead* attribute.

3.28 POP Limit dynamics node

A POP node that limits particles. The POP Limit node limits particles within a cube in 3d space. This is useful to automatically *cull or bounce particles* that fly too far. This operator modifies the P and v attributes.

3.29 POP Local Force dynamics node

A POP node that applies forces within the particle's frame. It applies a force to a particle. It does this within the particle's own facing, as determined by the *orient* attribute. This operator modifies the force attribute.

3.30 POP Location dynamics node

A POP solver that generates particles at a point. POP Location node generates particles for particle simulations. Additional POP nodes can be wired after it and will implicitly only affect particles created by this node.

3.31 POP Lookat dynamics node

A POP node makes a particle look at a point. It reorients a particle to look at a given point or direction. This operator modifies the *orient, torque, target w, and spin resist* attributes.

3.32 POP Metball Force dynamics node

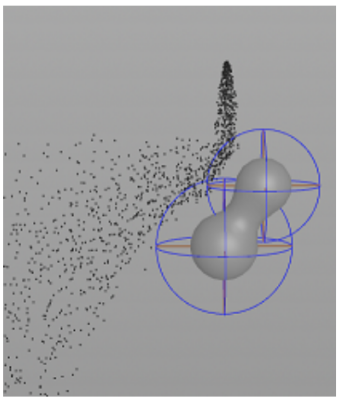

Fig. 3.6 Output of a POP Metball Force dynamics node

A POP node that applies forces according to metaballs. It applies a force to particles according to how close they are to metaballs. The metaballs must be set up with *force* attributes using the Force SOP. Particles within range of the metaballs will be influenced by them, scaled by the weight of the metaball. This operator modifies the *force* attribute.

3.33 POP Object dynamics node

Converts a regular particle system into a dynamic object capable of interacting correctly with other objects in the DOP environment. The result is ready for use by the POP Solver.

3.34 POP Property dynamics node

A POP node that sets various common attributes on particles. It adds various common POP attributes to particles. This operator modifies the *p scale, mass, spin shape, bounce, friction, dynamic friction, drag, drag center, drag shape, drag exp* and *cling* attributes.

3.35 POP Proximity dynamics node

A POP node that sets attributes based on nearby particles. This operator sets attributes on the input particles that record which other particles are closest and how far away they are. This operator modifies the *nearest, nearestdist,* and *numproximity* attributes.

3.36 POP Replicate dynamics node

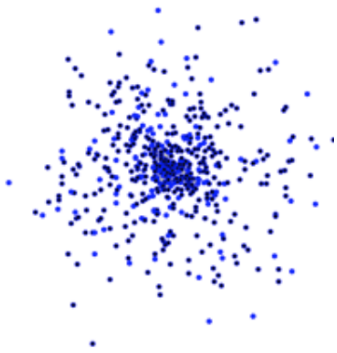

Fig. 3.7 Output of a POP Proximity dynamics node

A POP node that generates particles from incoming particles. POP Replicate node designed generate particles for a POP network. It does this by replicating given particles in the input stream. It uses the up-stream set of particles to generate a new stream of particles by replicating the source particles.

3.37 POP Soft Limit dynamics node

A POP node that creates a spongy boundary. It slows particles down when they hit the boundary, giving a less jerky response than a limit or collision. This operator modifies the *force* attribute.

3.38 POP Source dynamics node

A POP node that generates particles from geometry. POP Source is a node that generates particles from geometry, often a referenced SOP network.

3.39 POP Speed Limit dynamics node

A POP node that sets the speed limits for particles. It adds attributes to specify the maximum speed that particles can move. This clamping is done after all the forces are integrated, so can be useful to tame unruly particles. If the attributes are negative, no clamping is done. This operator modifies the speed min, speed max, spin min, and spin max attributes.

3.40 POP Spin dynamics node

A POP node that sets the spin of particles. It directly sets the spin, or angular velocity, of particles. The attribute to store angular velocity, ω, is measured in radians per second, which differs from the degrees per second used in this interface. To convert between the two, the *radians()* and *degrees()* VEX functions can be used. For convenience, the local variable *old spin speed* is defined which stores the particle's previous angular velocity measured in degrees per second. This operator modifies the ω attribute. It ensures the orient attribute exists.

3.41 POP Spin by Volumes dynamics node

A POP node that uses the vorticity of velocity volumes to spin particles. The Spin by Volumes POP is designed to make it easy to add spin to a particle system from the vorticity of a fluid simulation. Often the fluid simulation will be simulated as a separate pass and the velocity fields read off disk. However, the particles can be live-linked to an existing simulation. This operator modifies the *torque, ω, target w, spin resist, and orient* attributes.

3.42 POP Sprite dynamics node

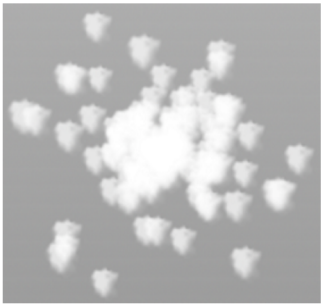

Fig. 3.8 Output of a POP Sprite dynamics node

A POP node that sets the sprite display for particles. It adds attributes to specify the sprite display of particles. This operator modifies the *sprite path, sprite uv, sprite rot, and sprite scale* attributes.

3.43 POP Steer Align dynamics node

Applies force to agents/particles to align them with neighbors. This is a *crowd behavior* node, but one can also use it on regular particles (see the *Output* attribute parameter). One can also choose whether each agent is **aware** of all the agents within a certain radius, or only the agents in its *field of view*.

3.44 POP Steer Avoid 2.0 dynamics node

Applies anticipatory avoidance force to agents/particles to avoid potential future collisions with other agents/particles. This is a **crowd behavior** node, but you can also use it on regular particles (see the *Output* attribute parameter).

3.45 POP Steer Cohesion dynamics node

Applies forces to agents/particles to bring them closer to their neighbors. This is a `crowd behavior` node, but you can also use it on regular particles (see the *Output* attribute parameter).

3.46 POP Steer Custom dynamics node

The `VOP` network ends in a `Add Steer Force VOP` with parameters/inputs for *Only Add(boolean), Steer Force (float),* and *Weight (float)*. When *Only Add* is true, the steer force value is added to the existing steer force. When *Only Add* is false, the steer force value is multiplied by the weight and normalized by the `crowd solver` (see crowd force weights).

3.47 POP Steer Obstacle dynamics node

Applies force to agents/particles to avoid potential collisions with static objects. This is a crowd behavior node, but it can also be used on regular particles (see the Output attribute parameter). It sends out a certain number of rays in a cone along its forward direction, and tries to avoid collisions based on the closest ray hit. It looks for the optimal path based on the least obstructed ray, and uses environment normal directions as the braking direction. See crowd obstacles for how to set up obstacles using shelf tools.

3.48 POP Steer Path dynamics node

Applies force to agents/particles according to directions from a path curve. This is a crowd behavior node, but it can also be used on regular particles (see the *Output* attribute parameter). Once agents reach the end of the curve, they will continue to alternately move along the direction at the last point and be pulled back by the *Path variance*, making them bounce around the end of the curve. You can avoid this by using a *bounding box trigger* at the end of the curve to put agents that reach the end *into a new state*.

3.49 POP Steer Seek dynamics node

Applies force to agents/particles to move them toward a target position. This is a *crowd behavior* node, but you can also use it on regular particles (see the *Output attribute* parameter). To make agents chase the nearest goal: Use the *Attribute Transfer geometry node* to set up attributes on the goal points that contain the particle ID of the closest agent. Then set **Match method** to ***Point per particle*** and set **Goal ID** to the attribute you created with Attribute Transfer.

3.50 POP Steer Separate dynamics node

Apply force to agents/particles to move them apart from each other. This is a **crowd behavior** node, but you can also use it on regular particles (see the *Output attribute* parameter). POP Steer Solver 2.0 dynamics node Used internally in the crowd solver to integrate **steering forces**. This operator updates the force point attribute using the steer force, steer weight, and max force attributes.

3.51 POP Steer Turn Constraint dynamics node

Constrains agent velocity to only go in a direction within a certain angle range of its current heading, to prevent agents from floating backward. This node is used internally in the **Crowd solver**. It relies on the crowd system's steer force attribute, so unlike other POP Steer nodes it is not also useful for plain particles. It modifies agent/particle speed based on its orientation and target velocity direction. It will be scaled based on agent orientation/facing direction and velocity direction. Value of zero will result in speed unaffected, where at **Max Angle** the speed will be zero. Any values in between will be linearly scaled 0 to 1.

3.52 POP Steer Wander dynamics node

Apply forces to agents/particles to create a random motion. This is a **crowd behavior** node, but you can also use it on regular particles (see the **Output attribute** parameter). This node has a **Force** parameter, which controls the base speed of the wander, and **Amplitude**, which controls the size of the changes to the base speed. So, for example, high **Force** and low **Amplitude** would give fast motion with small deviations, while low **Force** and high **Amplitude** would give overall slower motion but with big swings in direction and speed.

3.53 POP Stream dynamics node

A POP node that creates a new stream of particles. It defines a new stream of particles. This lets you wire nodes, after which that will implicitly operate on the subset of particles that it defines. This operator modifies the force attribute.

3.54 POP Torque dynamics node

A POP node that applies torque to particles, causing them to spin. It applies a torque to particles that causes them to spin-up. This operator modifies the *torque* attribute.

3.55 POP Velocity dynamics node

A POP node that directly changes the velocity of particles. It directly alters the velocity of particles. It is important to note that zeroing the velocity will not stop the particle from moving! During the integration, the particle may receive forces giving it a non-zero velocity during the timestep. The *force and target v* attributes need to also be zeroed to make the particle stop. By default, the velocity parameter is multiplied by the scale parameter, and this value is written directly to the velocity attribute. The VEXpression can be used to make the outgoing velocity parameter be a function of the particle's original velocity using the @v variable. This operator modifies the v attribute.

3.56 POP VOP dynamics node

Runs CVEX on a particle system. The POP VOP operation runs CVEX over a set of particles. The CVEX shader can be defined as a .vfl file, a SHOP, or by building a CVEX VOP

network inside this node. The last is the most straightforward approach. Binding to the CVEX script controls what attributes are passed to which parameters of the script. The default AutoBind will use the name of the attributes to determine which parameter they should override. If that parameter is marked as exported in the CVEX script, the attribute will be written to and/or created. Otherwise, the attribute is just read to get the parameter's values. Optionally, you can also bind read-only field parameters. These will be sampled at the attribute's location in space and passed to CVEX.

3.57 POP Wind dynamics node

A POP node that applies wind to particles. It applies wind to particles by updating their *target v and air resist* attributes. Unlike POP Force, which continuously accelerates particles, POP Wind, instead, acts as a drag pulling particles to the ambient wind speed. This operator modifies the *target v, and air resist attributes*.

3.58 GRAVITY FORCE DYNAMICS NODE

The Gravity Force DOP applies forces on objects as if they were inside a gravity field. The amount of force is proportional to the object's mass. For example, if an object has twice the mass, it receives twice the force. However, because an object of twice the mass requires twice the force to experience the same net acceleration, the Gravity Force can be thought of as a raw acceleration. You can *add noise* to the force applied by this DOP by connecting a Noise DOP to the second input of this node, which adds the noise as subdata of the force data.

Chapter 4

Conclusion

Houdini provides several groups of light sources: Light linking and shadow linking. We can specify categories on an object using the parameter editor (Render tab, Shading sub-tab, Categories parameter). Objects can belong to more than one category – separate multiple categories with commas and/or spaces. Categories are similar to tags in Web applications; you can "tag" lights and objects, and use the unions and intersections of tags to control light linking. Category expressions are one or more category names. When specifying multiple name, join them with the (and) or (or) symbols. For example, keylight distant would specify all objects which have either the keylight or distant categories, while keylight distant would specify only objects that have both keylight and distant categories.

In general, rendering in Houdini requires a camera defining the viewpoint to render from, lights to illuminate the scene, and a render node representing the renderer and render settings to use. (Some render preview techniques allow you to render without the camera, lights, or a render node. In some cases Houdini will simply use the current view, a headlight, and default render settings to do preview renders.)

By default, Houdini uses the mantra rendering program to convert Houdini scenes into images. You can use other renderers such as Render man and AIR by using a the render node for the alternate renderer (and switching to shaders compatible with that renderer) instead of using the default mantra render node. Mantra has several different rendering engines, which use different algorithms to render the scene. You should use the physically based rendering engine unless you have a good reason to use another engine.

There are two kinds of compositing options in Houdini: global preferences and defaults which affect all compositing you do in Houdini, and project settings which affect a particular .hip file. Global preferences : The preferences let you control memory usage, efficiency, interface, and color behavior in the compositor.

Project settings : The settings in this window control the defaults for various COP options in this scene file. Changing a project setting can affect existing nodes that use that setting.

In general particle simulation can be used immensely and this can be used as a basis for future developments.

Chapter 5

Future Works

In this paper, we have determined the process behind creating a particle simulation demonstration in houdiniFX. The idea was inspired from an advert on television.

This project has not been done with visual clarity, there needs to be a lot of improvement in the overall outlook of the project. Instead of having a sub network, a digital asset with controllable features could be developed.

The path particle simulation could have been developed and rendered with better quality.

References

Sidefx.com. (2016). Dynamics nodes. [online] Available at: http://www.sidefx.com/docs/houdini/nodes/dop/_index [Accessed 4 Aug. 2016].

YouTube. (2016). Tutorial: Crowd Simulations in Houdini 15. [online]
Available at: https://www.youtube.com/watch?v=KPewfLuEy7s [Accessed 10 Aug. 2016].

YouTube. (2016). Houdini Crowd Introduction - Intro. [online]
Available at: https://www.youtube.com/watch?v=D3P0XYr16SY [Accessed 5 Jul. 2016].

YouTube. (2016). Houdini FLIP fluid Particle simulation. [online]
Available at: https://www.youtube.com/watch?v=d8_0WFOMFUg [Accessed 9 Jul. 2016]

YouTube. (2016). Introduction to Houdini 13 - Particles Tutorial. [online]
Available at: https://www.youtube.com/watch?v=aYd_QEbsLNY [Accessed 13 Aug. 2016].

YouTube. (2016). Introduction to Houdini 13 - Particles Tutorial. [online]
Available at: https://www.youtube.com/watch?v=aYd_QEbsLNY [Accessed 6 Aug. 2016].