Hear Ye! Hear Ye! You have finally arrived at particles. When you first heard about Houdini, chances are it was in a context related to effects and particles. Houdini's particle operations are very robust tools for simulating natural phenomenon like water, smoke, fire, and pirouetting pyroclast. They are also very capable tools for mimicking the dynamics of group behavior as seen in hordes of locusts, squadrons of seagulls, and even schools of fish. Many of the high-end effects you see in movies every day were born in POPs and then marshaled back and forth through the various contexts of Houdini until the stunning final look was achieved.

Here, at the beginning of your adventure into POPs, I thought it would be most helpful to get a little perspective from one of the most experienced and talented Houdini users in the world, Caleb Howard. So, without further delay, here begins the magic of particles, in his words.

**A WORD FROM CALEB HOWARD**

"People are like Particles. We are each born in our own time. Our birth is predictable, based on a straightforward understanding of how such things come to pass. We live our lives with the semblance of self-control. We prefer to feel that we choose our goals and make the decisions for how to surmount the obstacles in our paths. Even so, it is impossible to say, on any given day, what unforeseen events we may face or how tomorrow will compare to today. The events of our lives are, to a large degree, unpredictable. Because of this, the course of our lives is also unpredictable. People and particles—to use a somewhat geeky term—are nondeterministic in their behavior.

As we grow and age, we gather spin and speed. We feel we are responsible for making the decisions which direct the course of our lives. In truth, we may wield a staggering degree of control over the universe around us. By applying our heads, hearts, and hands, in accordance with the greater forces which guide us, we may direct our lives to a great extent. We choose our schools. We choose our careers. We choose our mates... And yet, at times, we can clearly see the greater statistical guidelines of our society. There are systemic rules which govern and limit our behavior to a great extent. There are laws which regulate where and how we may choose to drive. There are fashions and social trends which indicate what we should wear and how we..."
Particle Operators (POPs)

should behave. We mate according to cultural dictates. We can feel free, and while paradoxically knowing that our context is
governed by rules which constrain our decisions very narrowly. Like a sailor who knows the fixed currents and the fluctuating
winds, and may work them both to move in any direction of his choosing, so too may we overcome the prevailing constraints
of our lives to achieve the destinies of our own choosing. So may a particle manage to pull off unique and unexpected behav-
dor despite its motion being defined by external rules and forces larger than itself.

The needs of life and society provide us with our goals from day to day, and even influence the greater arc of our lifetimes. In
traffic and in careers, we seek similar objectives as do all people—food, fun, and a little sleep in between. The closer our cur-
rent station is to that of our neighbor, the closer we will match their life’s achievements, on the whole. We move in similar
directions towards similar ends, and yet we never lose the overwhelming sense of unique purpose which defines each of us. We
are so similar as to be indistinguishable, and yet so distinct that we often fail to recognize the kindred spirits with whom we
share the world. All this, and then we die. Just like particles.

It seems like the great majority of applications that particle systems are used for in the film effects market miss the potential
of philosophical introspection that particles truly provide. Born in the fire of a rocket’s thrust, burning for an instant of pure
acceleration, and then burnt out, gray, and drifting in a con-trail for 42 glorious frames of cinematic action. This isn’t just the
fate of a Hollywood Effects Artist, but is also often the microcosmic fate of the particles we create.

Birth rates and velocity curves, collision objects and wind vectors. These are the tools and jargon of we, the young gods of a
particle’s creation. Unthinking, we birth them by the millions, only to send them momentarily back to the oblivion from
whence they came. This is fine. It pays the bills and, as mercenary work goes, it is no worse than much of what goes on. There
is more available though. These crude forces are just the beginning—like training wheels. They barely hint at the graceful cre-
ations which an integrated environment like Houdini may help an aspiring creator bring to light. Like a simple chain of hydro-
carbons gives rise to a complex planetary biosphere, a diverse set of interlocking control tools driving individuated agents in a
particle system is sufficient to create life, in all of its diversity and detail. These needn’t be daunting aspirations—even for
someone new to Houdini, or to particles—for the secret to life’s rich pageantry is the unstoppable emergence of complexity
from simple rules which are iterated in a recursive environment.

On various occasions, Houdini particles have been given eyes, legs, gaits, brains, fingers, hungers, goals, and dreams. The true
beauty of the tools in Houdini is that we, the creators for whom the tools are built, are in no way constrained by whatever
preconceptions the tool builders may have held for how their tools should be used. Because of an excellent design philosophy,
Houdini embodies tools of uniform, and unconstrained data flow amongst such diverse types of information as geometry, fields,
images, files, and code. Amongst all of the different tools in Houdini, the seemingly humble particle has the greatest capacity
to embody the complexity of life. This is because it is nondeterministic—just like people. So you want a stampede of ten thou-
sand snakes? Well, particles are the solution. Do you need an army of thingies to fall into rank and then engage their common
lifelong foe? Particles are the way. Do you need a monkey to climb a tree, and then hurl poo at the camera? Particles again are a good way to go.

I have a long history with particles. In the beginning, I simply wanted to try a few things out. The following was for a personal project, but it eventually made its way into a moderately public view. I built a valley by painting an elevation map and using a point SOP to lift the walls of a valley around a grassy plain. The valley walls rose to vertical cliffs. Nothing could enter nor leave an area of about 100 square miles. On this terrain, I painted a river, running from the mountain top to the marshy wetlands in the middle. With a smoothing algorithm, and a procedural shader (SHOPs), the terrain was textured according to region with grasslands, forests, marshes, ridges and more. Using this map as a guide, I wrote a script (Hscript) to distribute plants, rocks, trees, and initial populations of four species: Herbivores, Carnivores, Parasites, and Scavengers. Each population was a particle system (POPs) born according to the species' breeding habits, and surviving according to fitness. They carried attributes of physical characteristics (height, sex, speed, color, and so on), and attributes of state (hunger, love, horniness, anger, and so on). For the sake of action, I created in them all a need for water, and in the valley only a single place to drink. I attached a camera object to each particle in turn, to let them see. In the compositing context (COPs), I attached a GL renderer (ROPs) and some image analysis networks (CHOPs) to make them understand what they saw. By these simple applications of different facets of Houdini's toolset, the creatures could see the water they craved. With some attribute communication trickery, they could tell their friends from their foes as well (Objects). Each particle was given a procedural body (SOPs), which worked to couple the particle's basic drives to the terrain.

Then I just turned it on. After just a week of the simulation, I had thirsty herbivores braving the predators to get to the water. The slower ones were eaten, the faster ones got away. The parasites evolved to avoid the heads of the creatures. The scavengers learned to wait until the predators were done before approaching the remains of a kill. All born from the simple particle, but empowered by advanced and diverse tools which could naturally and easily be associated with the simple, and yet complex, particle.

Do I feel anything when a particle I created is killed? I do. I identify greatly with particles, because particles are nondeterministic—like all life, like people. Because of resource issues, primarily a lack of near-infinite disk space, I haven't yet implemented the afterlife which I have designed for particles. It's in the works, though. It's the least I can do for them, as their loving creator.

God, I love Houdini. I really do. Magick, indeed.”

Caleb Howard
C.T.O., Cognitive Imaging Corporation
C.G. Supervisor, Sony Imageworks
Salutations to the POP Node

As in every context, POPs also uses nodes. Figure 8.1 shows an example of a POP node; in this case, the Source POP. Based on what you have already learned about node functionality, you can probably guess what the features of a POP node do. The POP node has input and output connection points, a bypass arrow, a Display flag, a Cook flag (the purple flag), and a Template flag. The Cook flag is slightly new and it is used to tell the system to which node it needs to cook. Let’s say you have three connected POPs and the Cook flag is on the second one. Even if the Display flag is on the third node, the simulation that you are viewing is only utilizing the input from the first two nodes. And finally, it has an icon displaying its type which can be MMB pressed to view a popup window containing all sorts of useful information about the state of the simulation at that node.

Particle Operations Are Cooked in a Simulation

Creating animation in POPs is somewhat different than creating animation in other parts of the package. In POPs, animation is achieved by creating a simulation based on a set of rules that you define. This means that the current state of a particular system is based on which conditions were present at the start and how those conditions interacted over time to get you where you are at the present. In other words, POPs are state dependent. This is different than your normal keyframed animation. Keyframed animation is created by setting keyframes at particular points in time and then having the software interpolate between those points based on some function.

For example, to animate a person doing a karate chop, you might set a keyframe with the hand above the head, and then set a keyframe of the hand at head level and out in front of the body, and finally set a keyframe with the hand below the head level and closer to the body. The software could take those three keyframes and create a smooth motion from start to finish. Although you can set keyframes on various parameters in POPs to gain finer control over the simulation, the particles are not simply interpolating from one keyframe to the next. The sum total of all the various forces at play are considered and the particles are affected by that calculation. Every discreet calculation of a simulation is called a *cook*.

So, the simulation cooks and then advances some defined period of time (which is often the next frame) and takes the state of the system at the previous cook and again calculates the sum total of all the rules and forces at play and applies the result to each particle, and so on, until your simulation is finished. Because of this, simulations cannot be cooked in reverse. If you scrub the playbar forward, Houdini will correctly update the state of the system at each frame based on the forces involved so long as it can cook the update faster than you are scrubbing through the timeline. However, drag backwards along the playbar and the system just stays at the latest frame to which it has progressed. This is because each cook is based on the one before it and it can’t figure out how a particle should be affected going backwards in time.

Simulations Are the Result of Interconnected Forces

In a particle simulation, you are creating a system of interdependent forces. That is to say, choices you make are most often only meaningful when compared to how other forces and choices are being implemented. Compare this to animating at the SOP level for instance. You can animate a box going from 0 to 10 in the world Y axis and you can easily see the mechanics involved. It starts at a value of zero on some particular frame, and then moves to a value of 10 by some future frame based on an interpolation segment function. With particles, giving a system a positive world Y axis force of 10 will only give a tendency to the particles. Where they end up at any particular time is a combination of this force and also many other factors that could be at play like initial velocity, mass, charge, gravity, and more.
Particle Simulations Are Interactive

You can change POP parameters while the simulation is running and interactively view the changes. However, it is important to remember that you are viewing what the system looks like when a change is made at a particular time and not what the system would look like if that change had been present from the beginning. It is often efficient to interactively adjust parameters while the simulation is running to get your look in the right ballpark and then reset and play from the beginning to ensure the change still works as you expected. Remember that you can keyframe almost every parameter in POPs so that if the behavior you want only works when you animate a parameter midstream, you can easily do that too. I basically just said it, but I’ll say it again more clearly because it is very important to remember. Any time you make a change in POPs, you want to reset the simulation by going back to the first frame and then playing it from there (unless, of course you are testing how an animated parameter would work.) Another way to update the system to reflect the change you made is to click the yellow flag in the upper-right corner of the viewport. Any time the simulation in the viewport is not accurate, the flag will turn yellow to indicate this. If you click on the yellow flag, the system will be reset and recooked up to the frame where you currently are. So, if you are frame 96, you won’t see how the system is developing in the viewport from frames 1 to 95. The downside is that usually means you get to sit there for a while and wait for it to catch up. In addition, I usually prefer to reset to the first frame so that I can see the system developing from the beginning again.

The Real Time Toggle

Because POPs use a simulation approach to animation, it is often beneficial to turn off the Real Time Toggle button. This button is shown in Figure 8.2. Consider it a moment and you will understand why. A simulation is state dependent and so is based on the cook before it. If the simulation is complex enough that cooking it takes longer than the chosen frames per second, the simulation will be trying to update itself based on incomplete (and therefore incorrect) previous cooks.

Turn off the Real Time Toggle and run a simple simulation. The frame indicator will speed by, going faster than your FPS. However, crank up the complexity a bit and it may well take 30 or more seconds per cook (or greater than what your frames per second setting is). In this situation, if you were to turn back on the Real Time Toggle, Houdini would not play back the simulation in a way that was useful. You can turn off the Real Time Toggle in order to avoid this problem.

Perhaps the best setting when playing with POPs is to set the option called “Play every frame but never faster than FPS”. Toggle on the Real Time Toggle and open the Global Animation Options window at the far right of the playbar and select that option under Realtime Behavior. When selected, the simulation will play as fast as possible but never faster than the designated frames per second, even if it could cook faster than that. As you can imagine, this is especially useful in the POPs context and it is often a good idea to have this option selected.
Particle SOP or Pop Merge SOP or Popnet SOP

When beginning the study of particle operations, there is sometimes some confusion about the various ways in which you can create a simulation. I'll first clarify the usage of the Particle SOP, and then the Pop Merge SOP, and finally the Popnet SOP. There are two main areas for creating particles: with a Particle SOP or inside a POP network. The POP network option can actually branch off and be in two places, as you can create a Popnet at the root /part level or inside SOPs using a Popnet SOP. Figure 8.3 shows these options. In this book, I most often use the latter method.

If you were to create a Popnet at the root /part level, you would then use a Pop Merge SOP to bring the particle primitives into SOPs for further use. All of that may have sounded a bit like Hog Latin. So, let's walk through an example of each of these methods in order to translate Hog Latin into Pig Latin, which we can all readily understand.

**Particle SOP**

The Particle SOP is the simplest method of creating and manipulating particles. It is useful for simpler simulations involving only one source, collision object, and/or attractor input. Also, the Particle SOP is available in Houdini Select and Escape, meaning you do have some capable particle abilities available even in these less expensive packages. If your needs are relatively straightforward, this SOP can be faster to implement. In many cases that demand more complexity, you will want to go into POPs. POPs offer more control than the Particle SOP in that you have more exacting birthing options, have access to a wider variety of forces, have access to a large number of local variables that represent particle attributes, have control over how attributes are modified and inherited, have control over grouping of particles, and more. It is important to remember that even in cases where you expect the required behavior could easily be achieved with a Particle SOP, it may be wiser to create it in POPs so that any changes to design that might come later can be more readily accommodated. Before you make the assumption that the Particle SOP is ancient, useless technology and skip this section, let me assure you that this operator is strong in the ways of the force. It was the premiere particle generator for all visual effects work in Hollywood up until 1997, when POPs were introduced. It was the engine that drove all the smoke, water spray, and heat shimmer effects in *Titanic*, *True Lies*, and countless other films. So, let’s check out a Particle SOP in action.

1. Open a spanky new session of Houdini if you don’t have one open already. Dive into the default model object or create a geometry object and dive into it if the model object isn’t present.
2. Lay down a Grid SOP and leave it at its defaults.
3. Append a Particle SOP. Press play and you will see that some particles are born, but they aren’t doing anything to jump around about. If you don’t see any particles, home your viewport to ensure the geometry is within your field of view. Also, to more clearly see the particles, you may want to toggle on the Display of Points button, which is located in the stowbar on the right side of the Viewer pane. To add a little interest, go to the Forces tab and add an External force of 1 in Y. Now you have some particles flying up. Notice how the particles are coming up kind of “in sheets” from the grid. That is...
because you are birthing by point number and the grid’s point numbers start at 0 in one corner and
go back and forth all the way to the other end. The operation births points in cardinal order. So, it
births the first particle at point 0, the second at point 1, and so on.

4. To see the point numbers, first put the Display flag back up on the grid and then turn on the display of
point numbers and dolly in if necessary to get a better look. The point numbers toggle is also in the
right stowbar. With point numbers turned on, you should be seeing something like what is shown in
Figure 8.4.

5. Because you can’t make the operation birth from points randomly, instead you can randomize the
position of the points on the grid before it feeds into the Particle SOP, which will effectively be doing
the same thing. Insert a Sort SOP in between the grid and Particle SOPs. On the Point tab in the
parameters for the Sort, change the Point Sort to Random. Make sure the Display flag is on the sort
node and you should see that the numbers are now randomly arrayed across the grid. Go ahead and
turn off the display of point numbers and continue.

6. Put the Display flag back on the particle node and view the simulation (remember to reset it first!). As
a final touch to make it yet a wee bit more interesting, add 5 to Turbulence in X, Y, and Z in the Forces
tab. Play the simulation again and you can see that it is starting to get exciting! Make sure to dolly out
a bit to see all the interesting little twists and twirls the turbulence creates. Figure 8.5 shows you the
results of this simulation after cranking up the Birth rate to 500 and running it to frame 300.

7. Using the Particle SOP, the particles are automatically brought into the SOP context. Why is this impor-
tant? Most often, you will use POPs to create an interesting behavior and then export that behavior to
other contexts for further manipulation. The most common context is SOPs, where you can attach
geometry and shaders to each point in the particle system. Using the Particle SOP, the particles are
automatically in SOPs and available for use.

**Pop Merge SOP**

If you create your POP network at the root `part` level, you will need to use a Pop Merge SOP to bring the behaviors into SOPs. One reason you might want
to use this method is if you want to reference the POP network several times from different areas of the package. Because a POP network is just a set of
instructions with no specific geometry context, you can easily use that network in many different geometry contexts.

1. Using the context menu, jump over to the POPs context. In the path field for all of the panes, you should now see `/part/popnet1/-`. You are now
inside a Popnet located at the root particle level (being `/part`).

2. Lay down a Location POP and press play. By default, you see an expanding cloud of particles. Go to the Attributes tab and set Velocity in Y to 1 and
Variance in all axes to 0.3. Play the simulation and you now have a jet of particles shooting upwards. As you play the simulation, interactively adjust the
variance in all axes (bring up the number ladder using the MMB on the Variance label and drag back and forth). You can see the width of the cone
grow smaller or larger as you adjust the variance values.
3. Now, let’s bring this simulation into SOPs so that you can later attach geometry to the particles. Jump over to the Objects context and create a Geometry object if one doesn’t already exist.

4. Jump into it and lay down a POP Merge SOP. This SOP will allow you to import a simulation from the POPs context. Press the + button in the POP Path field and select the location1 POP. You will need to expand the tree view to get down to it. Again, make sure that location1 is selected and press Accept. Press play and you should now see the simulation playing here in SOPs. The POP Merge just goes to the specified POP and retrieves the state of the simulation at that node.

**Popnet SOP (Called POP Network in the Tab Menu)**

When a POP network is dropped inside an object at the SOPs level, it is called a Popnet SOP. For example, say you have an object called lawnSprinkler and inside it is all the geometry and animation necessary to make the spigot rotate back and forth. It would likely be a good idea to drop a Popnet SOP inside the object and access POPs from there (which might look like /lawnSprinkler/popnet1). That way, if you ever wanted to package it up and distribute it as a digital asset, the particle behavior would already be included. If you had created the water behavior in the root level particle context (/part/popnet1), you would have to first take the time to integrate it properly before packaging the asset. Besides that, when you have the popnet inside the object, it just seems easier to press Enter to dive in or the u key to go back up just as you jump into or out of SOPs. Having a popnet there at the SOP level also makes more visual sense if you are using geometry for birthing sources, collision, or attraction. Everything is in one place and easy to access. Let’s jump through a quick example of this to see the glory.

1. Lay down a Geometry object and jump inside it. Press the Tab menu and drop a POP Network. Press Enter to jump inside it and you should now be in POPs.

2. Append a Location POP and press play. Turn on the display of points if necessary to better see the simulation. Currently, you have particles shooting off in all axes. Go to the Birth tab and change the Variance in Y to 0.1. Reset the simulation and press play. Tumble around in the viewport and you can see more of a pancake looking expansion.

3. Press u to jump back up to SOPs and you can see that you have the Popnet ready and willing to do your bidding. Most of the exercises in this chapter will use this method. Now that you understand where you can create particles, let’s move on to exploring how to control their birthing.

**Impulse or Constant Birthing**

In the Location and Source POPs, you will find two types of birthing possibilities. Impulse birthing is based on a per time step increment while constant birthing is based on a per second time increment. Each of these birthing types is accompanied by an Activation field. These two birthing possibilities operate independently of each other. This means that if you have both of their Activation fields set to 1 and some nonzero number in the Birth Rate fields, they will both be causing particles to be born simultaneously. The difference between impulse and constant birthing is a choice between pulse versus flow, respectively. If you want particles right now, use impulse birthing. If you want a nice flow of particles, use constant birthing.

The Impulse Birth Rate parameter births particles on a per time step increment, and this rate is affected by the oversampling set on the popnet node. By default, this oversampling is set to 1, which means one time step per frame. With this default oversampling, an impulse birth rate of 10 means that 10 particles are birthed per frame, because there are now 10 time steps per frame or 10 iterations through the entire POP network. What would happen if you set
oversampling to some number other than 1? If you set it to 2, the popnet will cook twice per frame. Now, it will birth 20 particles per frame with 10 of those being birthed at frame 1 and 10 more being birthed at frame 1.5. If you set oversampling to 3 with the same impulse birth rate, 30 particles would be born per frame with 10 being born at frame 1, 10 being born at frame 1.33, and 10 being born at frame 1.66.

The Constant Birth Rate parameter births particles on a per second increment, and this rate is affected by what the global animation frames per second setting is. For example, if FPS is set to 24, and Constant Birth Rate is set to 10, 10 particles will be born by the time 24 frames have passed. If FPS were set to 30, 10 particles would be born by the time 30 frames had passed.

In case that discussion seemed a little too esoteric, let’s put cement shoes on it and drop it into the lake of long-term memory.

1. Drop a Geometry object and jump into it. Next, drop a Popnet and dive into it as well.
2. Lay down a Location POP and go to the Birth tab. Change impulse activation to 1. This is saying that birthing particles based on a per time step increment is now activated. Because Oversampling for the Popnet is set to 1, the simulation will cook 1 time per frame. Next, set Impulse Birth Rate to 1. This says that you will now be birthing 1 particle per frame. Reset the simulation and advance to frame 2. MMB the location node and you can find out how many particles your system has birthed so far. The popup window shows that 9 particles have been born. Obviously, that seems a little high considering it should be birthing 1 per frame and you are on frame 2.
3. There is one more thing to tweak. There is another birthing option, Constant Birthing. As discussed earlier, because both Impulse and Constant Activation are set to 1, you are using both methods. This demonstrates why you will most often want to specifically use one method or the other. As you are currently looking at impulse birthing, set Constant Activation to 0. Now, no matter what number is in the Constant Birth Rate field, it is not activated and so won’t be considered.
4. Reset the simulation and again advance to frame 2. MMB the popup window again and, yeehaa!, you have two particles, just as you would expect. Advance to frame 30 and what do you see but 30 particles birthed.
5. Lay down another Location POP, off to the side that is unconnected to the current one. Let’s look at the constant birthing options now. Go to the Birthing tab and change Impulse Activation to 0 and Constant Birth Rate to 1. You are now looking at birthing on a per second basis. Reset the simulation and check the popup. You have birthed 1 particle as expected. Advance to frame 29 and check it again. You still have just 1 particle. As you have set the rate to 1, you are birthing 1 particle per second, which is every 24 frames because the frames per second is at the default of 24. (You might want to check what your FPS is set to in the Global Animation Options window.) Advance one more frame to 24 and check the popup window. You now have two particles.

So, when working with particles, remember that there are two rate options to choose from and, depending on the project at hand, it is likely that one or the other is more meaningful to use. Also remember to set the Activation field for the option not used to a value of 0 or you will be inadvertently using both methods.
Select a Source or Set a Context Source

Notice that there are four inputs on the Popnet node. These inputs can be used for referencing inside the network at the POPs level. For example, in the Source POP, the Geometry Source parameter says Use Parameter Values by default. With this selected, you use the SOP parameter field to navigate and choose the SOP you wish to use for the birthing source. As an alternative, you can change the Geometry Source parameter to First, Second, Third, or Fourth Context Geometry. These four choices correspond to the four inputs to the Popnet node. For example, you could connect a grid to the first input of the Popnet node and then use First Context Geometry in the Source POP to choose the grid for a birthing source. This is effectively no different than doing it the first way mentioned.

Select a Source

1. Lay down a Geometry object and jump into it. Drop a Grid SOP, which you will use for a birthing source. Lay down a Popnet and hop inside it.
2. Lay down a Source POP and click the + next to the SOP field to select a SOP node for a birthing source. Select the grid and press Accept. Go to the Attributes tab and change the Initial Velocity to Set Initial Velocity. Press play and you can see that particles are being birthed from the points that comprise the grid.

Set a Context Source

Delete the path reference in the SOP field so that it is blank again. Jump back up to the SOP level and now connect the grid node to the first input of the Popnet node. Jump back into the Popnet and change the Geometry Source to Use First Context Geometry. This tells the Source POP to go look for whatever is connected into the first input of the POP network of which it is a part and use the input as a birthing source. Press play and you see exactly the same thing as you did when selecting a source in the SOP field.

The latter method can sometimes be more visually informative. For example, let’s say you have a particle system that is simulating rain falling from the sky and splashing off a car driving down the road. You lay down a Popnet, connect the birthing grid into the first input of the Popnet node, the car geometry into the second input, and the ground geometry into the third input. Now, it is easy to see that these three pieces of geometry are being used as part of the particle simulation. You can quickly visually identify that they are related. Figure 8.6 shows this network setup.

Each Particle Has Its Own Space

Just as every object has its own space and so has an origin where the X, Y, and Z axes meet, so too does each particle have its own unique space and associated axes as well. The axes are colored just as they are elsewhere in the package with x being red, y being green, and z being blue. Interesting things are possible because every particle has its own space, like the ability to apply unique transforms on a per particle basis. Figure 8.7 shows five particles with the display of particles origins turned on.
A key concept to keep in mind when working with POPs or any other context in Houdini is to start simply when building a behavior and then build in the complexity. It often happens in production that you are working with complex files and run into an apparent dead-end. The first thing to always remember is to try to isolate what you are trying to do and determine whether it is achievable in a simple test-case scenario. This allows you, as much as possible, to remove all the other factors at play in the real file and just concentrate on the objective at hand. This also often has the benefit of making the test and tweak loop more responsive, as you don’t have to wait for the complex calculations in the real file to run.

**Z AxisAligned to Velocity**

A particle's Z axis is aligned to its velocity vector by default. In other words, a particle's Z axis is aligned to the direction in which it is traveling. In order to see a particle's axes indicator, you must turn on its display in the Pop viewport Display Options. On the Guides and Markers tab, toggle on the Particle Origin button. Before you close the Display Options window, check the size of the indicator display. If it is too small to see clearly, go to the Misc tab and increase the Scale Normal size until the axes are easily visible. Looking at Figure 8.7, you can easily determine which direction each of the particles is traveling by looking at the Z axis in blue.

At times, this auto-alignment can cause unwanted popping-like movement. For example, if you wanted to simulate a ball bouncing along the ground, this default behavior would cause problems. As the ball is falling down, the Z axis would point down, and when it started its bounce back up, the Z axis immediately flips to point in the upwards direction of travel. Let’s create this example to see how this can be a problem and then look at one approach to solving it. Figure 8.8 shows you the result of this exercise.

1. Lay down a Geometry object and jump inside it.
2. Lay down a Grid SOP and change its Size in X to 20 and Size in Y to 5. This will be the ground geometry.
3. For the ball, drop an unconnected Sphere operation and change its Primitive Type to NURBs.
4. Give it some color so that you can easily see the rotation later. Append a Point operation to the sphere node and rename it ballColor. Change Keep Color to Add Color. Hold your pointer over the Color label and a little popup window tells you what the attribute name for each of the fields is. They are diff r , diff g , and diff b . This stands for diffuse red, diffuse green, and diffuse blue. Enter $TX < 0$ in the diff r field and $TX > 0$ in the diff b field. You can see that this has the effect of coloring red the half of the ball with TX values less than 0 and blue the half of the ball with TX values greater than 0.
5. Append a Popnet to the grid so that the grid node is piped into the first input of the popnet and then jump into the popnet.