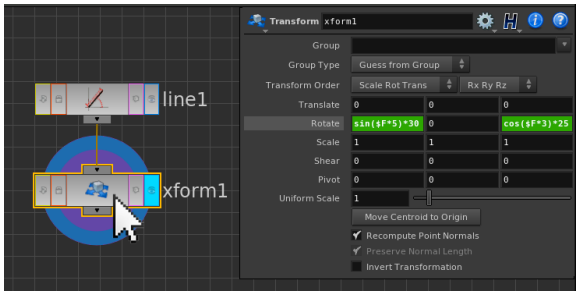
A SIMPLE PARTICLE WAND

Particles are a very versatile tool in Houdini. They can in essence be sculpted to create any shape and form required. Since Houdini 13, particles are now part of the Dynamics (DOPs) Level of Houdini, meaning that they will automatically interact with other dynamic solvers such as RBDs, Fluids, Volumetrics, Cloth etc. This lecture will however look at particle effects as a toolset in their own right.

In a **new Houdini scene**, create a **Geometry Object**, and inside at SOP Level, **replace** the **default File SOP** with a **Line SOP**. In the **parameters** for the **Line SOP** specify:

Length	**5**
Points	**20**

This will create a simple piece of geometry to emit particles from.

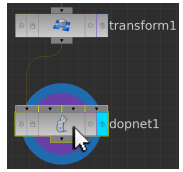


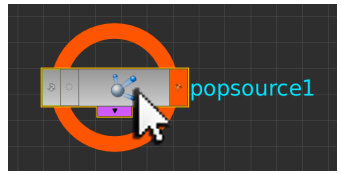
Append a **Transform SOP** to the Line SOP, and in its **parameters** specify:

Rotate	**sin($F*5)*30**	**0**	**cos($F*3)*25**

This will animate the line creating a simple wand movement to it.

To the **output** of the **Transform SOP** append a **DOP Network** and **double LMB** on it to **go inside it**.



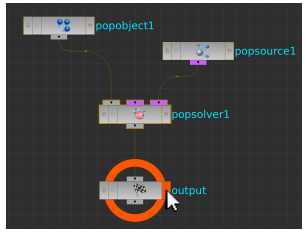


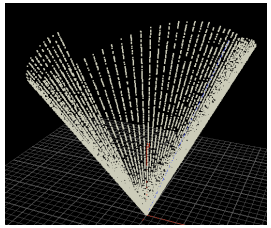
At **DOP Level**, create a **POP Source DOP**. This operator can read in geometry from SOPs and birth particles from them. In the **parameters** for the **POP Source DOP** specify:

Source >	
Geometry Source	**Use First Context Geometry**

Activating this option source the incoming geometry from the **first input** of the **DOP Network** node.

Alongside the POP Source DOP, **create** a **POP Object DOP** and a **POP Solver DOP**.

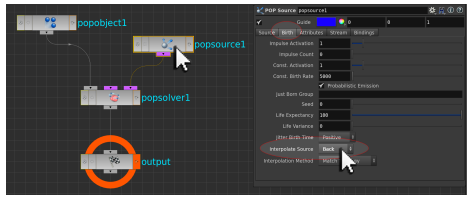


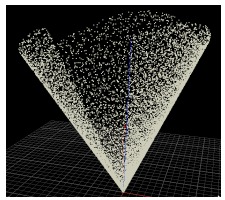


Wire the **POP Object DOP** as the **first input** of the **POP Solver**, and the **POP Source DOP** as its **third input**. This will generate and solve particles from the geometry being read in by the POP Source DOP. Wire the **output** of the **POP Solver** into the **Output DOP** and **activate** its **orange Display Flag**. When **PLAY** is pressed, a particle fan is created.

In the **parameters** for the **POP Source DOP** specify:

Birth >
Interpolate Source **Back**



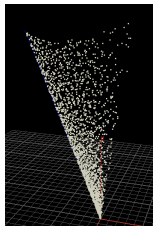


Reset the timeline to Frame 1 and press **PLAY** again. This time the particles are emitted more randomly from the edge of the line.

In the **parameters** for the **POP Source DOP** specify:

Birth >
Life Expectancy **0.75**
Life Variance **0.15**

When **PLAY** is pressed again, the particles die off after 0.75 seconds (with an additional life variance ranging between +0.15 and -0.15 seconds), creating a more ephemeral aesthetic.



IMPULSE ACTIVATION VERSES CONSTANT ACTIVATION

Examination of the Birth section of the POP Source DOP's parameters reveals two methods for birthing particles from incoming geometry. These are **Impulse Activation** and **Constant Activation**. By default Constant Activation is specified with a Constant Birth Rate of 5000 particles. **Constant Activation** works over **time**, meaning by default 5000 particles per second are born. **Impulse Activation** works each time the node cooks, meaning that an **Impulse Count** value will **birth** that many **particles each simulation frame**.

NOTE: The Constant and Impulse Activation parameters work as on/off switches. When these are set to a value of 0 no particle activation will occur. When set to a value of 1, particles will be born as per the Birth Rate value.

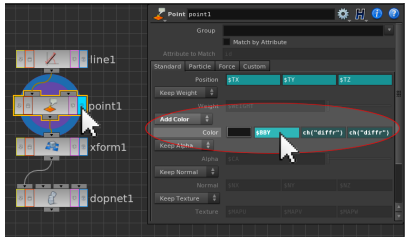
SCULPTING PARTICLES

Creating visually rich particle systems is a combination of POP Operators and any incoming geometry behavior. Adding **custom attributes** to incoming geometry can dramatically improve particle aesthetics above and beyond what the POP operators can do by themselves. The art of particle work is therefore the understanding of how incoming geometry can be used to drive particle systems in bespoke ways.

To the **output** of the **Line SOP append** a **Point SOP**. This can be used to add a black and white ramp across the line, so it is white at its tip, and black at its base. In the **parameters** for the **Point SOP** specify:

Add Color
Color **$BBY** **ch("diffr")** **ch("diffr")**

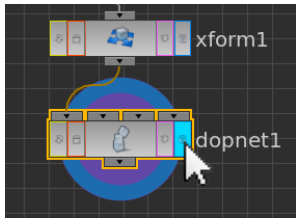
NOTE: Channel referencing can be activated to drive the green and blue channels from the $BBY expression of the red channel.

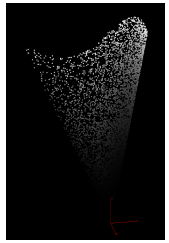


The visual end result of the **$BBY expression** is a **linear ramp** on the line. The colour of the ramp can be however be biased across the surface by wrapping the $BBY expression within a **Power Function**. This in essence bends the linear ramp to create a curve-based ramp instead. In the **parameters** of the **Point SOP**, modify the **red channel expression** to:

Color	**pow($BBY,5)**	**ch("diffr")**	**ch("diffr")**

This will bias the white region of the ramp more to the tip of the wand. This black and white ramp can be used to procedurally determine where on the line geometry particles should be born. **A custom parameter slider** can also be **created** on the **Point SOP** to procedurally control the numeric value specified in the Power Function, allowing it to be animated if necessary.





When the **Display and Render Flag** are reactivated on the **DOP Network**, and **PLAY** is pressed, the effect of the ramp can be seen on the particle colour. **See file H15_particle_wand_stage1.hipnc**

PASSING CUSTOM ATTRIBUTES INTO POPS

Custom attributes on geometry will by default be passed into a DOP Particle System. In this example, the curved ramp can be stored as a custom attribute and then passed into DOPS to control the emission of particles only towards the tip of the wand. This will also free up the colour of the particles allowing for another colour aesthetic to be assigned.

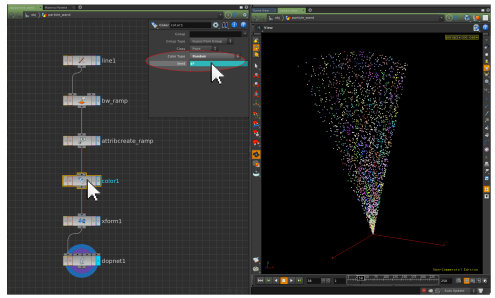
To the **output** of the ramp **Point SOP**, append an **Attribute Create SOP**. In the **parameters** for the **Attribute Create SOP** specify:

Name	**ramp**			
Value	**$CR**	**0**	**0**	**0**

GENERATING RANDOM PARTICLE COLOUR

Now that the ramp colour is stored as a custom attribute, new colour information can be assigned to the line geometry. **Append** to the **Attribute Create SOP** a **Color SOP**. In its **parameters** specify:

Color Type	**Random**
Seed	**$F**



Now when **PLAY** is pressed, the emitting particles are randomly coloured each frame.

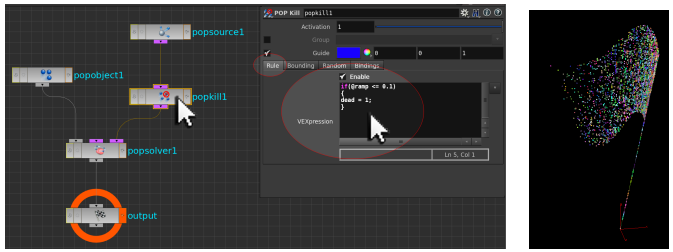
CONTROLING PARTICLE BEHAVIOUR USING CUSTOM ATTRIBUTES

Return back **inside** the **DOP Network**, and to the output of the **POP Source DOP** append a **POP Kill DOP**. The POP Kill DOP can **kill off particles randomly**, by using **Bounding Regions**, or by **VEXpressions**. **VEX** is a **higher-level Scripting Language** in Houdini also realized in node form as **VOPS** (VEX Operators). **VEXpressions** are however are short pieces of VEX code entered into nodes in order to set custom behaviors. **VEX Code differs** from **Houdini's Expression Language** in terms of **syntax** and **application**. It is **more verbose** that Houdini's Expression Language; but delivers many similar functions.

In the **Rule parameters** for the **POP Kill DOP** specify:

☑	**Enable**
VEXpression	**if (@ramp <= 0.1)** **{** **dead = 1;** **}**

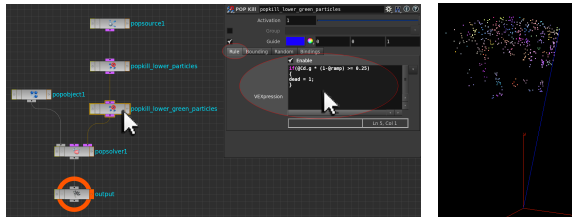
This will kill off particles generated towards the base of the wand, creating a flag of randomly coloured particles.



NOTE: In **VEXpressions**, attributes are called using the **@** symbol rather than $UPPERCASE, and all commands must be ended with a semicolon (**;**).

Append a **second POP Kill DOP** to the **POP network**. In it's **Rule parameters** specify:

☑	**Enable**
VEXpression	**if (@Cd.g * (1 - @ramp) >= 0.25)** **{** **dead = 1;** **}**



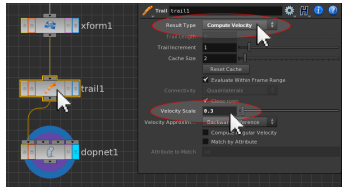
This VEXpression will compare the green value of each particle with its position on the ramp, removing green particles below a certain value. When **PLAY** is pressed, any green particles are thinned out towards the bottom area of the flag. **See file H13_particle_wand_stage2.hipnc**

ASSIGNING GEOMETRY VELOCITY TO THE PARTICLES

By default particles will inherit any velocity information found on the incoming geometry. As velocity is not normally present on geometry, it needs to be calculated and activated as a custom attribute. This can be done using a **Trail SOP**.

Return back to **Geometry Level,** and to the **Transform SOP append** a **Trail SOP**. In its **parameters** specify:

Result Type	**Compute Velocity**
Velocity Scale	**0.3**



This will compute the velocity but also scale its values down so it does not adversely affect the overall look of the particle system. When **PLAY** is pressed again, the movement of the particles is more fluid and gelatinous as a result of the inherited velocity. This velocity movement can also be added to further by **specifying** on the **POP Source DOP**:

Attributes >

Initial Velocity	**Add to inherited velocity**		
Variance	**0.2**	**0.2**	**0.2**

This will give each particle movement based upon the movement of the wand, plus and additional random direction variance, resulting in a more naturalized aesthetic.

GROUPING PARTICLES

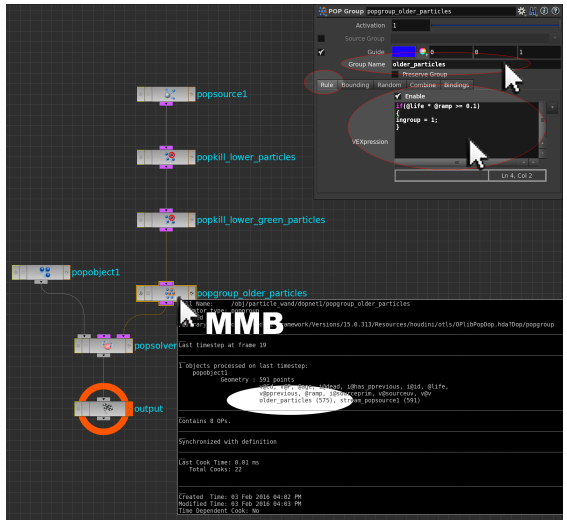
POPS have a number of **existing attributes** that can be called on its operators. One such variable is **@life**. This returns the percentage of lifespan of each particle as a value between 0 and 1. Append a **POP Group DOP** to the particle network, and in its **parameters** specify:

Group Name **older_particles**

Rule >

☑	**Enable**
VEXpression	**if (@life * @ramp >= 0.1)**
	{
	ingroup = 1;
	}

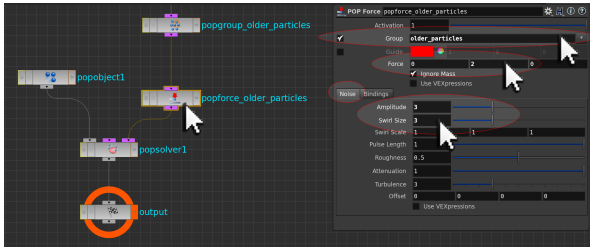
This will group any particles whose **$LIFE** value has exceeded **0.1**, and whose position is higher up the particle wand.



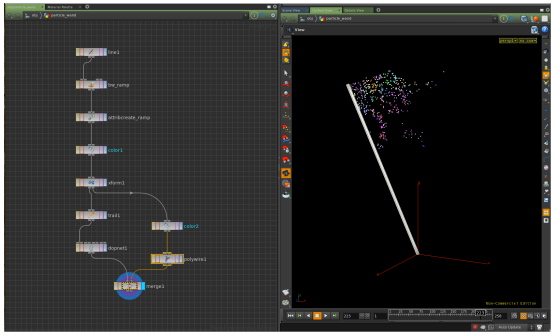
This grouping can be verified by **MMB** on the **POP Group DOP** after **PLAY** has been pressed. This grouping can be used to drive secondary particle animation effects on older particles before they die off.

To the **output** of the **POP Group DOP** append a **POP Force DOP**. In its **parameters** specify:

☑ **Group**	**group_older_particles**		
Force	**0**	**2**	**0**
Noise >			
Amplitude	**3**		
Swirl Size	**3**		



When **PLAY** is pressed again, the older particles have additional oscillation to their movement.



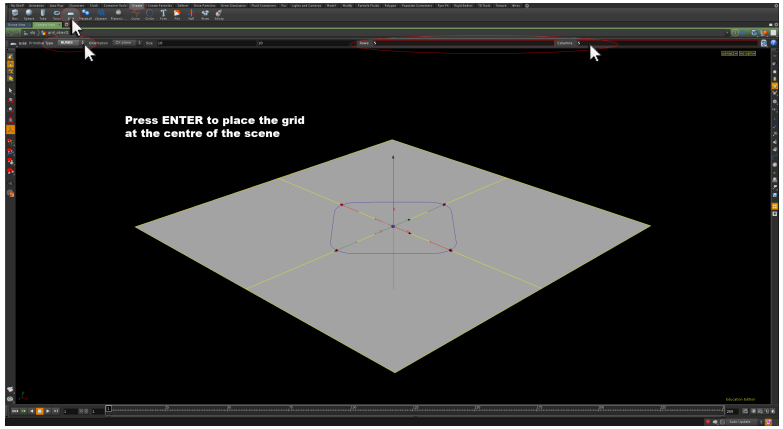
As a final step, a **Color SOP** and **Polywire SOP** can be used to create a piece of wand geometry from the original animated line, and merged with the final wand particle system.

See file H15_particle_wand_end.hipnc

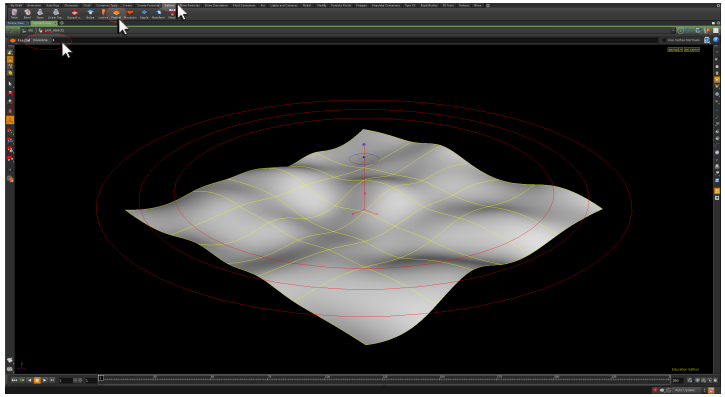
PARTICLES AS GEOMETRY MODIFIERS

This Ice-Breaker tutorial is based upon a classic old school Houdini example. In a **new Houdini Scene**, **maximise** the **Viewer Pane** (**Ctrl + b**), and using the **Create Shelf**, **create** a **Grid Object** and press **ENTER** with the **mouse over the Viewer** to place it at the centre of the scene. From the **Parameter** listing at the **top** of the **Viewer Pane** specify:

Primitive Type	**NURBS**	
Size	**10**	**10**
Rows	**5**	
Columns	**5**	



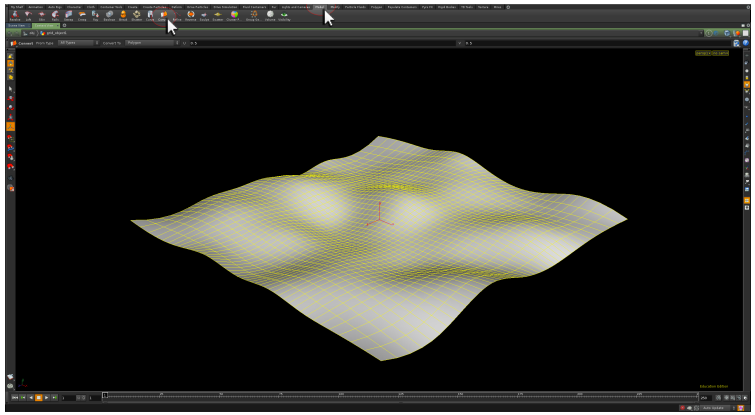
From the **Deform Shelf** activate a **Fractal Operation**. This will automatically append a Fractal SOP to the Grid SOP adding bumps to the grid creating an undulating terrain.



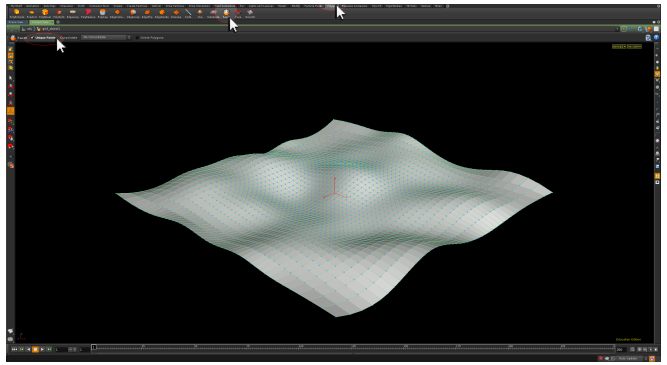
Using the **Parameter listing** located at the **top of the Viewer** Pane specify:

Divisions **1**

From the **Model Shelf**, activate a **Convert Operation**. This will turn the grid into polygons.



From the **Polygon Shelf**, activate a **Facet Operation**, and from the **Viewer** Pane **Parameter listing** activate the **Unique Points** tick box. This will cause each face of the grid surface to be calculated individually allowing the grid to be broken apart.



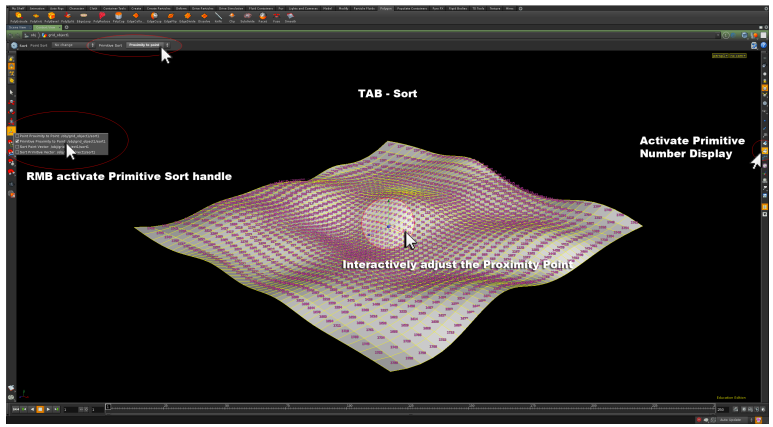
Using the **TAB Menu system** activate a **Sort SOP**. From the **Viewer** Pane **Parameter** listing specify:

Primitive Sort **Proximity to Point**

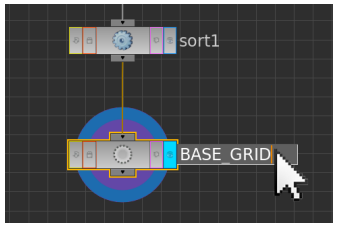
This will re-order the primitive numbers relative to a position in space. By default this point position is 0, 0, 0; however this can be modified interactively by **RMB** on the **Show Handle button** found on the **left side Viewer Stow Bar**, and choosing the **Primitive Proximity to Point** option to activate an **interactive handle** for sorting the primitive numbering.

Activating the **Display Primitive Numbers** from the **right hand side stow bar** of the **Viewer** can also aid interactive positioning of the point to sort by.

LMB adjust the **proximity position** of the **Sort SOP** in the Viewer, so it is slightly away from the centre of the grid. This will create a radial distribution of the primitive numbers relative to the position specified.



In the **Network Editor**, append a **Null SOP** to the end of the **Sort SOP** and rename it to **BASE_GRID**.

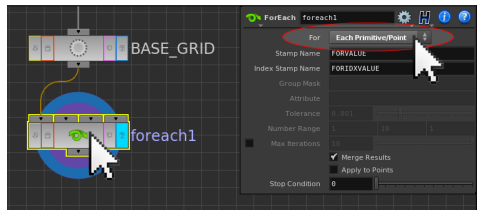


THE FOR EACH SOP

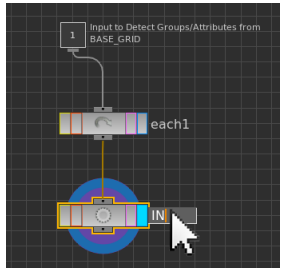
In order for the Ice Breaker effect to work, particles must be born at the centre of each primitive face. Prior to Houdini 13, such functionality was automatically given when initially sourcing geometry to birth particles from. Since Houdini 13, this automatic functionality has been removed. Creating points from the centre of each primitive face can however be achieved using a **For Each SOP**. These points can then be used to birth particles from.

To the output of the **BASE_GRID** Null SOP, append a **For Each Subnetwork SOP**. In it's **parameters** specify:

For Each Primitive/Point



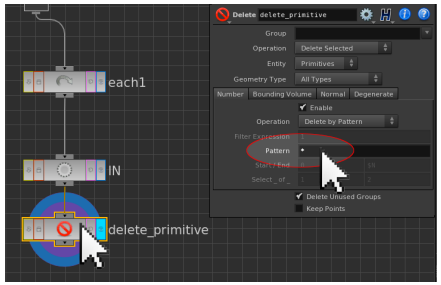
The **For Each SOP** behaves in a **similar** way to the **Point SOP** or **Primitive SOP** in the sense that it will **iterate through each point or primitive in turn and perform an action upon it**. The **For Each SOP** however **can also iterate through numbers**, **groups** and **attributes** as well as points and primitives; and the **actions performed** with each iteration **are not controlled by expressions** (as in the Point SOP or Primitive SOP) **but by SOP network chains** that can be constructed inside the For Each SOP node.



Go **inside** the **For Each SOP**, and append a **Null SOP** to the **Each SOP node**. **Rename** this Null SOP to **IN**. This node will become a reference that can be called later in the network chain.

To the **output** of the **Null SOP** append a **Delete SOP**. In its **parameters** specify:

Pattern *



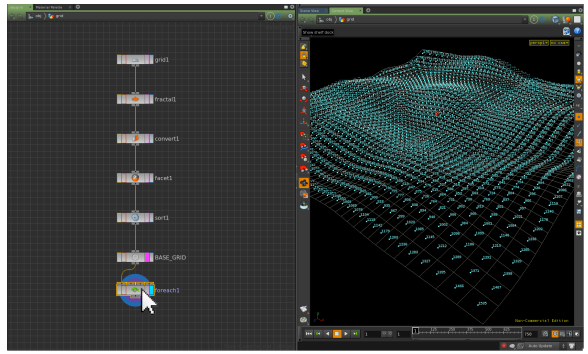
This will iteratively delete each primitive face in turn.

Append an **Add SOP** to the Delete SOP, and under the **Point section** of its **parameters** **activate Point 0** and specify:

☑ **Point 0** **prim("../IN",0,"P",0)** **prim("../IN",0,"P",1)** **prim("../IN",0,"P",2)**

The prim() function will return the centre position of the primitive face being evaluated, locating the resulting point at the primitive centre.

NOTE: Within the For Each SOP, each primitive is being worked upon individually. Therefore each primitive will in turn be evaluated as primitive number 0.



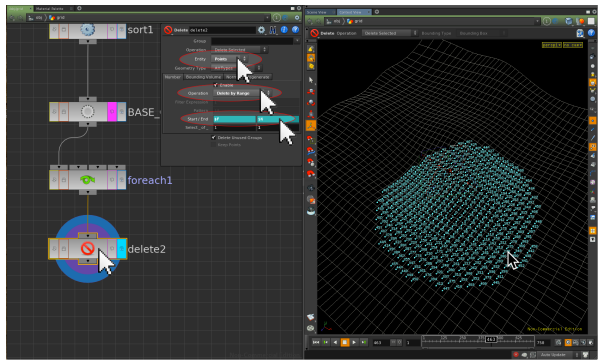
When the top level of the For Each SOP is viewed, a point has been assigned to each primitive face in turn. **See file H15_ParticleIcebreaker_stage1.hipnc**

GENERATING THE PARTICLES

Before particles can be generated from these points, they must be animated using a Delete SOP to ensure that the particles are birthed in the correct order relative to the effect being created. If no animated delete were used, particles would birth across the entirety of the grid. To the **output** of the **For Each SOP** append a **Delete SOP**. In it's **parameters** specify:

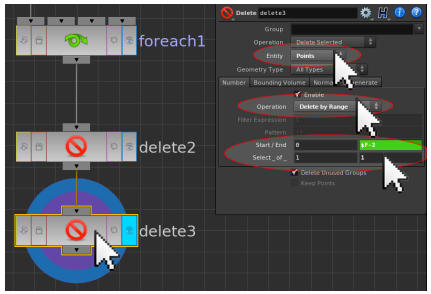
Entity	**Points**	
Number >		
Operation	**Delete by Range**	
Start / End	**$F**	**$N**
Select_of_	**1**	**1**

When **PLAY** is pressed, the primitive centre points grow radially over time.



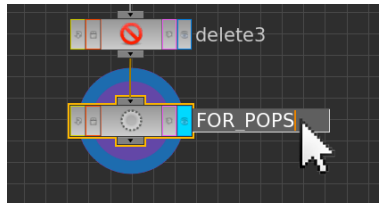
Append a **second Delete SOP**, and this time specify in its **parameters**:

Entity	**Points**	
Number >		
Operation	**Delete by Range**	
Start / End	**0**	**$F-2**
Select_of_	**1**	**1**

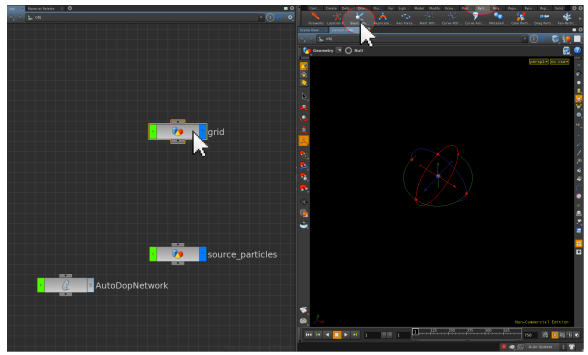


This will delete growing radial points just after they appear, creating the effect of a single point radially dancing over the grid.

As a final step, append a **Null SOP** to the network chain, and **rename** it to **FOR_POPS**.

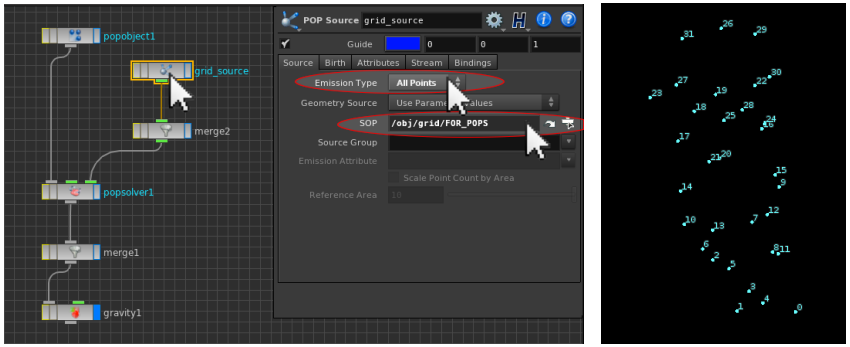


Return to **Object Level**, and with the **Grid Object selected**, **activate** the **Source Particle Emitter** button from the **Particles Shelf**. This will configure an **AutoDopNetwork** creating a particle system from the grid points, as well as creating a **source_particles object** reading the particles back out from the DOP Level.



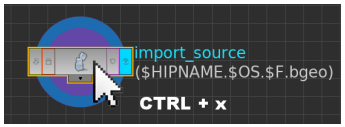
Inside the **AutoDopNetwork**, locate the **POP Source DOP** reading in the grid points, and in it's parameters specify:

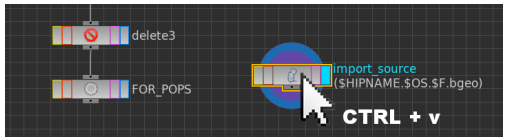
Emission Type	**All Points**
SOP	**/obj/grid/FOR_POPS**



This will explicitly call the FOR_POPS Null SOP as the geometry source and birth particles from its animated points. When **PLAY** is pressed, particles birth from the animated points of the grid.

At **Object Level**, go inside the **source_particles object** and **Cut** (**CTRL + x**) the **import_source SOP** reading in the particles.





Go back inside the grid object, and Paste (**CTRL + v**) the import_source SOP into the grid network. This will allow the imported particles to modify the BASE_GRID geometry.

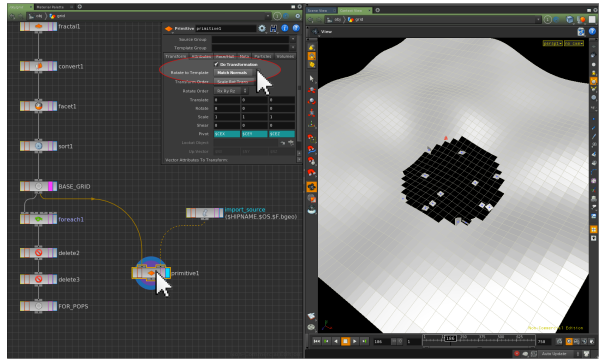
In a **new part** of the **Network Editor**, create a **Primitive SOP** wiring the **output** of the **BASE_GRID Null SOP** as its **first input**, and the **import_source SOP** as it's **second input**.

In the **parameters** for the **Primtive SOP** specify:

Transform >

☑ **Do Transformation**

Rotate to Template **Match Normals**



When **PLAY** is pressed, the individual primitive faces of the **BASE_GRID** are **copied** onto the **imported particles**, causing them to radially break away and fall from the remaining grid. See file **H13_ParticleIcebreaker_stage2.hipnc**

REFINING THE EFFECT

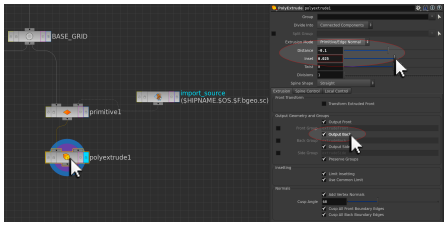
While this configuration establishes the basic effect, the associated networks can be further refined to improve it. To the **output** of the **Primitive SOP**, append a **PolyExtrude SOP**. In it's **parameters** specify:

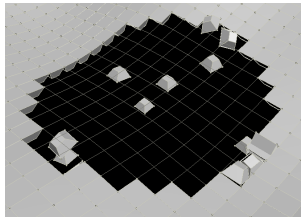
Distance **-0.1**

Inset **0.025**

Extrusion >

☑ **Output Back**





This will create chunks from the primitive faces falling away, allowing for the visual behavior of the source particles to be more easily determined.

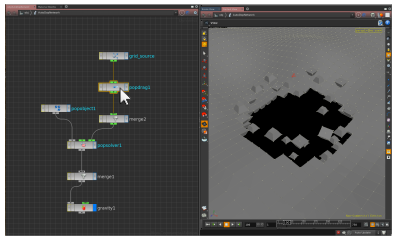
NOTE: The **PolyExtrude SOP** have been **revised** for **H15**. The **Extrusion Transform Controls** (both **Local** and **Global**) can still be **accessed** by **activating** the **Extrusion > Transform Extruded Front** tickbox.

Return back inside the **AutoDopNetwork**, and in the **parameters** for the **POP Source DOP** specify:

Birth >			
Impulse Activation	**$SF <= 150**		
Attributes >			
Initial Velocity	**Add to inherited Velocity**		
Velocity	**0**	**5**	**0**
Variance	**0.1**	**0.1**	**0.1**

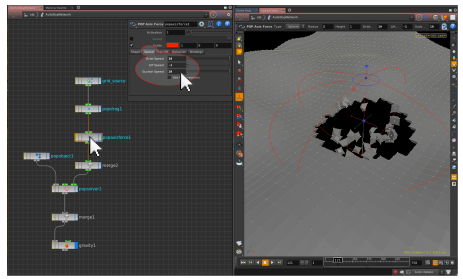
This will turn off the ice breaker effect after 150 frames, as well as giving the particles more initial lift when they are birthed.

Append to the POP Source DOP, **a POP Drag DOP**. This will add drag to the effect causing less of an initial lift to the particles.



Append to the POP Drag DOP, a **POP Axis Force DOP**. This DOP can be used to create more interesting particle movement. In the **parameters** for the POP Axis Force DOP specify:

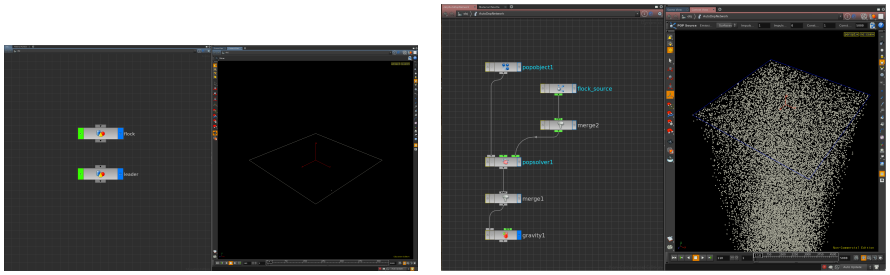
Shape >	
Center	(interactively position the centre over the icebreaker hole)
Radius	**2**
Speed >	
Orbit Speed	**10**
Lift Speed	**-1**
Suction Speed	**10**



When **PLAY** is pressed, the particles swirl towards the centre of the Axis Force. **See file H15_ParticleIcebreaker.hipnc**

SIMPLE FLOCKING

Particles can also be used for mimicking the behavior of real world flocking phenomena. Flocking could be used as a way to create procedural traffic systems, insect or bird swarming, or teams of players in a football simulation for example. Open the scene **H15_flock_begin.hipnc**. This scene contains two objects; one called **flock**, one called **leader**. At present the flock object is a simple grid that will be used to generate particles that will flock towards the leader object (which contains an animated point).

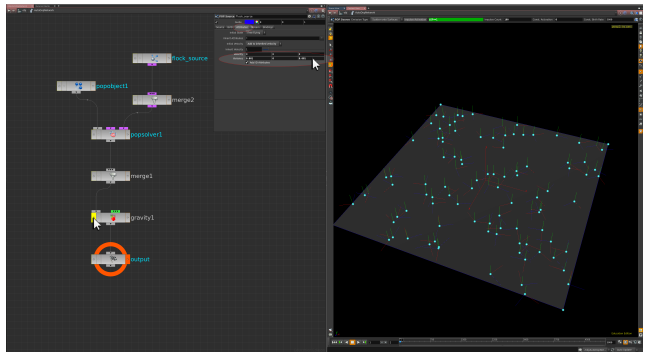


Maximise the **Viewer** (**Ctrl + b**), and **activate** the **shelves**. From the **Particles Shelf LMB** the **Source Particle Emitter** button. When prompted, **select the grid** and press **ENTER**. This will create a DOP Network reading in the grid as the birth source for a particle system. When **PLAY** is pressed, the particles are scattered across the grid and rain downwards.

In the **parameters** for the **POP Source DOP** inside the **AutoDopNetwork** specify:

Birth >			
Impulse Activation	**$SF==1**		
Impulse Count	**100**		
Constant Activation	**0**		
Attributes >			
Variance	**0.001**	**0**	**0.001**

Also **bypass** the **Gravity DOP** to turn off its influence. When **PLAY** is pressed, the particles are birthed on frame 1 only, are orientated randomly in the Z and X axis, and remain static on the surface of the grid.



PARTICLE DISPLAY OPTIONS

With the **mouse over** the **Viewer**, press **d** to bring up the **Display Options** and **specify**:

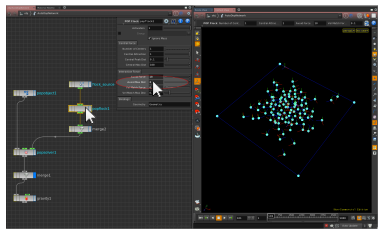
Guides>			
☑	**Particle Gnomon**		
Scale Normal	**3**		
Geometry >			
Point Size	**10**		

This will assign a Gnomon origin display to each individual particle, increase the Viewer display size of the Particle Gnomons, and increase the scale of the particles point display making them easier to see.

CREATING FLOCKING MOVEMENT

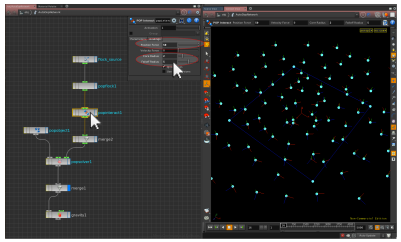
Append to the POP Source DOP, a **POP Flock DOP**. In its **parameters** specify:

Avoid Max Distance	**2**



When **PLAY** is pressed, the particles create a general swarm on the grid, avoiding each other whenever they get too close. This flocking effect can further be enhanced by **appending** a **POP Interact DOP** to the POP Flock DOP. In the **parameters** for the **POP Interact DOP** specify:

Position Force	**50**
Core Radius	**2**
Falloff Radius	**5**



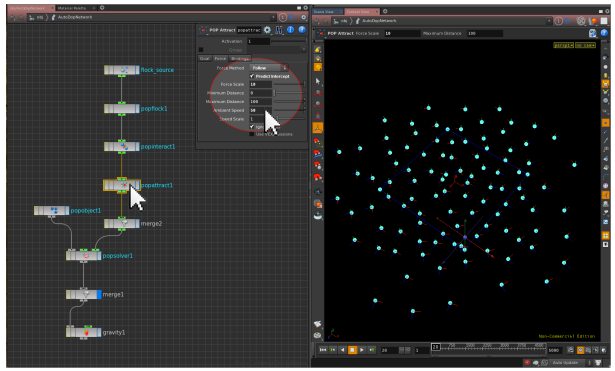
When **PLAY** is pressed, the particles are rapidly repelled from each other.

CREATING A FLOCK LEADER

While the effect of the POP Interact DOP is quite extreme, it can be naturalized by the addition of a leader that the flock will attempt to follow. This can be achieved by **appending** a **POP Attract DOP** to the POP Interact DOP. In the **parameters** for the **POP Attract DOP** specify:

	Goal >		
Goal	**point("/obj/leader",0,"P",0)**	**point("/obj/leader",0,"P",1)**	**point("/obj/leader",0,"P",2)**
	Force >		
	Force Method	**Follow**	
	☑	**Predict Intercept**	
	Force Scale	**10**	
	Ambient Speed	**50**	

This will read in the animated leader point from Object Level as the goal for the flocking particles, and control the speed and force of their movement towards it.



When **PLAY** is pressed, the flock are still repelled from each other; however very quickly start animating towards the animated leader point. **See file H15_flock_stage1.hipnc**

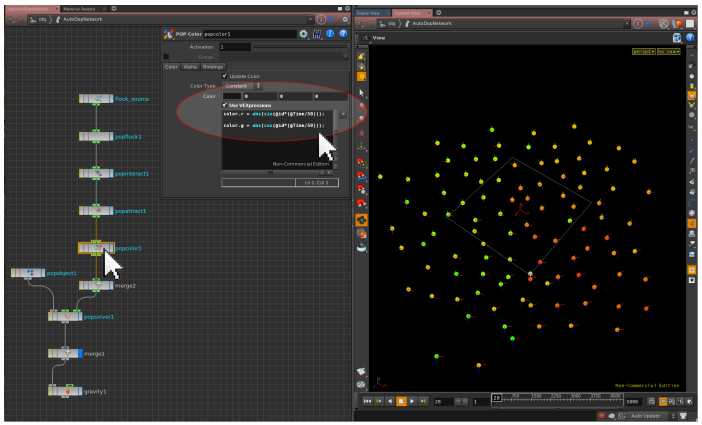
ADDING FLOCK BEHAVIOR

Currently the flocking particles have a swarming movement with no other character to the flocking behavior. Behavior can be added by assigning animated colour to individual particles, and then using this animated colour information to control behavioral aspects of each particle's movement such as speed.

To the **output** of the **POP Attract DOP**, **append** a **POP Color DOP**. In its parameters specify:

Color	**0**	**0**	**0**
☑	**Use VEXpressions**		
	color.r = abs(sin(@id*(@Time/50)));		
	color.g = abs(cos(@id*(@Time/50)));		

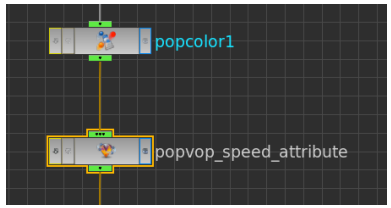
When **PLAY** is pressed, the flocking particles oscillate colour going from green to red to green again on a per particle basis.



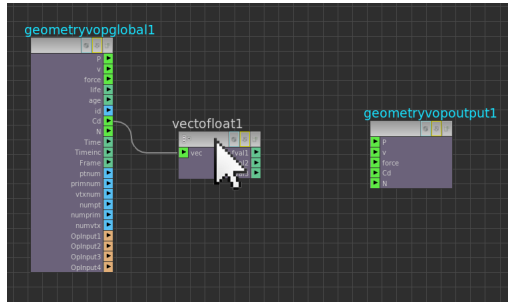
CREATING CUSTOM ATTRIBUTES IN POPS

Custom attributes for POPS can be created in two ways. Firstly a **VEXpression** creating the custom attribute can be assign using a **POP Wrangle DOP**. The second method is to use a **POP VOP DOP** to create an attribute using a **VOP Network**. Both methods will achieve the same end result.

Append to the POP Color DOP a **POP VOP DOP**.

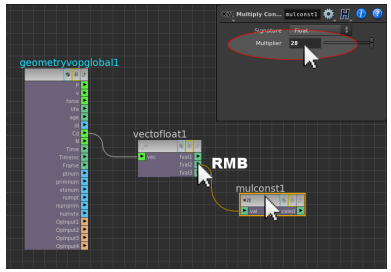


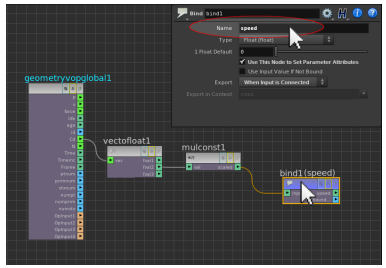
Double LMB on this node to go inside it, and in the **VOP Network** create a **Vector to Float VOP** wiring the output of the Cd (colour) attribute from the Global Parameters VOP as its input.



This will take the colour information from the flocking particles and split it into its 3 components (red, green and blue).

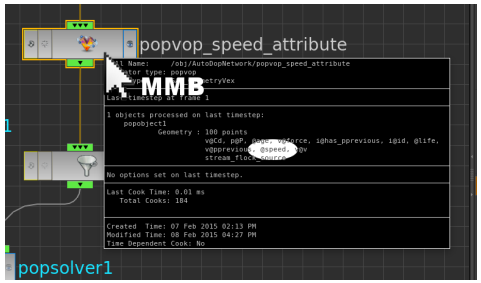
RMB on the **second output** of the **Vector to Float VOP** to append to the green channel a **Multiply Constant VOP**. In it's **parameters** specify a **Multiplier value** of **20**.





RMB on the output of the **Multiply Constant VOP** to create a **Bind Export VOP**. In it's **parameters** specify the **Name** for the custom attribute to export as **speed**.

Back at **DOP Level**, **MMB** on the **POP VOP DOP** to verify that the **speed attribute** has been created.

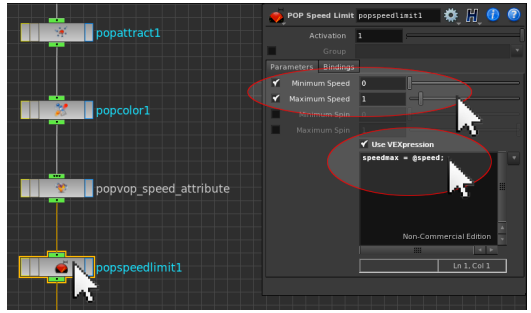


Were this custom attribute created using a POP Wrangle DOP and VEXpressions, the VEXpression code would be:

VEXpression **@speed = @Cd.g * 20;**

Append to the POP VOP DOP, a **POP Speed Limit DOP**. This is a very powerful POP that can literally stop and start particles at will. In the parameters for the POP Speed Limit DOP specify:

☑ **Minimum Speed** **0**
☑ **Maximum Speed** **1**
☑ **Use VEXpression**
speedmax = @speed;



The maximum speed of the particles will now be driven by the green colour information driving the speed attribute.

NOTE: The Maximum Speed parameter must be activated (☑) in order to be overridden by a VEXpression.

When **PLAY** is pressed, the flocking particles chase after the animated leader point when they are green; however lose interest in chasing it when they are red and come to a halt.

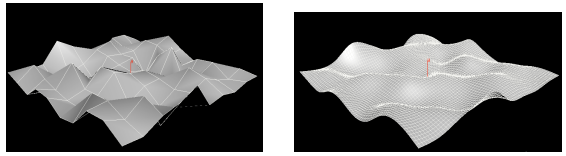
See file H15_flock_stage2.hipnc

ASSIGNING GEOMETRY AND A FLOCKING TERRAIN

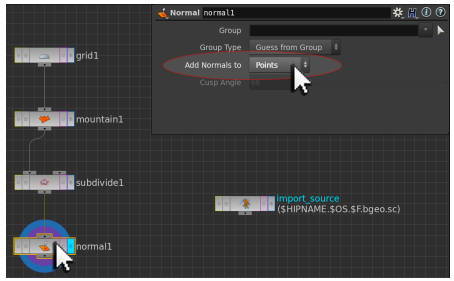
Although the flocking system has been configured on a flat surface, the flocking particles can be made to go up and down over undulating terrain. In the **source_particles object** create a **Grid SOP** and in its **parameters** specify:

Size	**80**	**80**	
Center	**0**	**10**	**0**

This will create a large grid situated above the particle flocking system. Append a **Mountain SOP**, setting a **Height** parameter value of **20**.



To the Mountain SOP **append** a **Subdivide SOP**, specifying a **Depth** parameter value of **3**. This will create a smooth undulating terrain that the particles can be projected onto.

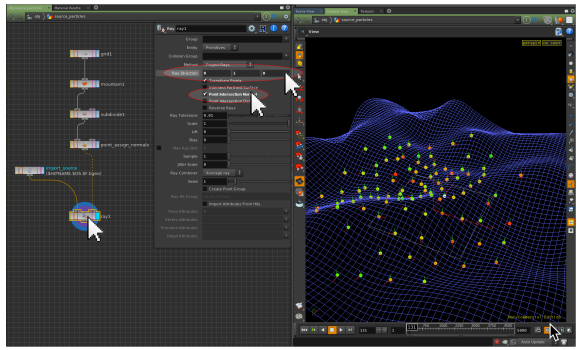


As a final step, **append** a **Normal SOP** and activate the **Add Normals to Points** option. This will ensure any projected particles are orientated onto the surface of the terrain.

THE RAY SOP

In a **new part** of the **Network** Editor create a **Ray SOP**. The Ray SOP can project points or primitives onto geometry specified by its second input. **Wire** the **output** of the **import_source SOP** as its **first input**, and the output of the **terrain Normal SOP** as its **second input**. In the **parameters** of the **Ray SOP** specify:

Ray Direction	**0**	**1**	**0**
☑	**Point Intersection Normal**		



When **PLAY** is pressed, the flocking particles navigate over the terrain.

NOTE: Activating **Point Normal Display** in the Viewer will reveal the new normals assigned to each point based upon the terrain undulations.

RECALCULATING THE DIRECTION OF PARTICLE TRAVEL

As terrain Normals have been assigned to each project particle, any copied geometry will be done so relative to this terrain Normals information. While this would align copied geometry onto the surface of the grid, the direction of travel from each particle would be ignored due to being overridden by this terrain Normals information.

If however the Point Intersection Normal option of the Ray SOP were not activated, the particle's gnomon would be used for orientating copied geometry. This would give direction of travel for copied geometry, but no orientation to the surface of the terrain.

In order for any copied geometry to have both terrain orientation and particle direction of travel, this **terrain Normals information** can be used to drive an **Up Vector** (particle orientation). Once this is set, **velocity information** from each particle can then be used to create a **direction of travel** (stored as new Normals calculation).

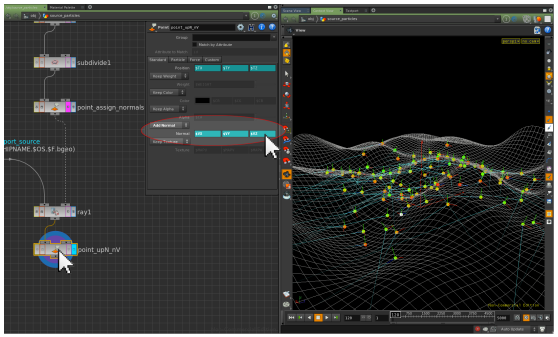
To the **output** of the **Ray SOP**, append a **Point SOP**. In its **parameters** specify:

Particle >
Add Up Vector

Up Vector	**$NX**	**$NY**	**$NZ**

Standard >
Add Normals

Normal	**$VX**	**$VY**	**$VZ**

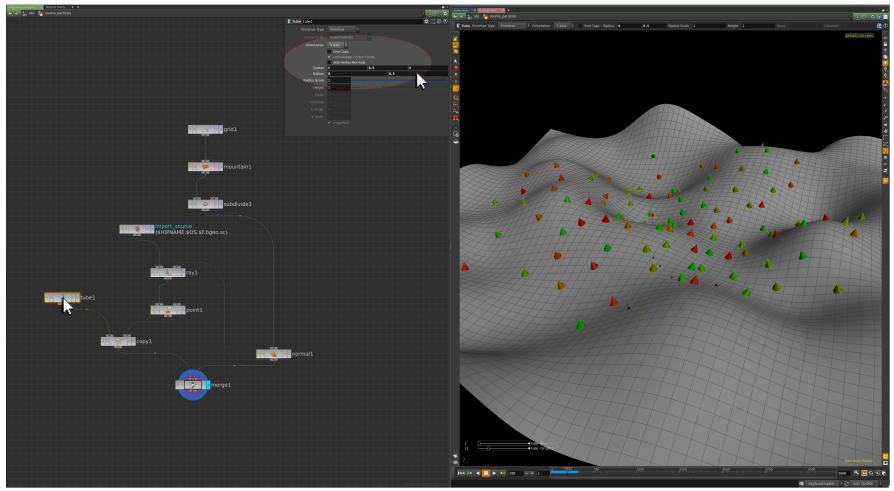


When **PLAY** is pressed the projected particles align with the terrain, and have a direction of travel specified by velocity driven normals.

In a new part of the **Network Editor**, create a **Tube SOP**. In its **parameters** specify:

Orientation	**Y Axis**		
Center	**0**	**0.5**	**0**
Radius	**0**	**0.5**	

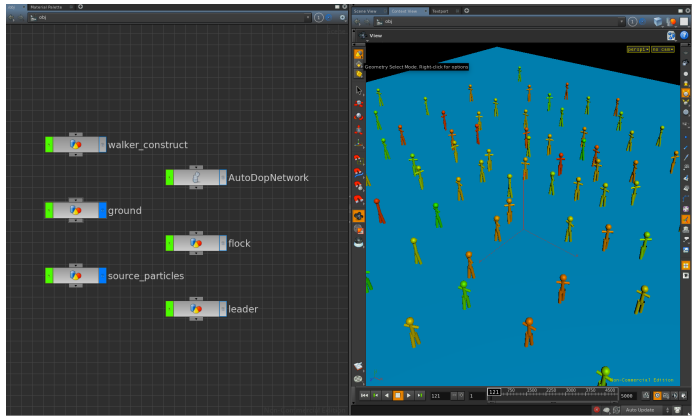
When this tube is copied onto the projected points, the cone shape Is oriented onto the grid and rotates relative to the direction of particle travel.



NOTE: The colour of the particles can be assigned to the copied tubes by activating the Copy SOP's Attributes > Use Template Attributes option.

As a final step, the copied tubes and terrain geometry can be merged together using a Merge SOP. **See file H15_flock_end.hipnc**

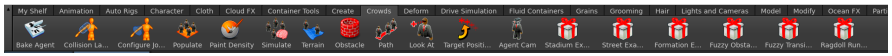
A higher-level implementation of this effect can also be created, where animated geometry is copied onto each particle. The speed attributes of the particle can then be used to control the animation of geometry (for example a walk cycle).



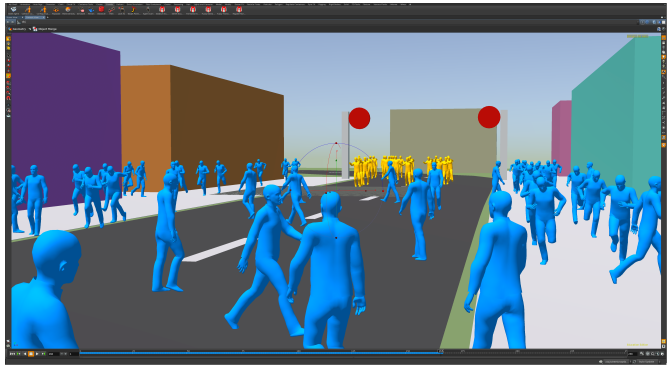
See file army.hipnc

HOUDINI'S CROWD SHELF

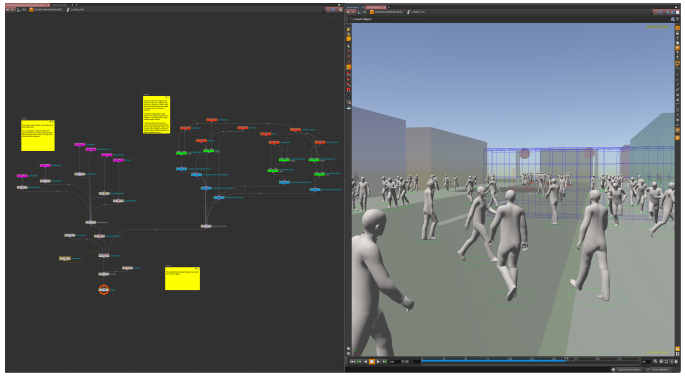
The principles outlined in the above two examples, form the basis of the new Crowd Tools implemented in Houdini 15.



A good place to begin when investigating how these Crowd Tools work is to implement one of the example files found on the Crowds Shelf. For example, activating the **Street Example** will create a simple scene with a crowd of people running away from a crowd of zombies. You will need to **save the scene** in **/transfer** before implementing these examples.



Examination of the **crowd_sim DOP Network** reveals that baked out animation cycles are transitioned between, using the Crowd Solver (which internally uses POP DOPs) to create and compile the effect.



Activating the **Houdini Help Browser** and going to the **Dynamics > Crowd Simulations** section will give an overview of how these crowd systems work, and what is required to generate them.

USEFUL POP VARIABLES

Local Variable	Description	POP Which Adds Attribute	Called in DOPs (VEXpressions) as:	Type	Called in SOPs as (post DOP import):
P	Particle Position	POP Location DOP; POP Source DOP	@P @P.x @P.y @P.z	Vector	$TX, $TY, $TZ
Age	Particle Age	POP Location DOP; POP Source DOP	@age	Float	$AGE
Dead	Dead Status	POP Location DOP; POP Source DOP	@dead	Integer	
Force	Particle Force	POP Location DOP; POP Source DOP	@force @force.x @force.y @force.y	Vector	
ID	Unique Particle Number	POP Location DOP; POP Source DOP	@id	Integer	$ID
Life	Particle Age as 0-1	POP Location DOP; POP Source DOP	@life	Float	$LIFE
v	Particle Velocity	POP Location DOP; POP Source DOP	@v @v.x @v.y @v.z	Vector	$VX, $VY, $VZ
Cd	Particle Colour	POP Color DOP	@Cd @Cd.r @Cd.g @Cd.b	Vector	$CR, $CG, $CB
Alpha	Particle Alpha	POP Color DOP	@Alpha	Float	$CA
PScale	Particle Scale	POP Property DOP	@pscale	Float	$PSCALE
Mass	Particle Mass	POP Propery DOP	@mass	Float	$MASS