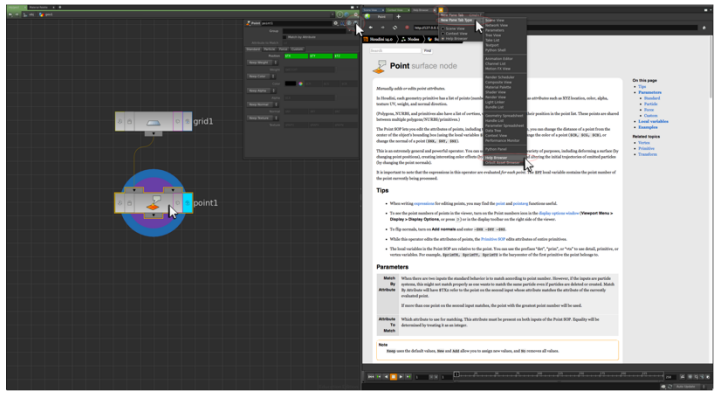
HOUDINI EXPRESSIONS

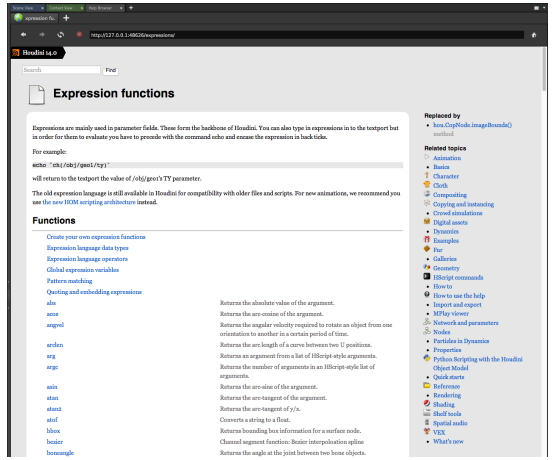
Expressions can be utilised as a **procedural way** of **modifying** or **controlling scene data** and **events** over **time**.

The first step when learning about expressions, attributes and functions is locating where to **find help**. A **Help Browser** can be activated as a New Pane Tab Type over the Viewer for easy Help access. Help for a specific operator can be found by **LMB** pressing the **?** button located in the **Parameters** of the operator.



Clicking on the **Help button** of a **Point SOP** for example will activate a floating **Help Browser** detailing and exampling the Point SOP. The operator help will also (where applicable) return **any local variables** for that operator. **Local variables** allow a way of inputting scene data directly into expressions.

LMB on the **Houdini 14.0** button in the **Help Browser** will activate the **main Houdini help page**. This Help Page can also be accessed through the main Help Menu. Here in the contents listing is a link (**Expression Functions**) to an alphabetically listing of **all Houdini Expressions**.



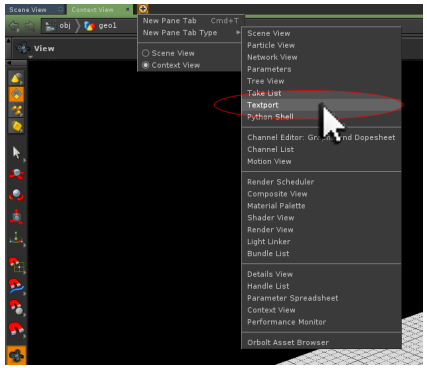
An attempt was made by SESI to replace this expression language with a Python based expression language. It never properly caught on as the **original expression language** is **easier** and **simpler to understand** especially for people from an art-based background. Similarly, many Houdini production artists still use the original expression language due to its ease, simplicity and efficiency.

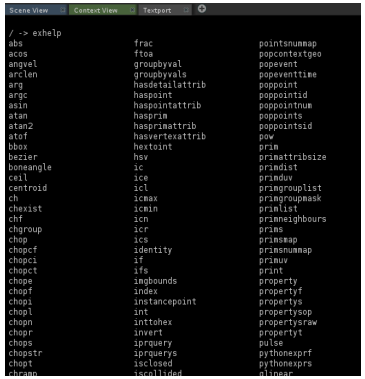
NOTE: When **different functions** are **combined**, they become an **expression**.

THE TEXTPORT

An **alternate way** to access **Expression Help** is through the **Textport**. This can be accessed by activating a **Textport** as a **New Pane Tab Type** over the **Viewer**. The **Textport** is a **Shell** based version of Houdini that can accept HScript commands. It can also be utilised as a way of returning Expression Functions help.

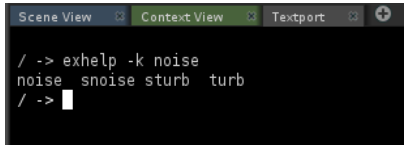
Entering the command **exhelp into a Textport** will **list all** of the Houdini **Expression Functions**. The command **clear into a Textport** will empty the Textport of any listed text.

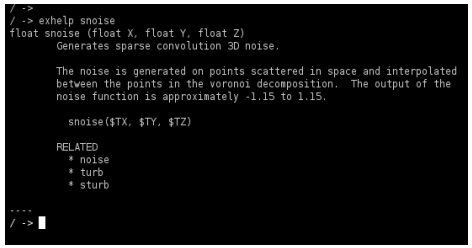




Entering the command **exhelp –k [keyword]** will search through all of the Expression Function help notes for the specific keyword, and then list all of the functions which contain it.

For example **exhelp –k noise** will list all of the expression functions that make reference to the word noise in their help notes. This can make identifying the probable use of an expression simpler if it is unknown, as well as grouping all of the expressions that might deal with a certain topic.





Entering the command **exhelp [function name]** will list the help for a specific function. For example the command **exhelp snoise** will return the help notes for the **Sparse Noise function**.

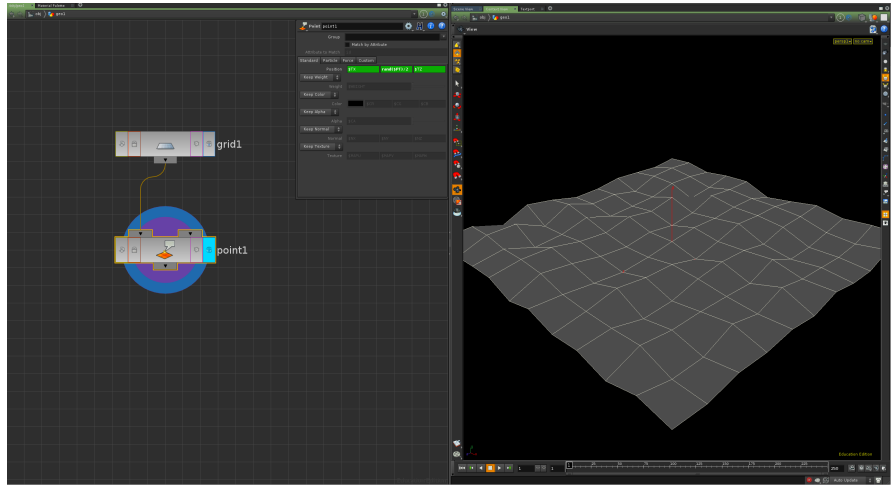
THE ALT + E TEXT EDITOR

Functions and **Expressions** are entered **directly** into **operator parameters** and can get very long when created. With text editing activated in any parameter, **Alt + e** can be pressed on the keyboard to **launch** the expression in a new **external text editor**.

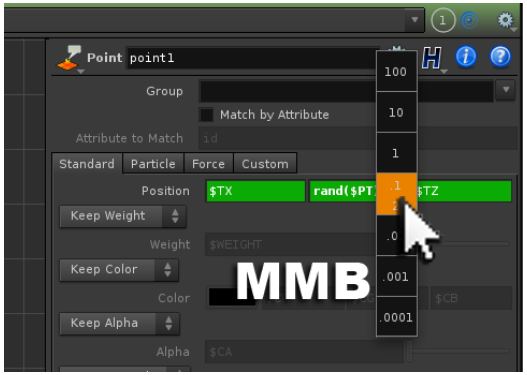
MATH FUNCTIONS

Functions return a value from whatever information is passed into it. Within the context of Houdini, there are mathematical functions, and Houdini native functions. An example of a mathematical function is **rand()**. This will return a (pseudo) random number between the values of 0-1 based upon its input. Math based functions are standard in all 3D software applications. In a new Houdini scene, create a **Geometry OBJ** and press **i** to enter inside it. Here create a **Grid SOP** and append to it a **Point SOP**. Modify the **Position** parameter of the Point SOP to:

Position	**$TX**	**rand($PT) / 2**	**$TZ**



The **Point SOP** will evaluate each point in turn and transform it in the Y axis to random height. Passing **$PT** into the **rand()** function changes the random number generated by using the current point as the random seed. If a constant seed was used (for example rand(3)), the random number generated would be the same for each point. Dividing the result by 2 simply reduces the effect of the operation.

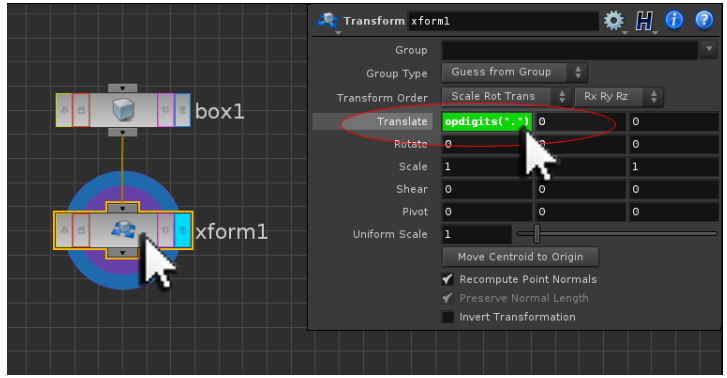


NOTE: The **MMB Number Ladder** can be utilised over any numeric value within an expression to adjust and create a new numeric value if required.

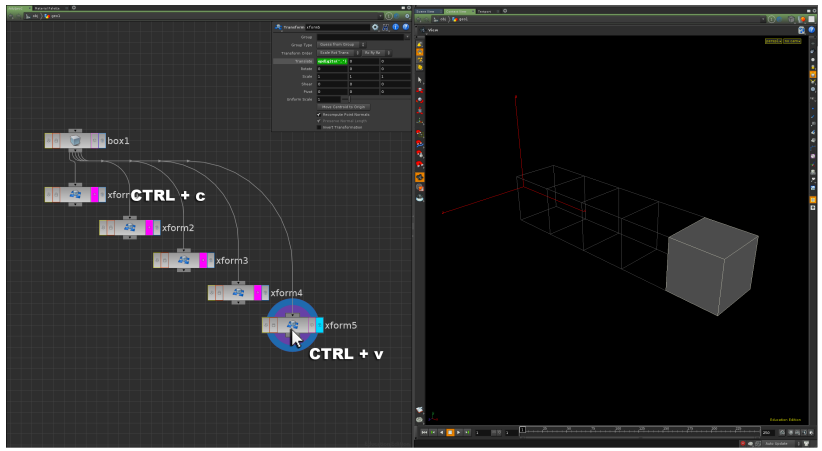
HOUDINI NATIVE FUNCTIONS

A Houdini Native Function is a function that returns a value from another part of the Houdini scene. These functions only exist with Houdini itself. An example of a Houdini Native Function is **opdigits()**. It is designed solely to return the end number of an operator name. In the same Houdini scene, create a **Box SOP** and append to it a **Transform SOP**. Set the **Translate parameter** to read:

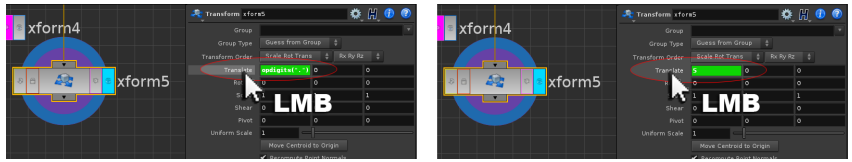
Translate	**opdigits(".")**	**0**	**0**



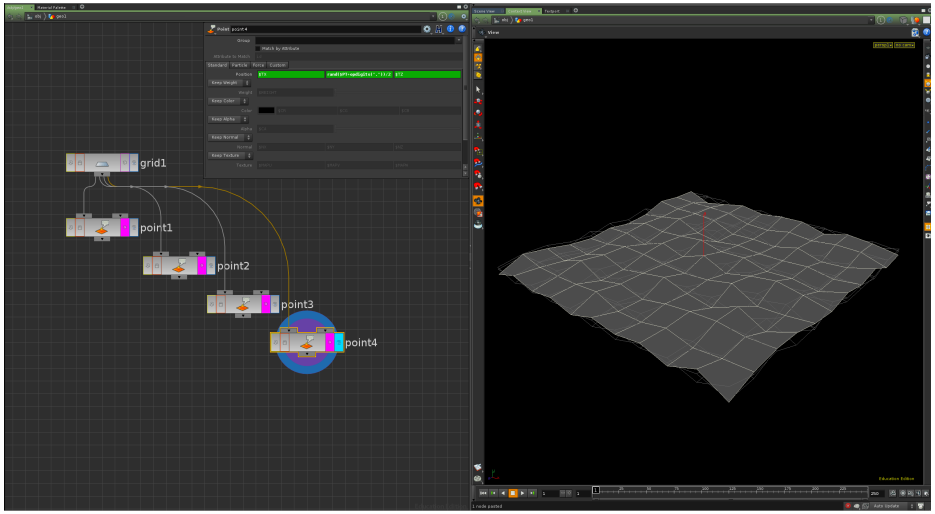
The Translate X parameter will now return the same number as the number at the end of the operator name (xform1 = 1). The **"."** returns the current operator, **".."** would return the number allocated to the Geometry OBJ the SOPs are contained within.



Copying and **pasting** this **Transform SOP** by using **Ctrl + c** followed by **Ctrl + v** will translate the original Box SOP by whatever the end number of the Transform Operator is.
NOTE: The numeric value of an expression can be returned by **LMB** on the **parameter name** to **toggle** between the **expression** and its **value**.



The **opdigits()** function can also be combined with the rand() function affecting the Grid SOP, so when a new iteration of the Point SOP is Copied (CTRL + c) and Pasted (CTRL + v), a new randomised grid is created as a result.



This can be achieved by modifying the Position Y expression to either:

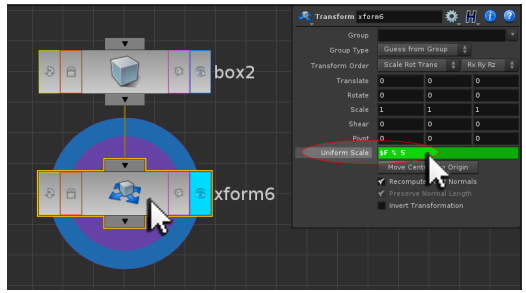
rand($PT + opdigits(".")) / 2 **or** **rand($PT) / opdigits(“.”)**

Both of these new expressions will have differing effects.

Question:
What happens to the visual result of these expressions if the size of the Grid SOP is increased?
Answer:
The scaling factor of the expression will need to be increased or decreased accordingly.

Question:
How might the size of the grid be linked into the expression?
Answer:
Channel References can also incorporated into expressions as procedural controls.

THE MODULUS OPERATOR
The **Modulus Operator** (**%**) can be utilised as a way of creating cycling number patterns. It returns the remainder of a divided by b, and can be used on both integers and floating point numbers. It can for example be utilised in the creation of animated looping textures.



Create a **Box SOP** and append to it a **Transform SOP**. In the **Uniform Scale** Parameter, enter the following expression:

Uniform Scale **$F % 5**

Now as the timeline is scrubbed, the box will scale up on each subsequent frame to a maximum of 4 before resetting to 0. The returned results for this expression are as follows:

Frame 0: Uniform Scale 0
Frame 1: Uniform Scale 1
Frame 2: Uniform Scale 2
Frame 3: Uniform Scale 3
Frame 4: Uniform Scale 4
Frame 5: Uniform Scale 0
Frame 6: Uniform Scale 1
Frame 7: Uniform Scale 2
Frame 8: Uniform Scale 3 etc.

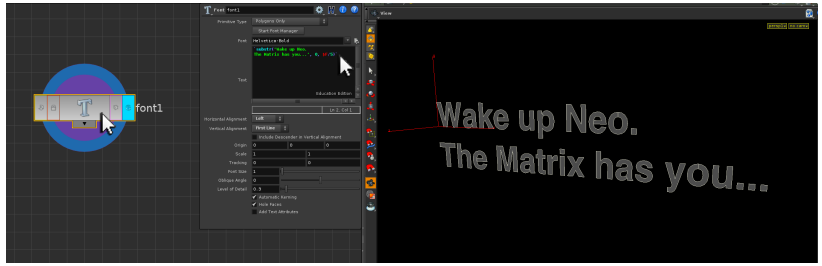
This cycle will occur indefinitely.

TYPEWRITER EFFECT

Entering the following expression into the text entry field of a **Font SOP** will cause each letter to be drawn on screen individually over time as if being typed:

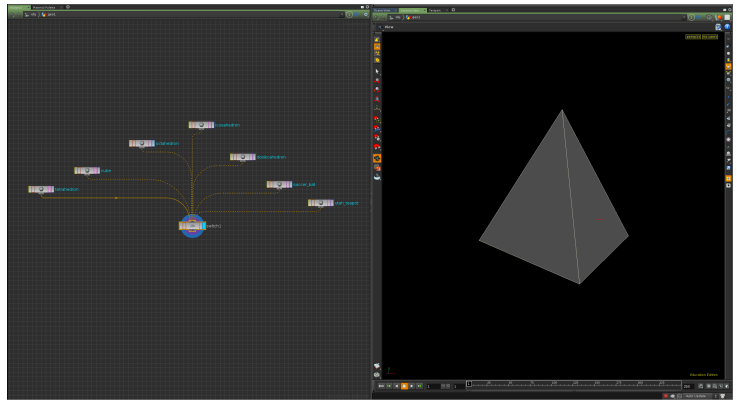
`substr("Wake up Neo.
The Matrix has you...", 0, $F/5)`

NOTE: entering any expression into a text field requires the use of **back-ticks** to evaluate the expression before returning the text string value.



CREATING A RANDOM INPUT SWITCH

Open the scene **random_switch_begin.hipnc**. This scene contains seven Platonic Solid SOPs wired into a single Switch SOP. Each of the Platonic Solids SOPs has been set to a different Solid Type Parameter setting.



At present, with the Display and Render Flag set to the Switch SOP a Tetrahedron Solid Type is visible (Input 0 of the Switch SOP). The aim of this exercise will be to create an expression for the Switch SOP that will randomly switch between the different inputs of the Switch SOP on a per frame basis.

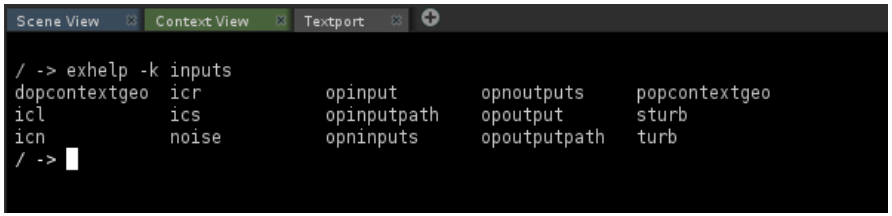
DEVELOPING THE EXPRESSION

The starting point in the creation of the expression is the rand() function. This function will output a random number between the range of 0 – 1. As the aim of the Switch SOP expression is to switch its inputs on a per frame basis, the rand() function can be initially set to the **Select Input** Parameter of the **Switch SOP** as:

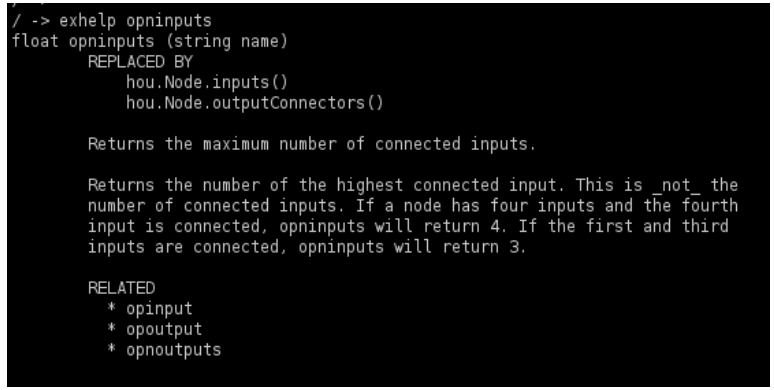
Select Input **rand($F)**

With each new frame, this expression will now output a different random floating point number between 0 – 1. Within the context of a Switch SOP, this expression is defunct because the numeric output range (0 – 1) does not match the input range (0 – 6) created by the total number of Platonic Solids SOPs being wired into the Switch SOP.

A solution to this is to modify the expression from **rand($F)** to **rand($F) * 6**. While this initially is a valid solution, it is unable to accommodate any change to the number of inputs coming into the Switch SOP. A better solution is to find an function that will automatically read the number of inputs for any given operator. A **Textport** search of the Expressions List using the command **exhelp –k inputs**, reveals that there are a number of functions that deal with inputs.



Using **exhelp [name of function]**, this list can be explored further to reveal that the most appropriate function to automatically return the number of inputs of an operator is **opninputs()**. This Houdini native function will return the number of inputs for any operator within the Houdini environment. The syntax for this function can be revealed by accessing the Help Card for the function in the Textport.



Using the opninputs() function as the scaling factor for the rand() function will automatically adapt to any change in the number of inputs assigned to the Switch SOP. In the **Switch SOP**, modify the **Select Input** Parameter to read:

Select Input **rand($F) * opninputs(".")**

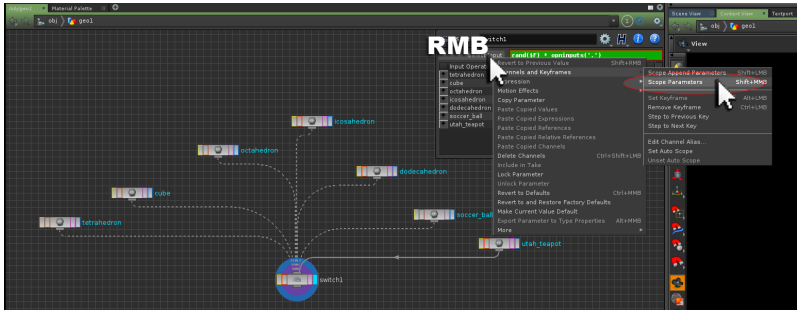
As the number of inputs for the Switch SOP needs to be calculated (also the location of the switching expression), the operator pathname can be set to **"."** This will tell the expression to look for the number of inputs wired into its host operator.

NOTE: if the total number of inputs from another operator was required instead, it could be accessed in the following way:

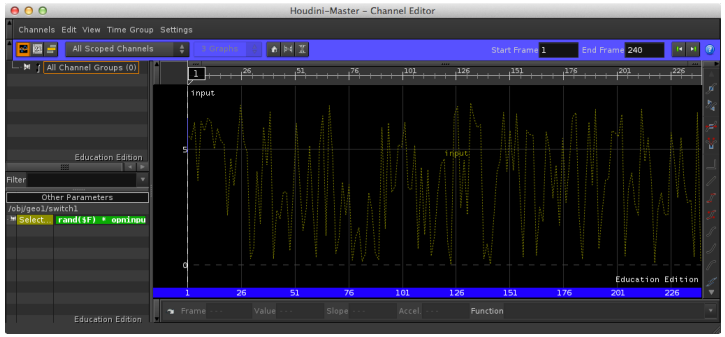
opninputs("../merge1")

CHECKING THE EXPRESSION ACCURACY

If the timeline is scrubbed through, the visual result of the expression seems to be correct. All of the Solid Types can be accounted for. Whether or not the expression is accurate is another matter. The accuracy of an expression can be ascertained by **Scoping the Channels** of the Parameter that the expression is assigned to (this is applicable for expressions which vary over time).



RMB on the **Select Input** Parameter of the **Switch SOP** and from the resulting contextual menu choose **Channels and Keyframes > Scope Parameters**. This will activate a **Floating Channel Editor** where the result of the expression can be viewed in graphical form.



From this graphical representation of the data, two problems can be ascertained. The first is that values go outside of the desired numeric range (7 instead of 6), and the second is that the random values never hit whole numbers. While within the context of the Switch SOP, these two problems are minor; It is however important to get any expression behaving accurately in order to help avoid any potential errors that may occur later on in a larger network.

FIXING THE EXPRESSION

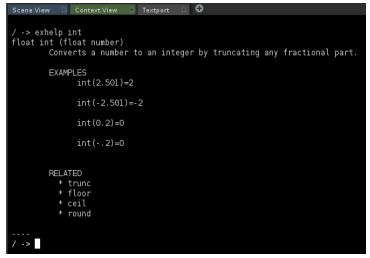
Ensuring that the results of the expression are always returned as whole numbers instead of floating point numbers requires the investigation of expressions utilised for converting floats into integers. This investigation can begin with the Textport expression help listings. In the **Textport** enter the command:

exhelp –k integer

This will return the following list of expressions that all in some way deal with integers.

ceil hextoint int oldrand rand rint round trunc floor

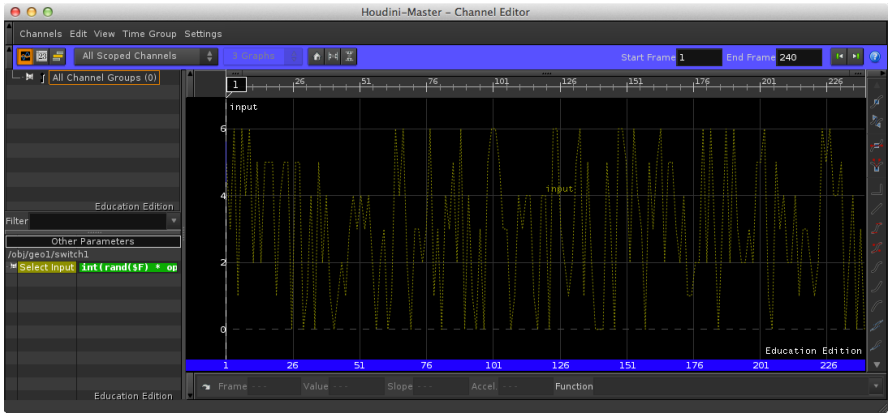
After examining each of the help cards associated with the expressions listed (**exhelp ceil** for example), the most appropriate expression to use in this example is the **int()** function as it strips of the float part of the number without rounding up or down.



Modify the Switch SOP expression to read:

Select Input int(rand($F) * opninputs("."))

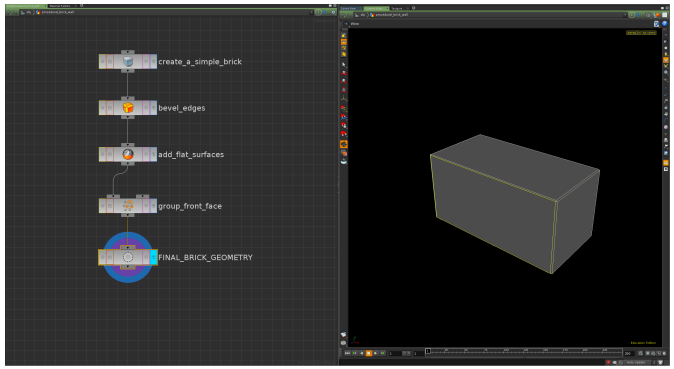
The result of this modification also fixes the number range problem as any floating point value above 6.5 (which may have been rounded up to 7 by the Switch SOP) has now been truncated back to 6. This can be verified once more in the Floating Channel Editor.



With this expression in place, the scene now accurately returns a different Platonic Solid type with each new frame. **See file random_switch_end.hipnc**

CREATING A PROCEDURAL BRICK WALL

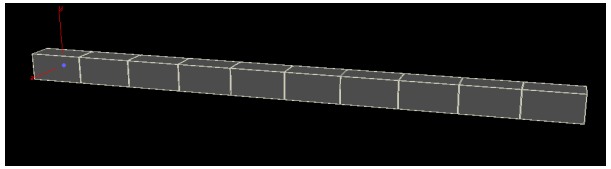
Open the scene **procedural_brick_wall_begin.hipnc**. This scene contains a simple brick geometry which has been bevelled and made to render as a flat surface. The brick also has its front face grouped for the purposes of texturing.



To the **FINAL_BRICK_GEOMETRY Null SOP** append a **Copy SOP**. In the **Parameters** for the Copy SOP specify:

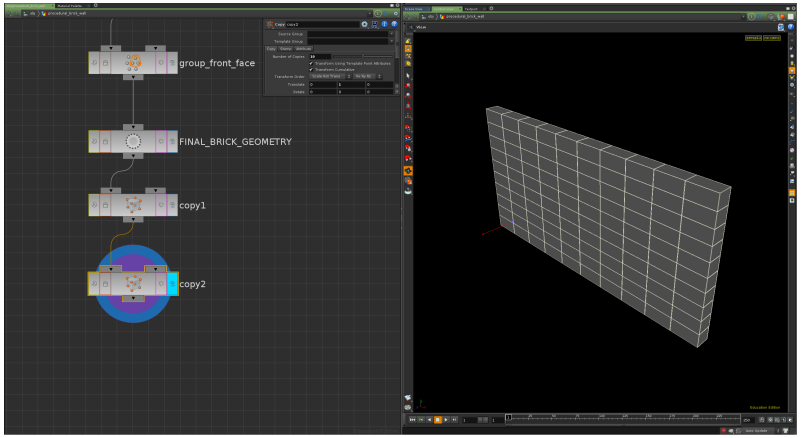
Number of Copies	**10**		
Translate	**2**	**0**	**0**

This will create the first row of bricks for the wall.



To the network **append** a **second Copy SOP**. This will allow for the row of bricks to be duplicated forming a basic wall. In the **Parameters** for the **second Copy SOP** specify:

Number of Copies	**10**		
Translate	**0**	**1**	**0**



As each subsequent row of bricks needs to be offset relative to the first, an If Statement in conjunction with a Modulus Operator can be utilised to do this. The context of an if statement can be understood by using the following pseudo-code:

if (I am awake) then
{ rise and shine; }
else
{ remain in bed; }

The expression syntax for an if() statement in Houdini is:
if(I am awake == TRUE, rise and shine, remain in bed)

CONDITIONAL OPERATORS

As the if() statement is reliant upon a condition returning s specific value, conditional operators can be used. The syntax for these operators is:

==	**equal to**
>=	**greater than or equal to**
<=	**less than or equal to**
!=	**not equal to**
||	**or**
&&	**and**
!	**not**

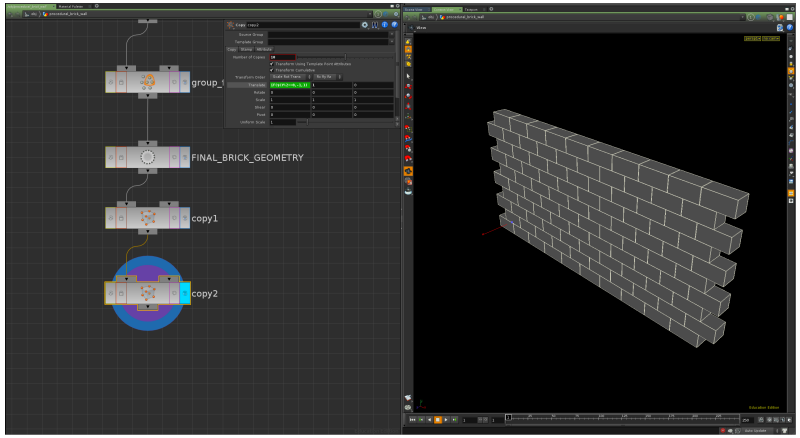
These conditional operators can also be combined. In this pseudo-code example, getting out of bed could require both the ‘awake’ and ‘wish to’ conditions to return a TRUE value before the ‘rise and shine’ option is implemented:

if(I am awake && I wish to get out of bed == TRUE, rise and shine, remain in bed)

In the **Translate Parameter** of the **second Copy SOP** specify:

Translate	**if($CY % 2 == 0, -1, 1)**	**1**	**0**

This will evaluate each new brick row copy and offset it either left or right by one unit depending upon if the condition $CY % 2 returns a zero or not. As the % 2 operator returns either a 0 or a 1 alternately, this ensures every other line of bricks is offset in the same way. A brick wall of any size can now be created, and because of the expression-based set up, it will always be correctly configured.



As the effect of the original Group SOP is added to with every copy of the brick created, the end result is that each front face of each brick is now part of the front_face group. This can be confirmed by **MMB** on the **icon** of the second **Copy SOP** node.

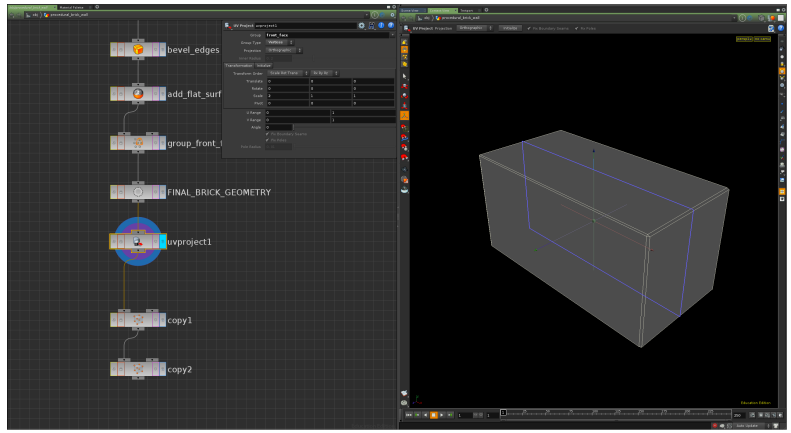
See **procedural_brick_wall_stage1.hipnc**

ASSIGNING TEXTURES

Expressions can also be utilised to control the procedural placement of textures. In this example, a single texture map will be assigned to each brick but randomly offset so that each brick remains unique. To the **FINAL_BRICK_GEOMETRY Null SOP** append a **UV Project SOP**. In the **Parameters** specify:

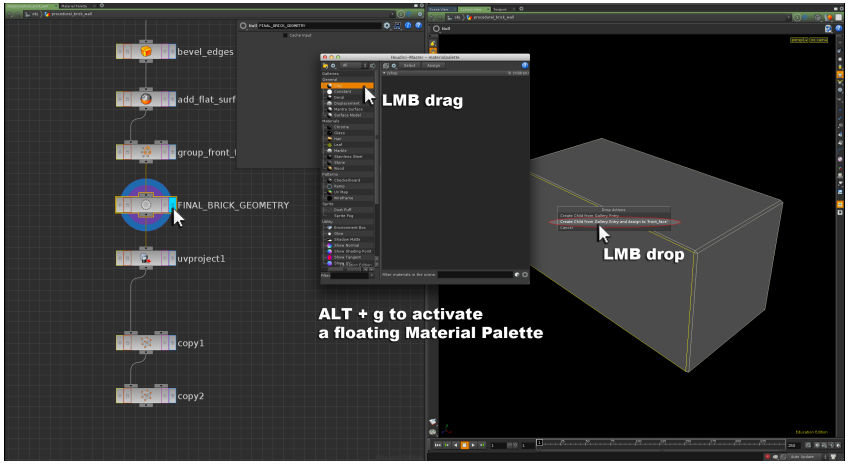
Group	**front_face**		
Group Type	**Vertices**		
Transformation >			
Scale	**2**	**1**	**1**

This UV Projection will also be carried forward onto each copied brick creating the wall.



Activate the Display and Render Flag for the FINAL_BRICK_GEOMETRY Null SOP, and ensure the View Pane is a Context View with Tool Mode activated. The front_face selection of the brick geometry should be visible in the Viewer.

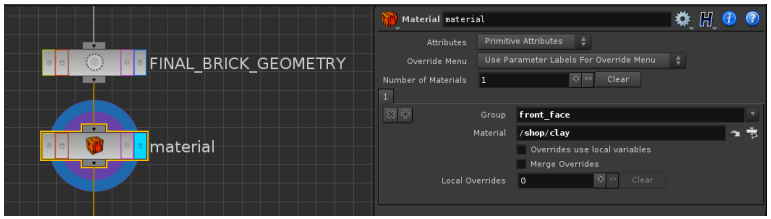
Press **ALT + g** to activate a **floating Material Palette**, and **LMB drag** and **drop** a **Clay Material** onto the **front face** area of the brick. This will activate a short list of **drop options**.



From the drop options list, choose:

Create Child from Gallery Entry and Assign to 'front_face'

This will create a **Material SOP** in the **Network Editor**, automatically assigned with the **Clay Material**.

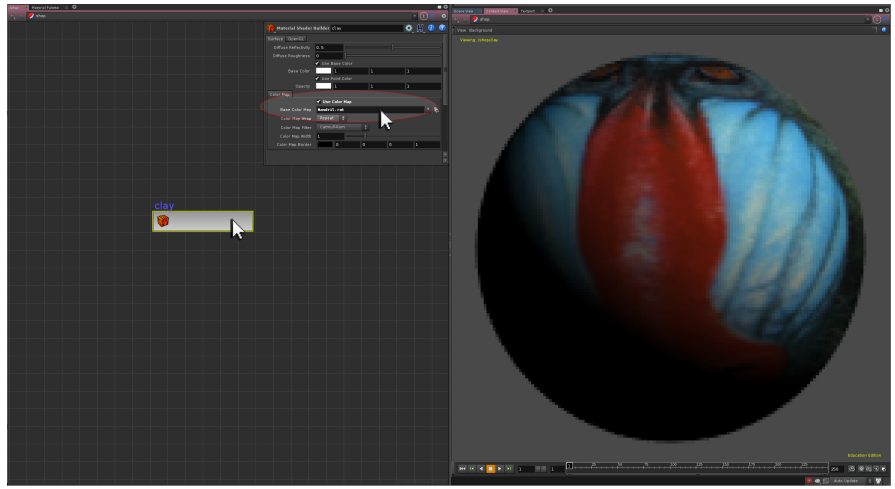


CONFIGURING THE MATERIAL

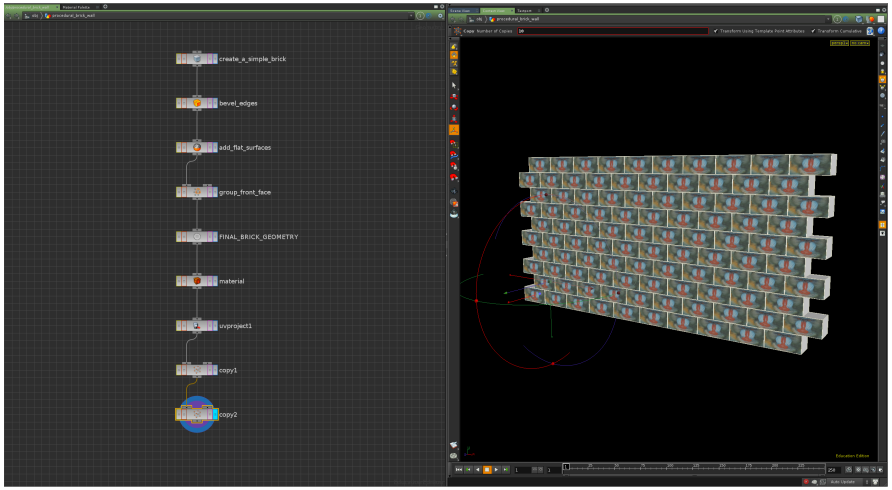
Using the Material SOP's Jump to Operator button, switch to the /shop level of Houdini. In the parameters for the Clay Material specify:

☑ **Use Color Map**

Base Color Map **Mandril.rat**



This will load a default Houdini texture map which can be utilised for testing purposes. Return back to Geometry Level and set the **Display / Render Flag** to the final node in the chain. The wall now has the Mandril.pic texture assigned to each brick.

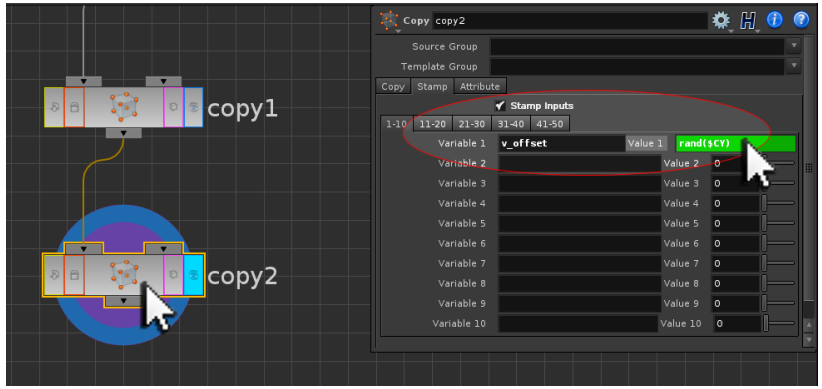


See file **procedural_brick_wall_stage2.hipnc**

OFFSETTING THE TEXTURE

In order to offset the texture placement on a per brick basis, the stamping functionality of the two Copy SOPs can be utilised. Under the Stamp section of the **first Copy SOP** activate the **Stamp Inputs** toggle option, and create a **Variable** called **h_offset** that has a **Value** of **rand($CY)**. This will create a varying value for each brick copy ($CY) creating the row of bricks.

Repeat this operation with the **second Copy SOP** creating a Variable called **v_offset** again with a value of **rand($CY)**. This will create a varying value for each stacked copy ($CY) of the brick row. These two values can now be passed back up to the UV Project SOP in order to horizontally and vertically offset the texture on a per brick basis.

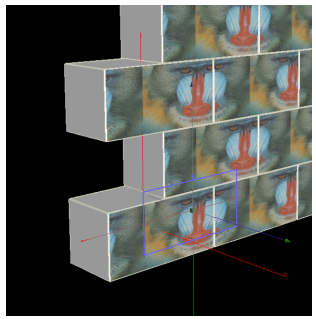


APPLYING THE STAMP FUNCTION

Activate the Parameters for the **UV Project SOP** and amend the **Translate X** Parameter located under the **Transformation** section to read:

Translate	**stamp("../copy1","h_offset",1)**	**0**	**0**

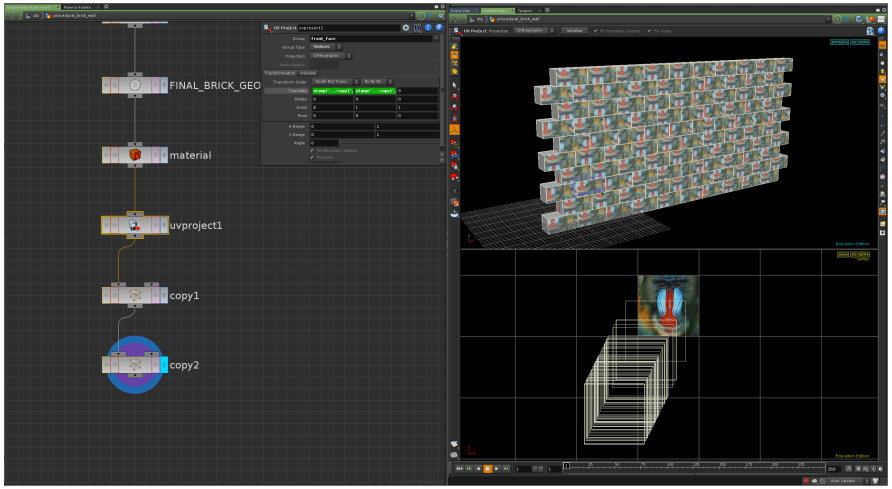
This will create a random offset in the x axis for each brick contained within the row. Note however that each stacked row of bricks has the same offset applied.



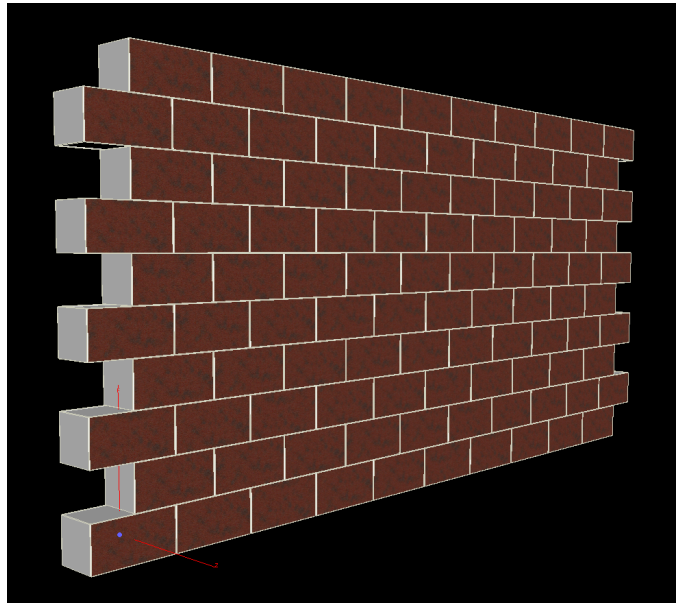
This can be rectified by incorporating a Stamp Function from the second Copy SOP as part of the Translate X expression by adding.

stamp("../copy1","h_offset",1) + stamp("../copy2","v_offset",1)

This entire expression can also be copied (Ctrl + c) and pasted (Ctrl + v) into the **Translate Y** Parameter of the **UV Project SOP** to give vertical variation on a per brick basis. Examination of the UV Editor also reveals the horizontal and vertical offset now taking place.



As each brick now has its texture individually placed, the **Mandril.pic** image can now be replaced by a tiling brick texture map.



Although each brick is now similar in terms of aesthetic, they are not identical giving a more naturalistic effect to the brick wall. Further texture placement randomisation could be achieved by passing the Stamping Values from the two Copy SOPs into the Rotation Parameter of the UV Project.

ADDING FURTHER BRICK VARIATION

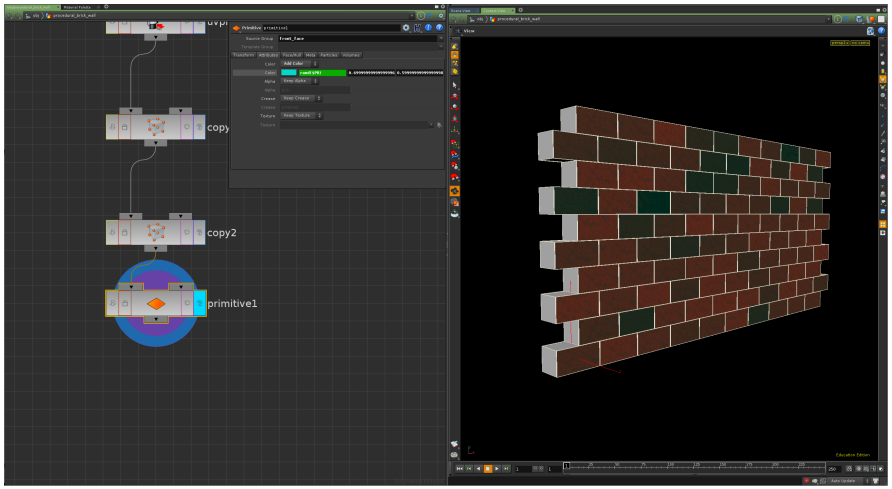
Another level of random variation to the bricks can be added by slightly colouring the geometry of each individual brick. This can be achieved using the **Primitive SOP**, which will assign a geometry colour attribute (Cd) which will be multiplied into the material calculation at render time.

To the **second Copy SOP** append a **Primitive SOP**. A Primitive SOP behaves in a similar manner to the Point SOP by cycling through each primitive of its input geometry in turn. Under the **Attributes** section of the **parameters** for the Primitive SOP, activate the **Add Color** drop down menu and **RMB** on the **Color parameter**. From the resulting menu choose **Delete Channels**. This will clear the default colour variables.

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

Source Group	**front_face**		
Color	**rand($PR)**	**0.7**	**0.6**

This will create a random red value for each individual brick primitive face being evaluated ($PR) and fixed values for the Green and Blue Color Attribute Channels.



THE FIT FUNCTION

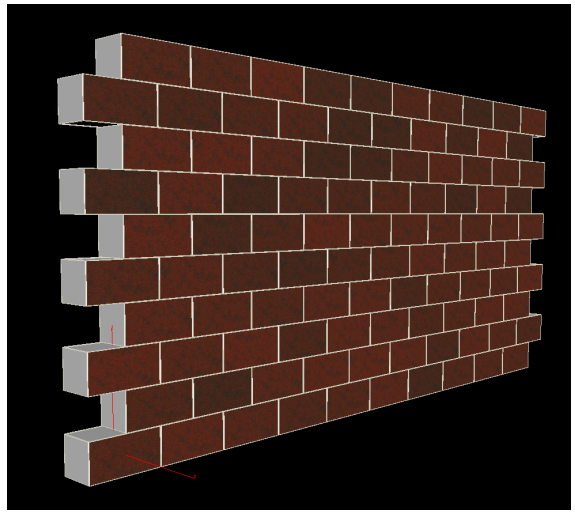
At present the dominant level of green in the random colour distribution is being caused by those bricks whose random red level assigned to them is very low. This can be corrected by re-ranging the 0-1 values created by rand() Function to a range that does not go beneath a certain value. This can be achieved by utilising the Fit Function. Modify the expression affecting the **Red Color Attribute Channel** from:

Color	**rand($PR)**	**0.7**	**0.6**

To:

Color	**fit(rand($PR), 0, 1, 0.5, 1)**	**0.7**	**0.6**

The pseudo-code for the fit function **fit (the value to be changed; the original minimum value; the original maximum value; the new minimum value, the new maximum value).**



In this example the default 0-1 value produced by the rand() Function has been re-ranged to 0.5-1. This means that while no value now returned will go beneath 0.5, the full scope of randomness is maintained; however has been fitted into a smaller range. The visual end result is a brick wall whose texture and colour are now being affected on a per brick basis. With this preliminary engineering complete, the wall's aesthetic can be further refined when necessary. See file **procedural_brick_wall_end.hipnc**

SIN AND COSINE FUNCTIONS

It is useful when dealing with mathematical functions to understand what they do, how they can be modified and how they can be animated. The simplest and often most useful mathematical functions are sin(), cos() which will respectively return sine, cosine numeric values. To visualise these mathematical functions, a Line SOP passed into a Point SOP can be utilised.

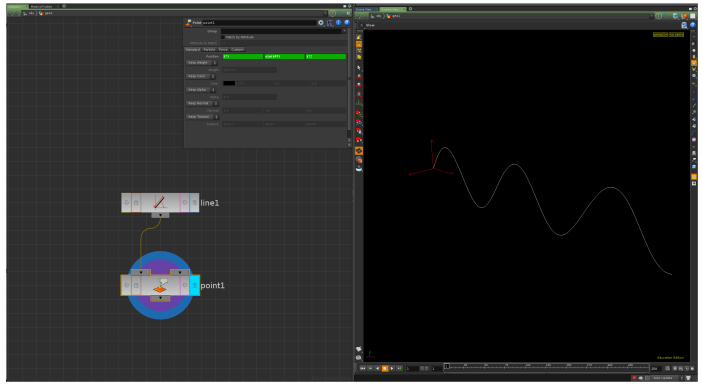
In a new Houdini scene, create a **Geometry OBJ** and press **ENTER** to go inside it. Here create a **Line SOP**, setting the parameters to:

Direction	**1**	**0**	**0**
Distance	**10**		
Points	**1000**		

This will create a horizontal line.

Append to the **Line SOP** a **Point SOP** and modify the **Position** parameter to read:

Position	**$TX**	**sin($PT)**	**$TZ**

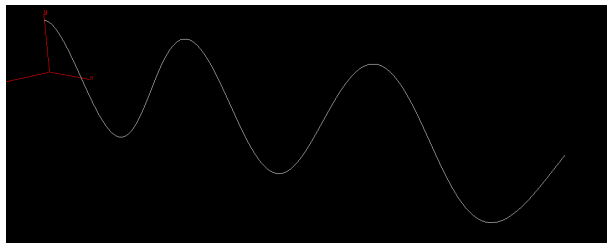


The Point SOP will evaluate each point in turn and perform the sin() function on it. As the variable $PT is being passed into the sin() function, the current point number will determine the sin() value generated. The visual result is a sine wave based upon the Y axis.

Using **Ctrl + c** & **Ctrl + v** copy and paste the **Point SOP** to create a second one. This time modify the **Position** parameter to read:

Position	**$TX**	**cos($PT)**	**$TZ**

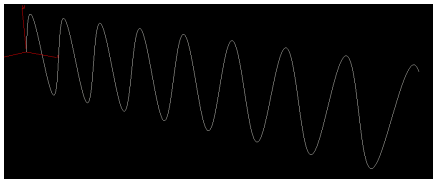
This will create a cosine wave.



MODIFYING THE FUNCTIONS

The syntax for the sin() function can be modified, either inside the brackets or outside of them. Modifying inside the brackets will change the wavelength and modifying outside of the brackets will modify the amplitude. Select the first Point SOP and modify the Position parameter to read:

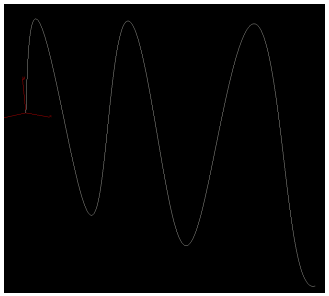
Position	**$TX**	**sin($PT * 3)**	**$TZ**



The frequency of the wave has increased along the length of the line. Dividing by 3 instead of multiplying will decrease the frequency. Select the first Point SOP and modify the Position parameter to read:

Position	**$TX**	**sin($PT) * 3**	**$TZ**

The height of the wave has now increased to range between 3 and -3.



ANIMATING THE FUNCTION

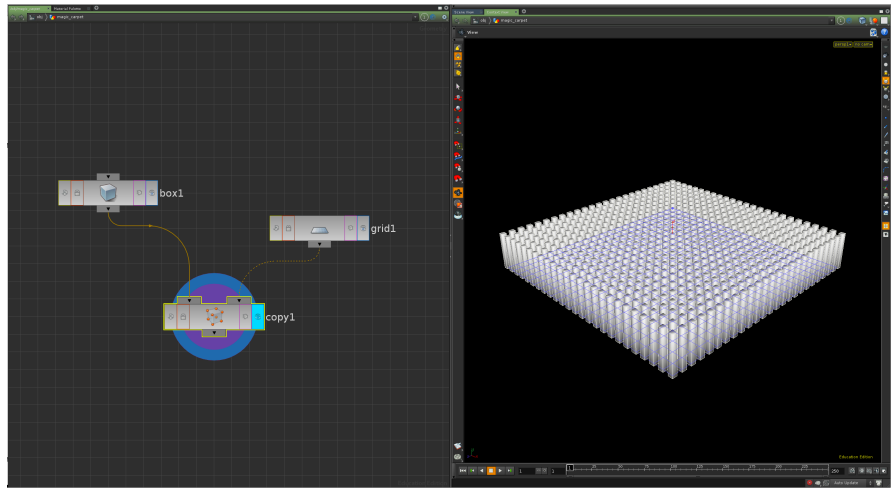
The frequency of the wave can also be animated by the addition of a time based variable (frames - $F or time - $T). Select the first Point SOP and modify the Position parameter to read:

Position	**TX**	**sin($PT + $F)**	**$TZ**

When play is pressed, the sine wave will appear to move. The speed of the movement can be controlled by multiplying the $F variable. For example, sin($PT + ($F * 10)) will speed up the animation. Conversely dividing $F by 10 will slow down the animation.

MAGIC CARPET

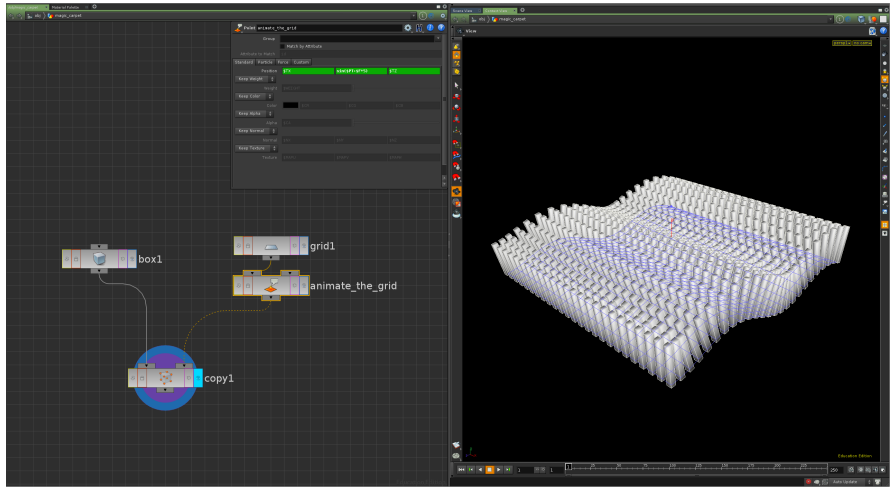
Open the scene **magic_carpet_begin.hipnc**. This scene contains a Box SOP copied onto a Grid SOP.



After the Grid SOP insert a Point SOP and modify the Position Parameter to read:

Position	**$TX**	**sin($PT+$F*5)**	**$TZ**

This will create a undulating wave movement to the Grid when **PLAY** is pressed.



As each copy of the Box SOP is being orientated on the Grid SOP by the grid's surface normals, these can also be animated to add to the effect. It is better to do this in a new Point SOP so that the effect can be bypassed easily if necessary. The Display of Normals can be activated from the Display Options found on the right hand side stow bar of the View Pane.

To the Grid SOP append a **second Point SOP**. In the **Parameters** set the **Keep Normals** Parameter to **Add Normals**. This will allow the existing normals to be seen and manipulated. Set the **Normals Parameter** to read:

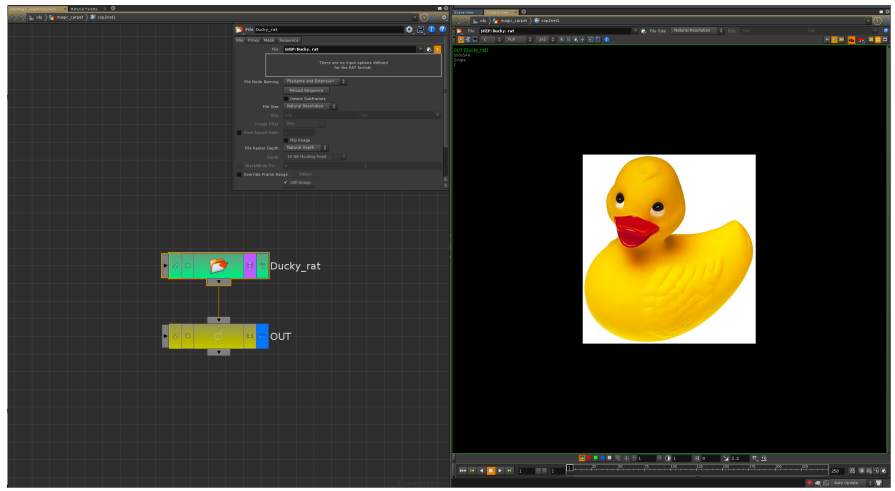
Normals	**sin($PT+$F*10)**	**3**	**cos($PT+$F*10)**

Now the boxes rotate with the normals, with the effect radius of the X and Z animation being controlled by the Normals Y parameter.

ASSIGNING GEOMETRY COLOUR FROM AN IMAGE

Insert a **UV Texture SOP** after the **Grid SOP** to assign UVs to the Grid (the default Parameters will work correctly for this example). UV co-oridinates will allow for an image to be assigned as a Colour Attribute to the points of the grid.

Create a Compositing Network (TAB - COP2Network), and go inside it. Use a **File COP** to read in the image **Ducky.rat**.

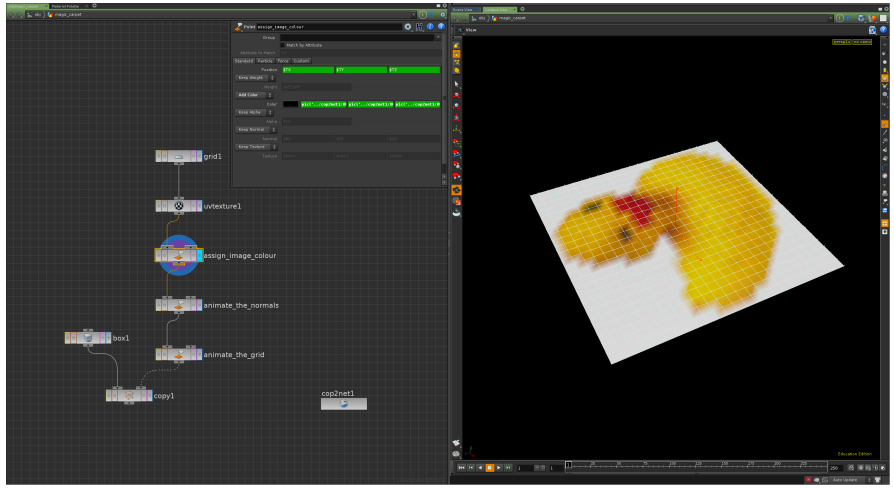


Wire this into a **Null COP** renamed to **OUT**.

Return to **SOP Level** and insert a **Point SOP** after the **UV Texture SOP**. Set the Keep Color Parameter to **Add Color** and insert the following expression:

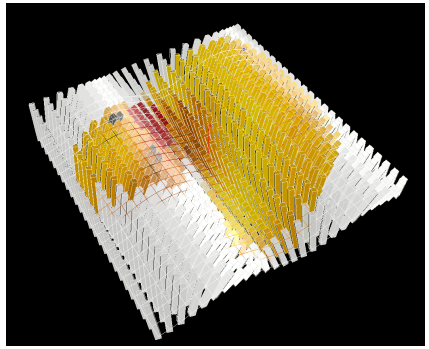
Color

(red channel)	**pic("../cop2net1/OUT",$MAPU,$MAPV,D_CR)**
(green channel)	**pic("../cop2net1/OUT",$MAPU,$MAPV,D_CG)**
(blue channel)	**pic("../cop2net1/OUT",$MAPU,$MAPV,D_CB)**

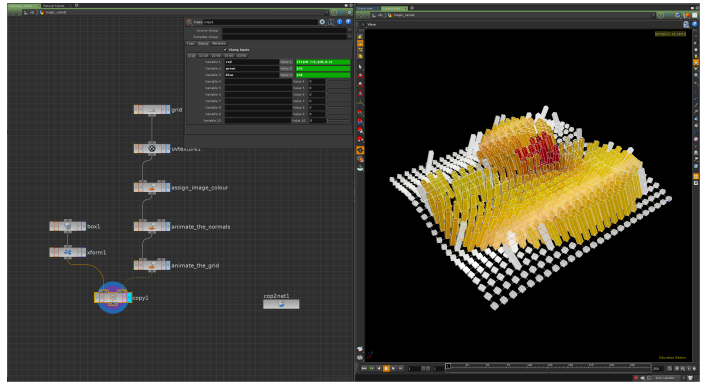


The image will now be mapped onto the geometry in accordance with the UVs assigned to it.

Go to the **Attribute** Section of the **Copy SOP** and activate the **Use Template Point Attributes** tick box.



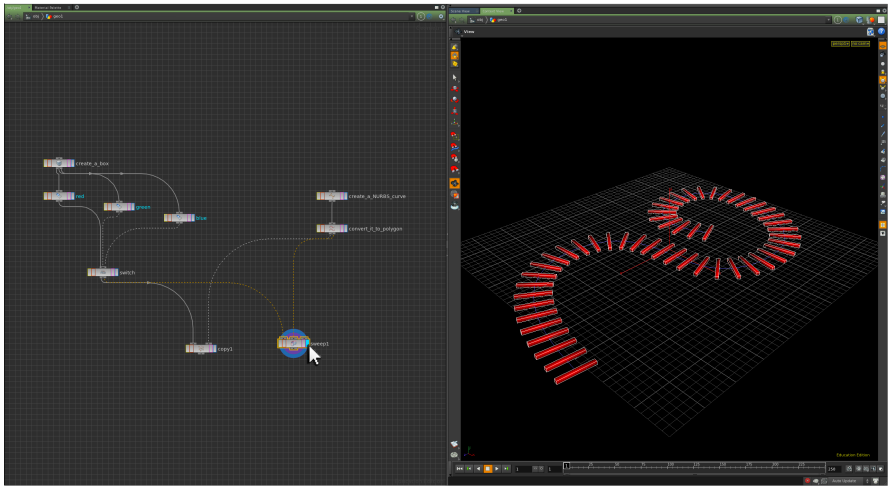
The colour has now also been copied onto the boxes. The colour information could now be utilised to control the scale of the copied boxes for example. This could be done by Copy SOP stamping the $CR, $CG or $CB values created back onto the original box geometry.



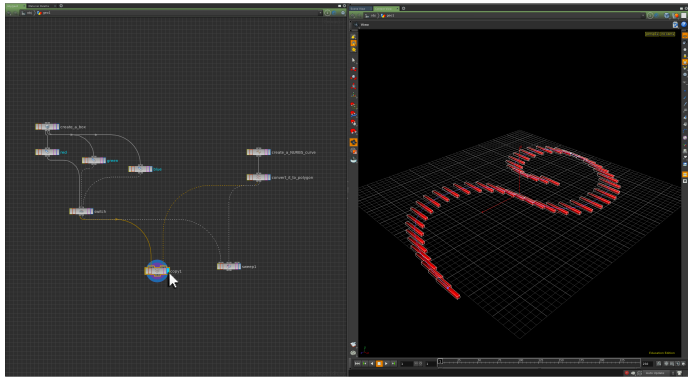
In this example, the various expressions utilised could all be assigned in a single Point SOP. Keeping expression-based work within individual operators however makes managing expressions simpler. **See file magic_carpet_end.hipnc**

COMBINED SWEEPING AND COPYING

Sometimes it can be advantageous to combine the functionalities of both the Sweep SOP and the Copy SOP in order to gain better control placing geometry onto curves to create effects. A Sweep SOP will place geometry on a curve as expected; however does not have any 'Stamping' functionality. A Copy SOP won't place geometry on a curve as expected; however does have tremendous 'Stamping' functionality. This example will look at how both operators can be combined. **Open the scene random_sweep_copy_begin.hipnc.**



This scene contains a box swept onto a curve. When the display of the Sweep SOP is active, a railway track effect is generated. If however the display of the Copy SOP is activated, the railway track effect is lost due to the boxes not having correct alignment to the curve. This is due to the curve not having predefined Normals to align the copied boxes to.

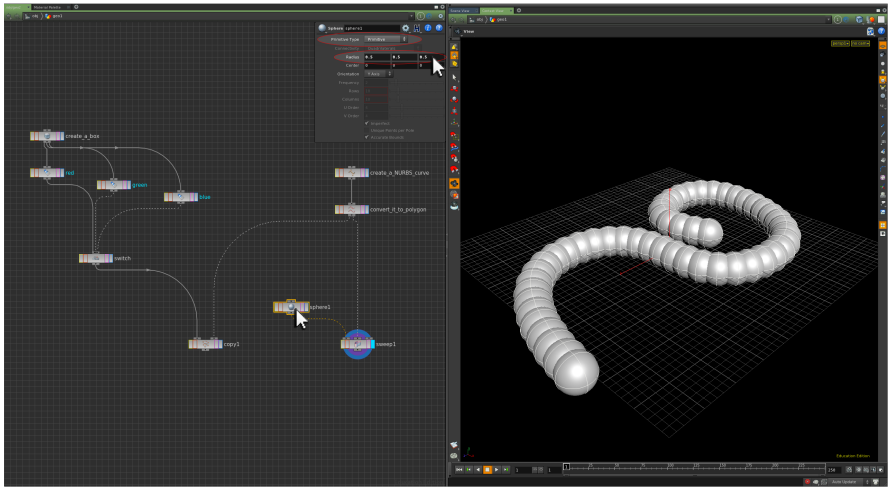


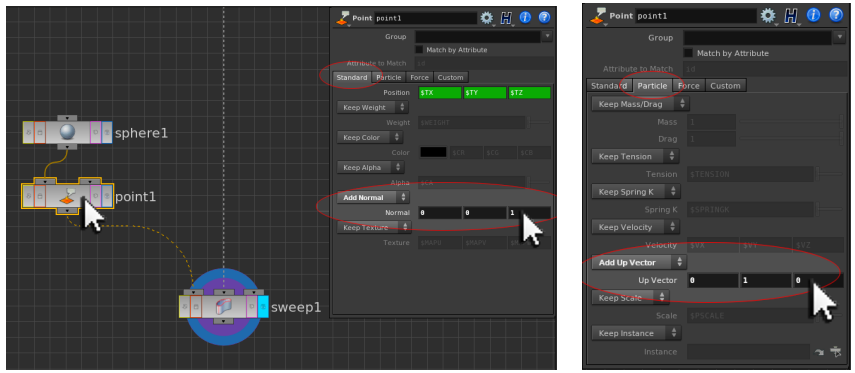
As the Sweep SOP inherently gives good alignment of the swept geometry relative to the curve, this functionality can be used to generate appropriate curve Normals that will create the same effect when using a Copy SOP. This in turn will unlock the 'Stamping' functionality of the Copy SOP for creating more interesting effects.

In the **Network Editor**, **create** a **Sphere SOP**, wiring its **output** into the **first input** of the **Sweep SOP**. In the **parameters** for the Sphere SOP specify:

Primitive Type	**Primitive**		
Radius	**0.5**	**0.5**	**0.5**

A Primitive Sphere is a **single central point** used to generate a sphere shape from. This means each swept primitive sphere is a single point swept onto each point of the backbone curve. Doing this retains the original curve point data that the Copy SOP can understand, as well as allowing for the point attributes of the sphere to be modified further.





ASSIGNING POINT NORMALS AND AN UP VECTOR

With the primitive sphere assigned to the Sweep SOP, both point normals and an up vector can be activated on the sphere before it is swept. This will give each swept sphere both an orientation direction as well as an up direction. This in turn will aid the correct orientation of any geometry subsequently copied onto them.

Append to the Sphere SOP a **Point SOP**, and under the **Standard** section of the **parameters** specify:

Add Normal

Normal **0** **0** **1**

NOTE: After activating Add Normal, delete the Normal channels to remove the references to $NX, $NY, $NZ.

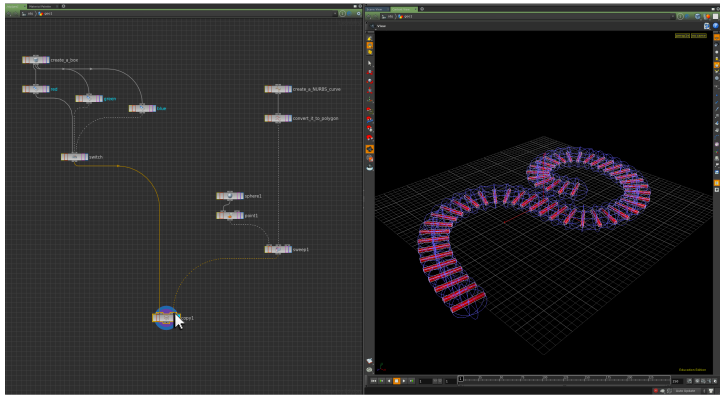
Repeat this process to add an Up Vector, by going to the **Particle** section of the **parameters** and specifying:

Add Up Vector

Up Vector **0** **1** **0**

NOTE: After activating Add Up Vector, delete the Up Vector channels to remove the references to $UPX, $UPY, $UPZ.

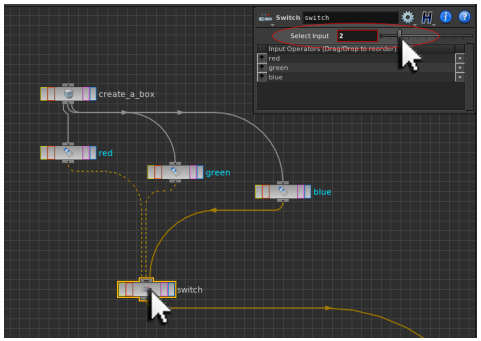
As a final step, the **output of the Sweep SOP** can then be **wired** as the **second input** to the **Copy SOP**. This will create the railroad track effect as before; however now the additional functionality of the Copy SOP can be used to create more bespoke effects.

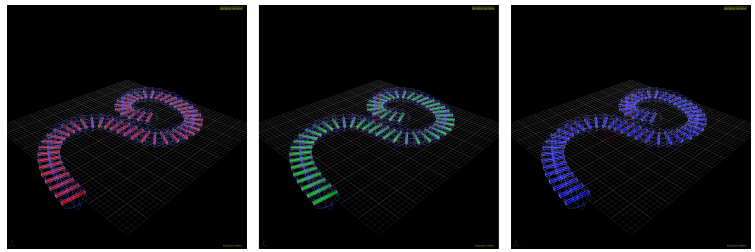


See file random_sweep_copy_stage1.hipnc

ACTIVATING COPY STAMPING

The current rail track geometry is a red coloured box. Alternate colours have also been created; however are currently inactive. As these alternate colours have been fed into a Switch SOP, modification of the switch mechanism can result in either red, green or blue rail tracks being uniformly created.



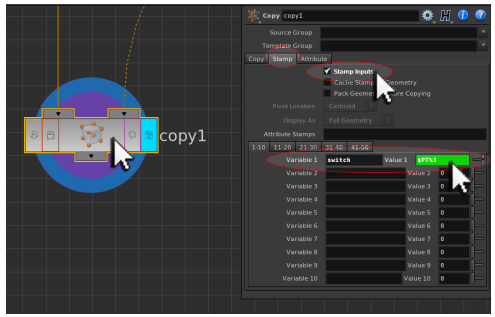


Copy Stamping can however be activated, so that each individual rail track geometry becomes red, green or blue. This is where an expression based overriding of the switch mechanism will choose a colour as each individual box is copied onto the rail track curve. Under the **Stamp section** of the **parameters** for the **Copy SOP**, specify:

☑ Stamp Inputs

Variable 1	**switch**	**Value 1**	**$PT%3**

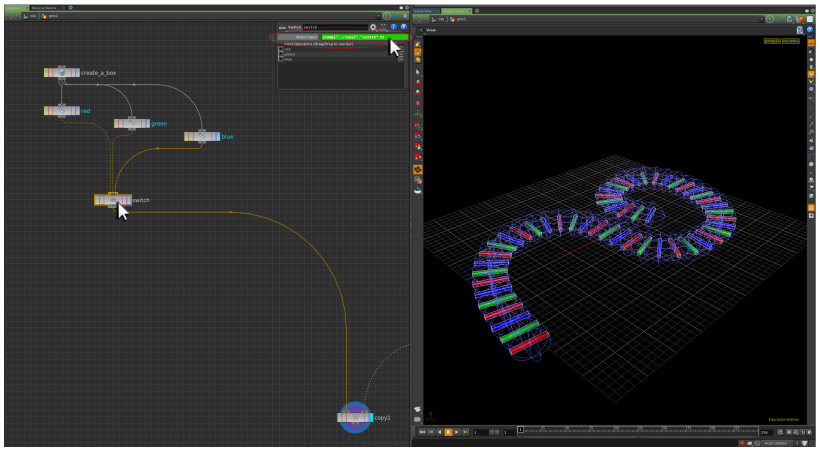
This will create a variable called switch for each copy point of the rail track curve, assigning a cyclic point value of either 0, 1, or 2 accordingly.



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

Select Input **stamp("../copy1","switch",0)**

This will call the switch variable each time the rail track box is copied onto the rail track curve.



Visually this results in a repeating red, green and blue allocation of colour to each piece of rail track geometry when it is copied.

ADDING RANDOMIZED COLOUR VARIATION

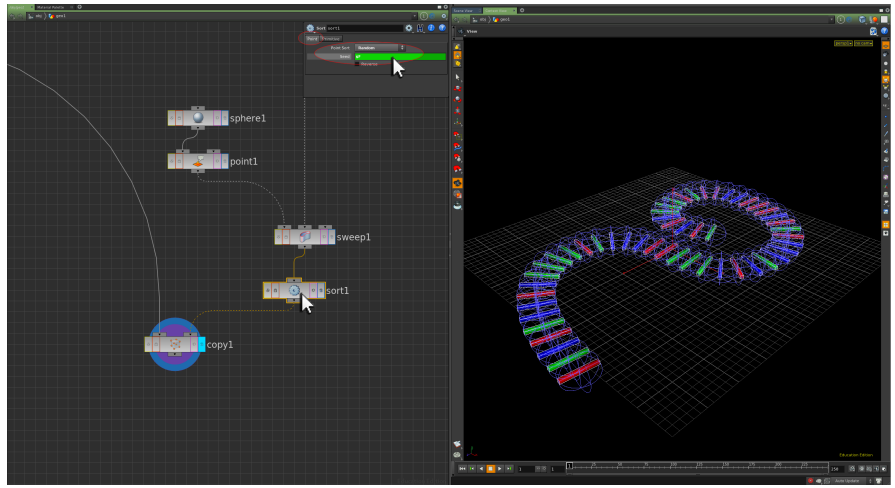
Currently the distribution of colour is a uniform cyclic event. This can however be both offset and randomized by modifying the point numbers of the incoming rail track curve geometry. A **Sort SOP** will allow for such **point number modification** to take place **without disrupting** the actual curve **geometry topology**.

To the **output** of the **Sweep SOP**, insert a **Sort SOP**. In its **parameters** specify:

Point >

Point Sort **Random**

Seed **$F**



Now each coloured rail track is randomized in terms of its distribution. When **PLAY** is pressed, a new randomized colour distribution occurs with each frame.

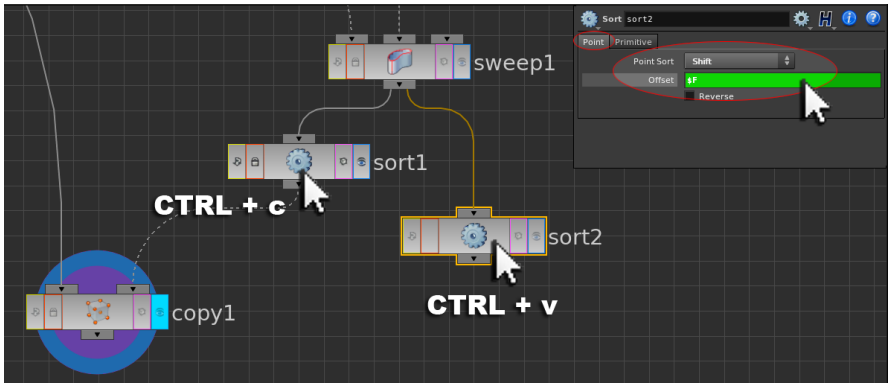
NOTE: Activating a Seed parameter of $F is optional depending upon the type of end effect being created; however is added to this example for completeness.

ACTIVATING COLOUR OFFSET

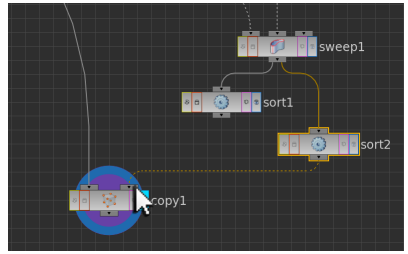
A variation on the randomized colour effect is to keep the original red, green, blue cyclic distribution; however offset the rail track curve point numbers so this cycle steps through each point in turn. **Copy (CTRL + c)** and **Paste (CTRL + v)** the **Sort SOP** to create a **second instance** of it. In the **parameters** for this **second Sort SOP** specify:

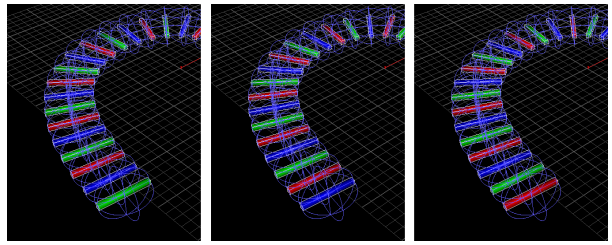
Point >

Point Sort	**Shift**
Offset	**$F**

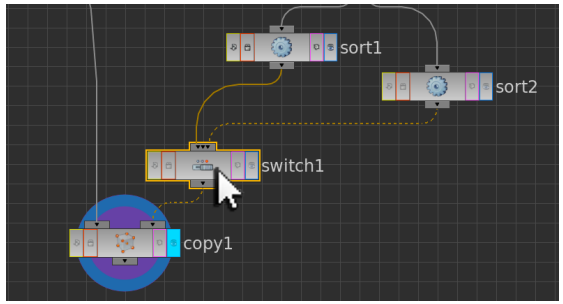


When the **output** of this node is wired as the **second input** to the **Copy SOP**, and **PLAY** is pressed, the cyclic red, green, blue colour distribution now steps through the rail track geometry on a per frame basis.





A Switch SOP can also be created to switch between the randomized colour effect and the offset colour effect.



See file random_sweep_copy_end.hipnc