Workshop 03: Using Expression

RAND vs. NOISE

We have tried rand() – to generate a random number from an input value. Let’s start from this simple example:

A LINE with
- Direction = [1, 0, 0]
- Distance = 10
- Points = 100

You can see from the y-positions of the points, the values returned by rand() is not that smooth. Because there is no relationship between the random values returned by rand(1.2) and rand(1.21). The values returned by rand(1.2) and rand(1.21) can be quite different.

*STX, STY, STZ are the (x, y, z) location of the point inherited from the line.

If you want “smoother” randomness, you can try noise(). Replace the rand(STX) in the above example by noise($TX,0,0) to see what happen. You may find that if the input of the noise() are closer to each other, the returned value will also be closer to each other. Therefore, noise(1.2) is quite close to noise(1.21), making the result looks “smoother”. The following graphs are the 1-dimensional graph of rand() and noise(), where you can see the major difference between them:
In Houdini, the noise() takes 3 inputs, which allows you to extend the idea to 3 different dimensions. In our simple example, we use the first input only. You can give zero to the other inputs.

In general, the parameter value defines the "frequency" of the noise, i.e. how fast the changes are. Therefore, noise($TX*2,0,0) will change faster than noise($TX,0,0).

You can extend the idea of noise() to 2-dimensional: say, replace the LINE by a GRID (with larger number of rows and columns), and replace noise($TX,0,$TZ) by noise($TX,$TY,$TZ) to see the result.

There are 2 important things you have to remember:

1. Theoretically the function noise() return a value between -1 to 1. Therefore, you may have negative value returned. (The actual value returned by noise() in Houdini is in a range around -0.7 to 0.7).

2. If all the 3 inputs are integers, then the noise() return zero. Therefore, try to avoid having integers in all 3 inputs of noise().

In general, the parameter value defines the "frequency" of the noise, i.e. how fast the changes are. Therefore, noise($TX*2,0,0) will change faster than noise($TX,0,0).

You can extend the idea of noise() to 2-dimensional: say, replace the LINE by a GRID (with larger number of rows and columns), and replace noise($TX,0,0) by noise($TX,$TY,$TZ):

Similarly, you can extend the idea to 3-dimensional: use a TORUS (again, polygon-type with larger number of rows and columns) to replace the GRID, and replace noise($TX,0,$TZ) by noise($TX,$TY,$TZ) to see the result.
TURB

The "turbulence" function turb() is the sum of noise() in different scale. The 4th input of turb(), we called it "depth", controls how many noise() should be sum up. You can visualize the result of turb() from the following 1-dimentional graphs:

- noise(x,0,0)
- turb(x,0,0,1)
- turb(x,0,0,3)
- turb(x,0,0,6)
Go back to our GRID example, try to replace \( \text{noise}(TX,0,TZ) \) by \( \text{turb}(TX,0,TZ,1) \), \( \text{turb}(TX,0,TZ,2) \), or \( \text{turb}(TX,0,TZ,3) \), and then you can see the result.

So far we have applied the \( \text{noise}() \) to control the points’ y-position. Of course, you can apply the results to other attributes, say, using \( \text{noise}() \) to control the color of a TORUS, etc:

The color attribute is in the **POINT OP** and change the attribute from **Keep Color** to **Add Color** in the POINT OP.

Even though the result of previous example looks good, you may still ask “\( \text{noise}() \) has negative value returned, so is it good for color?”

Suppose you want to **re-scale the range of the returned value** from \([-1,1]\) to \([0,5]\), of course you can do it yourself, say, by using:

\[
\text{noise}(TX, TZ, 0) + 1 \times 2.5
\]

Or, an easier way is to use the \( \text{fit()} \) function in Houdini:

\[
\text{fit( noise($TX, TZ, 0), -1, 1, 0, 5 )}
\]

where the second to the fifth input of \( \text{fit()} \) means old minimum, old maximum, new minimum and new maximum respectively.

Another useful function is \( \text{clamp()} \), in case you want to clamp the return value within a minimum and a maximum:

\[
\text{clamp( noise( $TX, TZ, 0), -0.2, 0.4 )}
\]

**Exercise 1: \( \text{noise}() \) with time**

How to animate \( \text{noise}() \)? One basic idea is that the inputs of \( \text{noise}() \) can depend on time. Therefore, try to include \( T \) (or \( F \)) in the inputs of \( \text{noise}() \) to create a “horizontal moving wave”. (Animation will be shown during the class.)

\( \text{ch()} \)

Let me use a simple GRID as an example: suppose I want the number of rows in a GRID is always equals to 4 times the x-size of the GRID. How can I do it in a more automatic way? We can “link” the rows in x-size of the GRID together, so that whenever I change the x-size of the grid, the rows will be automatically calculated.

In Houdini, each parameter also has a name. If my GRID is called “/obj/geo1/grid1”, then its x-size parameter will be called “/obj/geo1/grid1/sizex”. (Put your mouse on the parameter name “Size” of a GRID, and wait for a second, then you can see the parameter name.)

So, we can put down \( 4 \times \text{ch("/obj/geo1/grid1/sizex")} \) in the row parameter of our grid. Try to change the grid’s x-size, you can see the number of rows followed.
Now, go back to our noise(). Suppose I want to control the magnitude of noise() by writing:

```
noise($TX, $TZ, 0) * 5
```

Note that the value “5” in the above example is used to control the noise() magnitude. However, can we have an easier way to fine-tune this value, instead of modifying the expression directly? A common trick is to create a NULL OP, which will be used as a “global control panel”. We add an extra attribute in this NULL, and then use the ch() function to replace the “5” in the above expression.

**Here are the steps:**

Step 1) Create a NULL OP. (It is not necessary to turn on its blue-flag.)
Step 2) Right-click on the NULL and choose “Edit Parameter Interface…”
Step 3) Create an extra attribute for this NULL, and assign a meaning name (say, “magnitude”) to the attribute:

1) Choose an attribute type.
2) Press this
3) Give the attribute a name and a Label.
4) Optionally give it a range.
5) Press this
Step 4) Now, we can replace the above expression by:
\[
\text{noise($TX,$TZ,0) * ch("/obj/geo1/null1/magnitude")}
\]

Exercise 2: global control panel

Continue on the previous “horizontal moving wave” exercise. Try to use a NULL to create a “control panel” for controlling the wave’s horizontal speed, vertical speed, and the wave’s vertical magnitude.

Hint: turn on the “Interactive Update” option (at the lower-right-corner), so that you can see the value of parameters change during play-back. This will slow down the machine a little bit, but can give you better idea about what are your parameter values when the scene is playing.

SIN and COS

The function \( \sin() \) and \( \cos() \) are suitable for creating periodic effects such as periodic movement or color. The value of \( \sin() \) goes from 1 to -1, and then goes back to 1. The \( \cos() \) is just the same as \( \sin() \) but with different “starting position”.

Some possible variations of using the \( \sin() \) or \( \cos() \) functions:
A more general form of using the sin() function is:

\[
\text{magnitude} \times \sin( (\text{input} + \text{starting}._{\text{offset}}) \times \text{frequency} )
\]

where \text{magnitude} scales the return value, \text{starting}._{\text{offset}} controls the starting point of the \text{sin}(), and \text{frequency} controls how fast the \text{sin}() will complete one cycle.

Let’s use a \textbf{SPHERE} to test the \text{sin}() function. Connect it to a \textbf{TRANSFORM} and then apply two \text{sin}() formula with different speed to the transform’s translate-x and translate-y parameters:

**Circular and Spiral motion**

The \text{sin}() and \text{cos}() is also suitable for creating circular motion, circular shape, spiral motion, or spiral shape. For example, using the following formula creates \textit{circular motion}:

\[
\begin{align*}
\text{x-position} &= \text{magnitude} \times \sin( \text{frequency} ) \\
\text{y-position} &= \text{magnitude} \times \cos( \text{frequency} )
\end{align*}
\]

Note that if “\text{magnitude}” in x-position and y-position are different, then it creates elliptical motion. Also, try to use different “\text{frequency}” for x-position and y-position to see what happen.

This creates \textit{spiral motion}:

\[
\begin{align*}
\text{x-position} &= \text{frequency} \times \text{magnitude} \times \sin( \text{frequency} ) \\
\text{y-position} &= \text{frequency} \times \text{magnitude} \times \cos( \text{frequency} ) \\
\text{z-position} &= \text{frequency} \ (\text{optional})
\end{align*}
\]

**Circular and Spiral path**

Similarly, starting from a \textbf{LINE} with several points, the following formula creates \textit{circular shape}:
x-position = magnitude * sin( $PT * frequency )
y-position = magnitude * cos( $PT * frequency )

And this creates **spiral shape**: 

- x-position = $PT * magnitude * sin( $PT * frequency )
- y-position = $PT * magnitude * cos( $PT * frequency )
- z-position = $PT (optional)
**TRAIL and POLYWIRE**

A single moving point is not interesting enough. Sometimes you can create interesting visual effects by connecting the points of the path of an moving object. In Houdini, you can either use **TRAIL** (to trace an object’s moving path), or **ADD** (to connect points together to form a path).

Let’s **trace a point’s moving path**, using the **TRAIL**. TRAIL can be used to trace the movement of objects or points over the past several frames:

We can also ask the TRAIL to connect the moving points together, to form a “path”:

If we do it this way, TRAIL output polygonal lines, which will be rendered with a default thickness. To change the line’s thickness, you can add an attribute called **width**, using **ATTRIBCREATE**:
Besides using the width attribute, another way to render lines is to convert it to a POLYWIRE:

Exercise 3: TRAIL of points

Try using TRAIL to create the following animation:

ADD

Another way to “connect points into path” is to use ADD. Different from TRAIL, which connects the path of a point, ADD connects point 0 to point 1 to point 2, etc. We can understand this by using the following example:
And this is the result:

This DELETE deletes the particles, because we don’t want to render them. In this example, particle is the primitive #0, and the line is the primitive #1.

And this is the result:

The previous example explains the idea, but it is not interesting. Assume in the above example the GRID has 10x10 = 100 points. Try to set the options of ADD as the following:
In this setting, point 0 will connect to point 100, then point 200, 300, etc. Point 1 will be connected to point 101, 201, 301, etc. The result will be different from the previous one.

Exercise 4: connect particles’ path

Try to make the following patterns:

**Hints:**

Left pattern: Start from a CIRCLE on the ZX plane and keep rotating the circle. Generate particles and connect them together.

Right pattern: Start from a CIRCLE on the ZX plane and with enough Divisions. Generate particles, and assign a random (or noise) particle starting velocity to each point using POINT. The velocity should also change from frame to frame.

** Week 03 END **