Houdini
3D ANIMATION TOOLS

HSCRIPT EXPRESSIONS

Animation and visual effects are creative fields built on a highly technical foundation and it is important to balance your artistic aspirations with technical know-how. Houdini's node-based workflow is designed to make CG technology more artist-friendly while providing you with the flexibility to shape things any way you want. Expressions offer another way to enhance the creative process by giving you complete control over your scene data.

To help you learn how expressions can help you in your work, Side Effects has posted an 18 lesson video tutorial series along with this document. This material covers a variety of simple yet essential expression building blocks including Global, Standard and Custom Variables as well as Math and String Modification functions.

Understanding how expressions can be used effectively in Houdini is key to giving yourself the creative edge and allowing you to unleash your imagination!

SIDE EFFECTS SOFTWARE

INTRODUCTION

ATTRIBUTES, VARIABLES AND HSCRIPT EXPRESSIONS.

ATTRIBUTES, VARIABLES AND HSCRIPT EXPRESSIONS IN HOUDINI

When you type a value into a parameter field in Houdini, you assign this value to that parameter and the associated node reacts accordingly. If you type the value “5” into the Translate X field of an object, then it will move 5 units in the view port, and it will stay there until you type in another number. These values may also be keyed each frame, and thus allow the user to change the values over time to achieve their desired animation.

Expressions let you go beyond these simple concepts to open up a new world of possibilities. Throughout this tutorial series, you will learn how data can be passed to parameters to automate a variety of effects while maintaining the ability to go back and make changes later.

ANATOMY OF AN EXPRESSION

An **expression**, is typically any value that is not either a simple string (text such as a file path) or number. This can be something as simple as a **variable**, a **math equation** or an **expression function**. In this series, we will be looking at HScript Expression Functions (as opposed to Python) which are native to Houdini. HScript can be a very fast and concise way to retrieve and manipulate information.

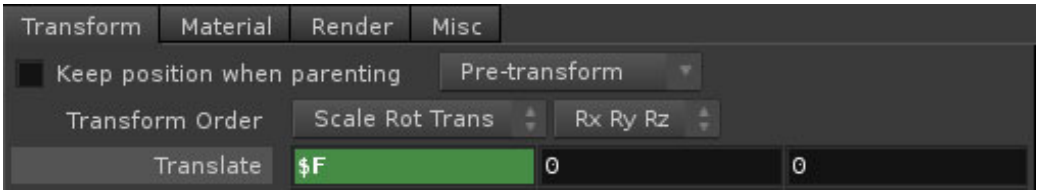
Variables are essentially aliases, short code words which call up information that has been stored somewhere in your file (or even in your operating system). They may be accessed by using the dollar sign: $ followed by the variable name i.e.: the **global variable** $F can be used to retrieve the current frame. It is also possible to create **attributes** on your geometry which may be accessed using a **custom variable**. For example, it is possible to have a color attribute on your geometry which can affect the shading of your geometry or even be used to abstractly trigger some other effect you want to achieve. For example it is possible to state that all red geometry will emit particles whereas the particles will collide only with green geometry.

As you can see, once you begin to use **attributes** as triggers for effects it becomes very desirable to be able to do simple math and comparisons each frame in order to achieve these operations. The next stage of building expressions therefore will include simple math such as addition, subtraction, multiplication, and division.

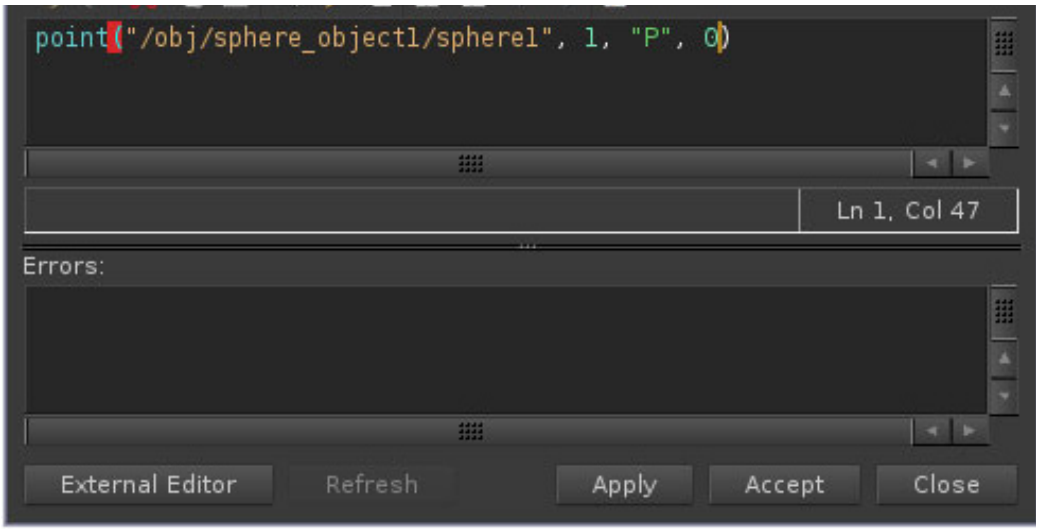
Finally, as you get deeper into building complex effects it will become necessary to use **expression functions**. While it is possible to call up certain information using a $VARIABLE, it is possible to tell your parameters what to do with these variables using functions. Functions are in turn small aliases which you may call upon to perform more advanced operations on the **arguments** that you feed into the function. For example, a channel reference: ch() is a simple function which will return the information that is found within a given channel/parameter. By typing ch(“tx”) into the Uniform Scale parameter of a sphere object, you will be able to tell the sphere to scale based on it’s transformation along the x-axis - “tx” in this case is the argument. There are many functions within Houdini, and it is possible to layer functions within functions to create intelligent, procedural systems.

WRITING EXPRESSIONS

It is usually possible to enter expressions directly into a parameter by simply typing into a field:

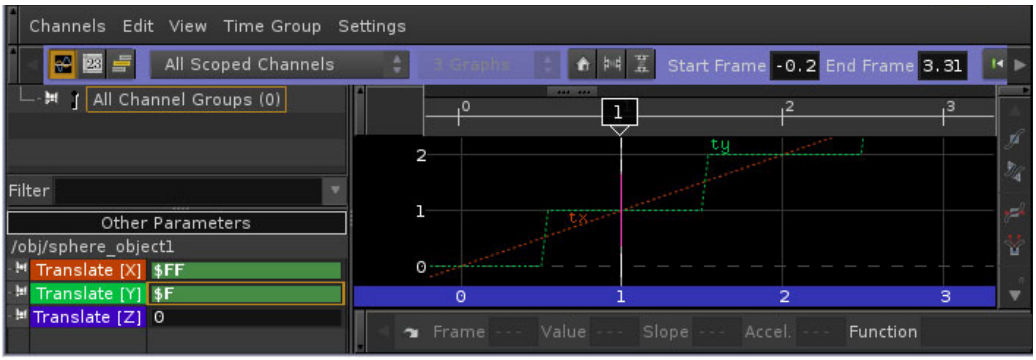


However depending on the complexity of your function, or the type of parameter (a toggle for example) you may instead opt to use the **expression editor**. The expression editor can be opened by right clicking on a parameter and selecting it from the menu, or by placing the mouse over the parameter and pressing **Alt - E**.



If you want to understand what an expression is doing then you can scope the parameter into the channel editor and view the expression in graph format. In the image below, you can see the difference between an animation curve created using $F (the current Frame) and another using $FF (the current "Floating" Frame).

You can scope a parameter by pressing **Shift - LMB** on a parameter, or **RMB-clicking** on the parameter and choosing **Channels and Keyframes > Scope Channels**. This will also bring up the channel editor with the curve in place.



EXPRESSION SYNTAX

Here are some of the main syntax elements which you will encounter again and again when using expression functions:

( )	**Brackets**	Brackets are used in order to list all of the arguments that will be contained within a function. Whenever a function is used, there must be an open bracket and a closed bracket. One of the easiest mistakes to make when creating expressions is to accidentally have the incorrect number of brackets.
" "	**Quotation**	When a string must be used as an argument inside a function, it must be contained within quotation marks. This can include parameter names, and file paths.
` `	**Back Ticks**	When an HScript expression has been entered into a string type parameter, it must be enclosed in backticks in order to be evaluated as an expression. For example, if you are attempting to use an expression function inside a file path, you will need to include the backticks on either end of your function.
' '	**Apostrophe**	Text inside apostrophes are not expanded. It may be necessary at times to use these characters inside strings to prevent a variable from being recognized as a variable.

VIDEO 01

DATA TYPES AND SIMPLE EXPRESSIONS.

TYPES OF DATA

There are four main types of data that exist within Houdini:

Float: Floating point numbers are single numerical values which may contain decimals i.e.: 15.5. It is possible to break this down further into **integers**, which do not contain decimals. An Alpha attribute/channel would be a good example of a float value.

Vectors: Three floating point values. These values can be used to represent positions, directions, normals or colors (RGB or HSV).

Strings: Strings are data which are not numbers, but text. A number is not represented as numerical in a string but as a text symbol. You can enclose strings in double quotes (“) or apostrophes (‘). Variables inside double-quotes are expanded. Strings inside apostrophes are not expanded. It is possible to create custom variables which make use of strings. We will demonstrate this at the end.

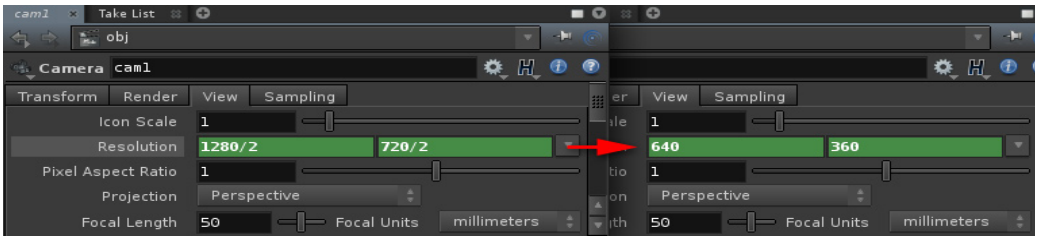
Matrix: Sixteen floating point values representing a 3D transformation matrix. We will not be discussing matrices in this lesson as it tends to be more mathematically advanced.

When one enters an expression into a parameter, it is possible to left-click the parameter in order to toggle display of the expression or the result of the expression in the UI.

SIMPLE MATH

`+ -`	Add, subtract
`* / %`	Multiply, Divide
`%`	Modulo returns the remainder after division has occured i.e. 4%3=1, 4%20=4
`^`	Raise to exponent i.e. `3^3=9`
`e`	Exponential notation is a concise way of representing large numbers with a minimal amount of digits, for example 3.2e-5 (equivalent to 0.000032).
`()`	Grouping

In the image below you can see how a simple math equation is used to reduce the resolution on a camera. The correct values are used by the camera and the math equation remains in place. This can be very helpful when you are exploring an idea and tweaking a parameter value.



COMPARISONS

You will often find yourself wanting to do comparisons between parameters and attributes. It is possible to do simple comparisons, where the statement returns 1 when true, and 0 when false. This is particularly useful for switches.

`< >`	Less than, greater than
`==`	inquire as to whether it is equal
`!=`	not equal to
`||`	or
`&&`	and

Expressions may be simple references to specific data using an alias, or variable. We access variables using $ followed by the variable name. There are several types of Variables including Global, Standard, Local, and Custom.

GLOBAL VARIABLES

Data that is accessible in all contexts. There are different types of **global variables**, which have various purposes.

Environment Variables: Useful for setting up your Houdini environment.

`$HIP`	This defaults to the directory containing the current Houdini file.
`$HIPNAME`	The name of the current .hip file
`$WEDGE`	The current **wedge** information
`$OS`	Operator String - contains the current OP's name
`$JOB`	A custom variable that determines where your jobs are located
`$HOME`	Your **Home** environment variable - specify in OS environment variables
`$ACTIVETAKE`	Contains the name of the current take
`$CH`	Current channel name
`$TEMP`	Where temporary (crash) files are saved - specify in OS environment variables

SUPPORT FILES: *02_Environment Variables*

PLAYBAR VARIABLES

There are various variables that are associated with time in Houdini:

`$F`	The current frame. This is very useful, especially for filename numbering. It allows us to introduce the element of time into our expressions. In the case of file naming we can add padding to `$F` `$F3` gives us a padding of 3 meaning that Frame 1 = 001 We can also use `$F` on a translation parameter tell it to move over time We can offset `$F` with simple math. `$F-1` tells frame 1 to return a value of 0

Certain elements, work in units of "Time" which refers to seconds rather than frames:

`$FPS`	Playback speed in frames/second (set in **Global Animation Options**)
`$T`	Current time in seconds

We can use `($F-1)/$FPS` to determine the current frame's time in seconds. This can be useful to know when we see parameters like **start time** on a particle network and want to be able to determine the **start frame**. By telling the Start Time to be `40/$FPS`, the particle simulation will begin on frame 40.

Particle and dynamic networks use time rather than frames because we often need subframe data:

`$FF`	Floating point frame number
`$SF`	Simulated frame number

These become important variables to remember when we adjust the over sampling. i.e.: if collision geometry isn't sampling between frames, we can have problems with accuracy therefore we want to over sample (sample more than once per frame).

It is important to cache out simulations using `$SF` for DOPs and `$FF` in POPs so that we keep subframe data.

It may also become essential when using longer expressions to use the Expression Editor (Alt + e) or right click menu on a parameter.

SUPPORT FILES: *03_Playbar_Variables*

VIDEO 02

ENVIRONMENT VARIABLES.

VIDEO 03

PLAYBAR VARIABLES.

Houdini
3D ANIMATION TOOLS

VIDEO 04

BUILDING A TIMECODE

OTHER PLAYBAR VARIABLES

`$NFRAMES`	The number of frames in the animation
`$RFSTART`	Frame number of the first frame shown in the playbar. The playbar can show a subset of the total number of frames, allowing you to focus on a particular section of a longer sequence
`$RFEND`	Frame number of the last frame shown in the playbar
`$FSTART`	Frame number of the first frame of animation (set in Global Animation Options)
`$FEND`	Frame number of the last frame of animation (set in Global Animation Options) `$RFSTART` and `$RFEND` control the subset of frames shown in the playbar
`$TLENGTH`	Total length of animation in seconds
`$TSTART`	Start time of animation in seconds
`$TEND`	End time of animation in seconds

You will recognize `$RFSTART` and `$RFEND` from the **flipbook** menu. They are used to flipbook the visible region in the playbar. These variables may also be usable if you want to render a time code into your animation.

More global variables can be found in the Houdini Documentation:
http://www.sidefx.com/docs/current/expressions/_globals

Note that these include mathematical constants, CHOP, ROP and COP variables.

SUPPORT FILES: *O4_Building_a_Time_Code*

VIDEO 05

STANDARD VARIABLES

STANDARD VARIABLES

Commonly used variables that are unique to specific contexts, however are common enough that they may be recognized across multiple contexts. Note that some variables such as $CR, $CG, $CB, $CA are actually attributes: Cd and Alpha. Attributes are data that are easy to lock down, and you can see that they exist if you MMB onto a node.

`$TX,$TY,$TZ`	Point Position
`$BBX,$BBY,$BBZ`	Point Position within bounding box (ranges 0-1)
`$YMAX,$YMIN`	The max/min values of the geometry on the Y-Axis. Note this goes for X,Y, & Z
`$CEX,$CEY,$CEZ`	Centroid of the input geometry
`$CR,$CG,$CB,$CA`	Diffuse Point Color, and Point Alpha
`$NX,$NY,$NZ`	Point Normal Direction
`$PT,$PR`	Point Number or Primitive Number
`$NPTS,$NPRIMS`	Total number of points or primitives

SUPPORT FILES: *O5_Standard_Variables*

VOLUMES AS ATTRIBUTES

Each volume primitive may be used to represent a set of data. This is important to note when working with fluid dynamics to understand what purpose the various fields serve.

`$X,$Y,$Z` Bounding Box Information in Volumes

SUPPORT FILES: *06_Volumes as Attributes*

VIDEO 06

VOLUMES AS ATTRIBUTES.

PARTICLE VARIABLES

There are some very commonly used Standard Variables that are usually associated with particles. Many of the standard variables can be found on the Point SOP.

`$VX,$VY,$VZ`	Point Velocity values
`$ID`	Particle ID
`$PSCALE`	Particle Scale
`$LIFE`	Percent of total life used (from 0 to 1)

More standard variables can be found in the Houdini Documentation:
http://www.sidefx.com/docs/current/nodes/sop/standardvariables

SUPPORT FILES: *07_Standard_Particle_Variables*

VIDEO 07

STANDARD PARTICLE VARIABLES.

LOCAL VARIABLES

Many variables that are common to one context are not necessarily common to another context. Not all variables are standard. In fact, some nodes have **local variables**, which will be listed in the help cards on a node-by-node basis.

On the Transform node for example:

`$GCX, $GCY, $GCZ` Centroid of the input group

SUPPORT FILES: *08_Local Variables*

VIDEO 08

LOCAL VARIABLES.

ATTRIBUTE SIGNATURES

Sometimes variables can also be attributes which will be stored in the geometry and you can access them further down the chain. Often you will want to access one of these variables across contexts. These variables however have signatures which do not necessarily allow immediate cross-context access. We can create our signature mapping using the Attribute Create node to get around this.

We can also use functions designed specifically for accessing attributes: `point()` `detail()` `vertex()` or `prim()`

SUPPORT FILES: *09_Attribute_Signatures*

VIDEO 09

ATTRIBUTE SIGNATURES.

Houdini
3D ANIMATION TOOLS

VIDEO 10

ATTRIBUTE CLASS

ATTRIBUTE CLASS:

While attribute signatures distinguish which attributes are readable in which contexts, we must also think about attribute class.

- Attribute signatures are invisible
- Attribute class is discoverable by MMB on a node
- Classes include point, primitive, vertex, and detail
- We can use the attribute promote sop to convert the classes from one to another
- We may want detail attributes to discover things like min or max values for geometry

SUPPORT FILES: *10_Attribute_Class*

VIDEO 11

CUSTOM ATTRIBUTES WITHIN DOPS

CUSTOM ATTRIBUTES WITHIN DOPS

While it can be necessary to use Attribute Create to create proper attribute signatures to work with existing attributes, it also is used to create custom attributes. In O5_Standard_Variables, we saw that Cd, and Alpha can be used to perform alternate operations.

Often pre-existing attributes are used due to ease of access, however it is also normal to create new attributes. Usually a custom variable is used because the pre-existing attributes must be used for their intended use and cannot be hacked for other purposes

Sometimes new attributes are created as it can allow for extra control in dynamic simulations. Many of the options available on a dynamic solver may be overridden with a custom attribute

SUPPORT FILES: *11_Custom_Attributes_Within_DOPs*

VIDEO 12

CUSTOM ATTRIBUTES WITHIN VOPS

CUSTOM ATTRIBUTES WITHIN VOPS

In the same way that attributes may overwrite parameters in DOPs, they can do this in VOPs. VOPs (VEX Operators) are a very powerful method for processing geometry in Houdini and are particularly useful for doing geometry displacements, or creating custom shaders.

Variables are easy to access in VOPs and use the variable name i.e.: Alpha not the $VARIABLE i.e.: $CA.

Parameter VOPs are typically used in shaders to access attributes. Import attribute can be used in VOPSOPs to acquire attributes from the geometry. Add attribute can be used in VOPSOPs to create an attribute that can be used elsewhere in your network.

SUPPORT FILES: *12_Custom_Attributes_Within_VOPs*

VIDEO 13

CUSTOM ATTRIBUTES

CUSTOM ATTRIBUTES

Custom attributes are usually used as masks/multipliers for other effects, i.e.: scaling noise, scaling blend shapes, or triggering particle emission.

Custom attributes (point and primitive) can be displayed by turning them on in the display options (d)

SUPPORT FILES: *13_Custom_Attributes*

FUNCTIONS

Functions are the core of expressions. They are snippets of text which tell Houdini to perform various operations based on the arguments you provide. Functions can be data fetching, data modification, or mathematical operations.

There are 382 available functions. These can be found either in the Houdini Textport using the exhelp command, or in the Help Documentation:

www.sidefx.com/docs/current/expressions

If none of these functions do exactly what you want, it is also possible to create your own functions. When creating your own functions, it is a good idea to explore using Python.

While it may seem intimidating with so many functions to learn, there is really only a small collection of expressions that you will use over and over. A good way to learn functions is to try using a few each day and see what they do.

There are often "related" functions. It is a good idea to investigate related functions at the same time. Often there are categories. i.e.: there are 34 DOP specific functions. If you do a lot of work in DOPS, it would be prudent to learn what these do and think of possible uses.

EXAMPLES OF FUNCTIONS THAT WE'VE USED SO FAR

`dopobjscreatedby()`	Fetches an object from a dopnet
`opinputpath()`	Fetches the name of the node plugged into the current one
`ch()`	Returns the float value of the channel in the function
`constant(),bezier()`	Used in key frame animation
`trunc(),int()`	Converts number to an integer by truncating a fractional part
`padzero()`	Adds frame padding in instances where $F3 is insufficient
`fit(),fit01()`	Returns a number which is fit into the supplied range
`rand()`	Generates a random number based on the seed value
`if()`	Used for complex comparisons between values
`point(),vertex(), prim()`	Used to return the attribute value on a given node
`distance()`	Used to calculate the distance between two points

OTHER COMMON FUNCTIONS

Be sure to look into the functions below as they are useful/common functions that you will eventually want to use in your files. With the below functions, as well as the ones we've already used in your arsenal, there will be little that you cannot accomplish.

Data Manipulation

`abs()`	Returns the absolute float value. i.e.: `abs(-7)=7`
`clamp()`	Prevents numbers from going outside a specified range
`ceil()`	Rounds fraction up to nearest integer
`floor()`	Rounds fraction down to nearest integer
`frac()`	Returns the fractional component of the number i.e.: `frac(9.7)=0.7`

Math

`pow()`	Mathematical operation used to do power operations i.e.: `pow(2,5) = 2*2*2*2*2`, whereas `pow(2,0.5)` = the sqrt of 2
`length()`	Returns the length of a vector i.e. `length($VX,$VY,$VZ)` = speed
`cos(), sin(), tan()`	Mathematical functions for making wave-like animation i.e.: `sin($F)`
`noise(), turb()`	Functions used to generate noise

Houdini 3D ANIMATION TOOLS

VIDEO 14

CONTINUED LEARNING CONT'D

Data Retrieval

`pic()`	Returns a color value from a compositing node based on supplied UV's i.e.: `pic("/img/img1/color1", $MAPU, $MAPV, D_CLUM)` would return the luminance value from a picture.
`opfullpath()`	Returns the full path of a node. i.e.: `opfullpath("../node") = /obj/object1/node`
`chs()`	Returns the string value of the channel in the function
`chramp()`	Returns the specified channel from a ramp
`bbox()`	Returns min, max, and size values for bounding box of a node. i.e.: `bbox("../node", D_YSIZE)`
`centroid()`	Returns the centroid of a given node, i.e.: `centroid("../node", D_Y)`
`npoints()`	Returns number of points in the specified node i.e.: `npoints("../node")`

Forced Evaluation

`eval()`	Used in the event that a variable itself contains an expression which you need to evaluate. i.e.: `set foo = 1+2 `eval($foo)``

Copy Stamping

stamp()	One of the most useful functions. Used to perform various operations on a per point/attribute/copy basis using the copy operator.

VIDEO 15

FINAL EXAMPLE

USING ATTRIBUTES TO CREATE EXPRESSIONS

Depending on the functionality that you need, sometimes you may wish to make use of Detail attributes to create strings that we can later call as an expression. This can be a useful way to semi-automate things that would otherwise be very difficult to manage.

A prime example of this is being able to select particles in the viewport, but then delete them based on their existing $ID attribute.

SUPPORT FILES: *15-18_Delete_By_ID*