1 VOPs (VEX OPerations)

1 INTRODUCTION

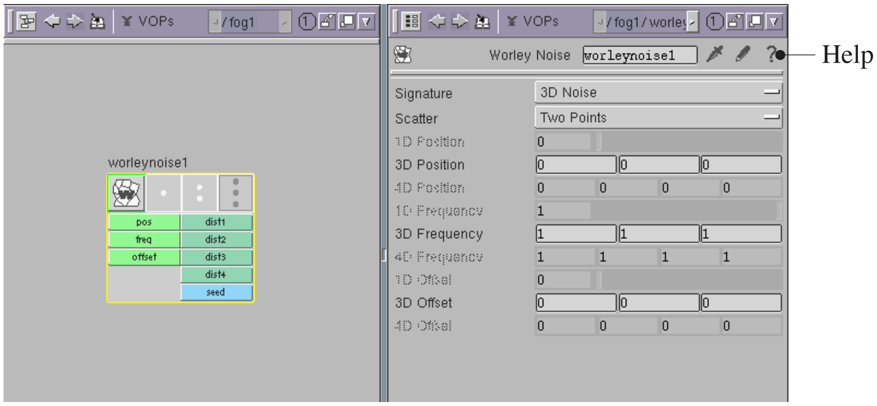
1.1 VOPS

VOPs provide you with a visual programming method for writing OP elements within Houdini, such as shaders and surface deformers. The underlying language used by all VOPs is Houdini's VEX language.

Instead of writing code, VOPs allow you to visually hook nodes containing snippets of VEX code together to define not only shaders, but all sorts of other logic within Houdini – they simplify the process of writing code for you, and provide you with a rich graphical interface to the underlying Power of Houdini's engines.

Many of the VOPs contain subnetworks of simpler VOPs, allowing you to understand, learn, and modify a function's behaviour easily. One such example is the Fire VOP, which includes Noise, Spline and Mix VOPs (among others). Open it to investigate how it's wired up.

WHERE TO LOOK FOR OP HELP



Signature	3D Noise			
Scatter	Two Points			
1D Position	0			
3D Position	0	0	0	
4D Position	0	0	0	0
1D Frequency	1			
3D Frequency	1	1	1	
4D Frequency	1	1	1	1
1D Offset	0			
3D Offset	0	0	0	
4D Offset	0	0	0	0

HELP

Every VOP has built-in Help. For VOPs which you don't find here, click on the '?' icon in the VOP's parameters for up-to-date help.

1.2 VOPS VS VEX

Unlike other OP types in Houdini, VOPs simply generate a piece of VEX code for you. Basically, VOPs provide a graphical wrapper for the VEX language. The whole of a VOP network comprises the 'circuit' of a VOPnet, which is roughly interchangeable with code that you can write within VEX. VOPs provide you a simpler graphic way to generate the code that would normally have to be written by hand in VEX.

1.3 VOP CONTEXTS

VOPs Operate in multiple Contexts – that is, they can use more or less the same code to work on different sorts of data. For example, parts of the VOP network you create to transform pixel data in a COP context could be reused to transform point positions in a SOP context. This is discussed further in: *VOP Contexts* p. 196.

MAIN VOP CONTEXTS

- VEX Compositing Filter
- VEX Compositing Generator
- VEX Displacement Shader
- VEX Fog Shader
- VEX Geometry Operator
- VEX Image3D Shader
- VEX Light Shader
- VEX Motion and Audio Operator
- VEX Particle Operator
- VEX Shadow Shader
- VEX Surface Shader

1.4 WHAT YOU NEED IN A VOP NETWORK

OUTPUT VOP

Any VOP Network needs at least one Output VOP. The Output VOP is the particular result to which you wire all your workings within a VOP network to provide a result. The inputs you provide to an Output VOP come from the following sorts of VOPS:

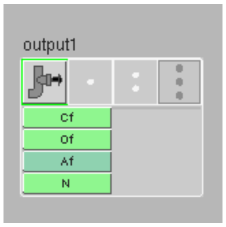
- Global Variables
- Parameters
- Constants
- Other VOPs

You need to wire these into one or more of the inputs of the Output VOP.

The only exception to this is the Parameter VOP – which implicitly sends its outputs to the Output VOP, even if you don't see a physically wired connection between them.

example

If the result you are seeking is a shader – this output VOP will have inputs for Cf, Of, Af, and N – Surface Colour, Surface Opacity, Surface Alpha/Transparency, and a Normal – for which you will have to provide the inputs.



The inputs of the Output VOP depend on the context. Thus, the COP or CHOP context will require different inputs for you to fill.

1.5 WHAT MAKES VOPS UNIQUE

VOPS ARE NOT ANIMATABLE

They just generate code to be used by, say – a SHOP. By wiring Parameter VOPs into the VOPnet, you expose those parameters, say, at the SHOP level, where they *can* be animated.

POLYMORPHISM

The polymorphic property of VOPs allows them to run the same code on multiple data types. One such example is the Mix VOP (which calls several flavours of the 'lerp' VEX function). Mix allows floats, vectors and vector4's on its inputs and blends the inputs in a linear fashion.

VOPS ARE STRONGLY TYPED

VOP inputs are strongly typed. A 'typed input' means simply that: you have to hook integer outputs to integer inputs (a single number, no decimals); float outputs to float inputs (numbers with decimal points); and vectors to vectors (a group of three numbers indicating a direction), and so on. The inputs and outputs are colour-coded on the VOPs themselves to make this typing more apparent.

colour legend

Blue	Integer and Toggle.
Light Green	Vector
Dark Green	Float, etc.

matching types

You can connect disparate types in VOPs using one of the Convert VOPs:

- Degrees to Radians
- Float to Integer
- Float to Matrix

- Float to Vector
- HSV to RGB
- Integer to Float
- Matrix to Float
- Vector to Float

There are more. These conversion VOPs allow you match the different types that VOPs expect in their inputs.

VOPS ACCEPT A VARIABLE NUMBER OF INPUTS

Whereas other OP types accept only a particular number of inputs, VOP nodes can accept any number of inputs. The types of inputs and outputs VOPs provide are analagous to variables – for example, in VEX, you have the variable *Cs* – which stands for surface colour. In VOPs, you have these variables listed explicitly as inputs and outputs on the VOP tiles. One such example is the Add VOP.

1.6 MAIN FAMILIES OF VOPS TO KNOW

PRIMARY WORKFLOW VOPS TO KNOW

Output VOP	Everything connects into here. It cannot be deleted or duplicated.
Parameter VOP	Allows you to specify parameters for input into your VOP Network, and the user of that network (e.g. a SHOP shader).
Constant VOP	When you don't need a value to change outside the VOPnet (like parameters do), this allows you to define a constant value for use within the VOP network. Constants aren't visible outside of VOPs.
Global Variables	Allow you to access all the predefined variables available to you in a given context, whether they are writable (like the Output VOP), or only readable.

OTHER CATEGORIES OF USEFUL VOPS

- Materials (Wood, Marble, Fire, Bricks, etc.)
- Patterns (Noise)
- Displacements
- UV Shading
- SuperMat (Phong Blinn Cook & Texture Maps)

1.7 SHOPS WITH VOPS

One of the places where you can use VOPs is within SHOPs (i.e. shaders). Once you place a SHOP – If the SHOP is comprised of VOPs instead of VEX code, you'll be able to select *Edit VOP Network* from the SHOP's pop-up menu.

If not – you can select *Create New VOP Type*... from the menu, and it will embed the SHOP into a new VOP which you can save to disk.

Once you've done this, then, you can place the created SHOP in a VOP network, and have access to its Inputs and Outputs from within a VOP network.

1.8 COPYING AND PASTING PARAMETER VOPS

When you copy and paste a Parameter VOP that has already been setup (e.g. Image(string)) – it results in a Parameter VOP that is partially uneditable – the parameter type is inaccessible.

The reason for this is that if you have two Parameter VOPS that both use the same Parameter Name (let's say 'uv'), then only one of them gets to actually specify the characteristics of 'uv'.

Specifically, whichever Parameter VOP was created first gets to define what 'uv' is like. The second and third Parameter VOP with that parameter name does not get to specify the label and data type, etc. Why? Suppose you had two Parameter VOPs that both defined 'uv'. One of them could define it to be a float, and the other define it to be a string. In the resulting VEX code 'uv' would be declared twice:

```
shader foo(int uv = 0; vector uv = "hi")
```

which yields a syntax error. Or 'uv' would be declared only once, but in the code it would get used as if it were a float and a string. Again, this yields a syntax error.

The other approach would have been to enforce that each parameter name must be unique. So copying and pasting a 'uv' Parameter VOP would get you a second Parameter VOP with the parm name 'uv2'. This was not done because it can be very convenient (and help the organization of your network) to be able to access the same parameter in several locations in your network without having to have all the wires leading back to the same single Parameter VOP.

2 VOP CONTEXTS

These are the same contexts that VEX operates with.

2.1 INTRODUCTION

There are several VOPs which are common to all OP contexts. These functions evaluate a channel (or parameter) and return its value. When evaluating string parameters at a specified time, the time value must be the same for all evaluations of the parameter. If the time is not constant over the all the points/pixels being evaluated, results are not predictable.

2.2 THE COP CONTEXT

The COP evaluation works by setting either the R, G, B, A or the C1,C2, C3,C4 variables. The variables are initialized to the color of the first input (or zero for R, G, B and 1 for A if there is no input). It then calls the COP function. If any of the variables are modified, then they will be used to determine the new color for the current pixel.

The RGBA variables are for programmer convenience; they read and write directly to the Color and Alpha planes. Reading from RGB and A will give you the value of the first input's color plane and alpha plane. If one or both of these planes do not exist, 0 will be returned. If you are writing to R, G, B or A and either Color and/or Alpha doesn't exist, the planes will be created for you. If you only write to R, a full Color plane will be created, but with the G and B channels set to 0.

The channel variables, C1 to C4, are more flexible. They allow you to write to any plane's channels. If a channel of a plane does not exist and you write to it, nothing happens. For example, when cooking the Alpha plane (with only 1 channel), if C2, C3 or C4 is written to, the data is lost; only C1 is valid. However, no error occurs, since it is possible to have planes with different sizes in the same sequence. You can read from a channel variable in exactly the same way as reading from the R,G,B and A variables. The channel variable will be filled with the pixel values from the plane in the first input corresponding to the one being cooked.

You should not mix writing to R,G,B,A and C1,C2,C3,C4 in the same VOP function. While it will not harm anything, it can lead to confusing results if you are writing different things to R and C1 and the function is cooking the Color plane.

GLOBAL VARIABLES – COPS

Variable	Type	Read-Write	Description
XRES	int		The horizontal resolution of the image being processed (in pixels).
YRES	int		The vertical resolution of the image being processed (in pixels).
AR	float		The aspect ratio (width to height).
IX	int		Horizontal position of the pixel currently being shaded. In the range of (0, XRES-1)
IY	int		Contains the vertical position of the pixel currently being shaded. This will be in the range of (0, XRES-1).
X	float		Contains the horizontal location of the center of the current pixel being shaded in the range 0 to 1. Zero being the left hand side of the image, and 1 being the right hand side.
Y	float		Contains the vertical location of the center of the current pixel being shaded in the range 0 to 1. Zero being the bottom of the image, and 1 being the top.
H	float		Contains the hue of the current pixel. The hue is expressed as a floating point number between zero and one. HSV is calculated based on the Color plane (C), not the current plane.
S	float		Contains the saturation of the current pixel. The saturation is expressed as a floating point number between zero and one.
V	float		Contains the value (or intensity) of the current pixel. The value is expressed as a floating point number between zero and one.
R	float	•	Contains the value of the red channel of the current pixel. The value is expressed as a floating point number between zero and one. If the COP does not have a color plane (C), one will be created.
G	float	•	Contains the value of the green channel of the current pixel. The value is expressed as a floating point number between zero and one. If the COP does not have a color plane (C), one will be created.

B	float	•	Contains the value of the blue channel of the current pixel. The value is expressed as a floating point number between zero and one. If the COP does not have a color plane (C), one will be created.
A	float	•	Contains the value of the alpha (transparency) channel of the current pixel. The value is expressed as a floating point number between zero and one. If the COP does not have a color plane (C), one will be created.
C1 - C4	float	•	Contains the value of the plane component 1, 2, 3 or 4. Unlike the R,G,B or A variables, these components represent the current plane's values.
PNAME	string		Contains the name of the current plane being cooked.
PI	int		Contains the index of the current plane being cooked.
PS	int		Contains the size of the current plane being cooked (the number of channels).
AI	int		The array index of the current plane that is being cooked (from 0 to AS-1).
AS	int		The array size of the current plane.
NP	int		Returns the number of planes in the image (ie, for a RGBA image, this would be 2 since the COP must contain two planes, Colour and Alpha).
NI	int		Returns the number of images in the sequence (i.e. a sequence with a frame range of 1-19 would have 20 images in it).
F	int		The frame number of the current image.
SF	int		The starting frame of the current sequence.
EF	int		The ending frame of the current sequence.
I	int		The index number of the current image, which always starts at zero for the first image in the sequence.
TIME	float		The time of the current image.
TINC	float		The time increment between frames at the global frame rate.
FR	int		The frame rate of the current sequence.

2.3 THE POP/SOP CONTEXT

The POP and SOP contexts have the same variables and functions. Both of these contexts allow processing of point attribute data. With the POP context, the points are typically used in a particle system, however, this is not a requirement of the context. In general, a function written for POPs will work as a SOP or vice versa.

Export variables in the POP/SOP contexts will cause new attributes to be created on the geometry. The attribute will have the name and size of the export variable.

GLOBAL VARIABLES – POP/SOP

Variable	Type	Read-Write	Description
ptnum	int		Contains the point number of the point being processed.
Npt	int		Contains the total number of points geometry.
Frame	float		Contains the current frame. This may not be an integer value.
Time	float		Contains the current time (in seconds).
TimeInc	float		Contains the time increment for particle simulations. In the SOP context, it contains the time increment between frames.
P	vector	•	Contains the position of the current point.
v	vector	•	Contains the velocity of the current point.
accel	vector	•	Contains the acceleration of the current point.
Cd	vector	•	Contains the colour (RGB) of the current point.
id	int	•	Contains the value of the id attribute. ***Warning!*** If this value is modified, it is possible to generate duplicate id's for particles which can cause bad problems.
age	float	•	Contains the age associated with the current point. This represents how many seconds a particle has been alive.
life	float	•	Contains the expected lifetime (in seconds) of the current point. It is possible that a particle may die earlier than expected (if a collision or some other event occurs).
pstate	int	•	Contains the state of the current particle. This is an integer bit field which has the following bits defined (see Notes, below).

notes

pstate – The state of the current particle.
This is an integer bit field which has the following bits defined:

- 0x01 - The particle is a ‘primary’ particle (not birthed off an existing particle).
- 0x02 - The particle will die before the next frame.
- 0x04 - The particle is flagged as stopped.
- 0x08 - The particle has collided
- 0x10 - The particle is stuck to static or moving geometry.
- 0x20 - The particle is associated with a rigid body dynamic simulation.
- 0x40 - The particle is currently active
- 0x80 - The particle motion is overridden by a CHOP.

The bit-field associated with this variable may change in the future.
Please see *$HFS/houdini/vex/include/pop.h* for the latest info.

2.4 THE CHOP CONTEXT

The CHOP context allows users to change values of channels in a CHOP.
Each CHOP function works on a single sample of a single channel of a CHOP.

GLOBAL VARIABLES – CHOPS

Variable	Type	Read-Write	Description
V	float	•	Contains the value of the current sample. This variable should be set to the new value by the function. The variable is initialized to the value of the first input’s channels.
I	int		Contains the index or sample number of the current channel.
S	int		Contains the index of the start of the current channel. This is the index of the first sample.
E	int		Contains the index of the last sample (end sample).
SR	float		Contains the sample rate for the channel.
L	int		Contains the length of the channel (total number of samples).
C	int		Contains the channel number for the current channel. When processing multiple channels, this is the index of the channel currently being evaluated.
NC	int		Contains the total number of channels the CHOP will affect.

2.5 THE 3D IMAGE CONTEXT

The Image3D context is used by the stand-alone program i3dgen to generate 3D texture images. In turn, these 3D texture images may be used by the texture3d() function calls in VEX to efficiently evaluate the 3D texture images.

Any export variables in the image3d context will cause additional channels to be created in the 3D texture map.

GLOBAL VARIABLES – IMAGE 3D

Variable	**Type**	**Read-Write**	**Description**
P	vector		Contains the position being evaluated.
density	float	•	Specifies the value of the density channel at point P.

3 SHADING CONTEXTS

3.1 INTRODUCTION

The shading contexts share many common attributes. Each context represents a different stage in the rendering pipeline. Displacement shading is done first, followed by surface shading and then fog/atmosphere shaders are run. During surface and fog shading, light and shadow shaders may be run in order to compute illumination.

3.2 COMMON GLOBAL VARIABLES

The global variables which are used in all shading contexts are:

Variable	**Type**	**Description**
Cf	vector	The final color for the surface. The vector represents the RGB color for the surface.
Of	vector	The final opacity for the surface. A value of {1,1,0} will be opaque in red/green, but let through blue light from behind.
Af	float	The final alpha for the surface. This is the value which is placed in the alpha channel of the output image.
P	vector	The position of the point on the surface being shaded. In light or shadow shaders, the P variable contains the point on the light source.
Pz	float	The z component of the point being shaded.
Ps	vector	In light & shadow shaders, this is the position of the point on the surface being illuminated.
I	vector	The direction from the eye to the point being shaded. This may or may not be a normalized vector.
Eye	vector	The position of the eye.
s	float	The parametric s (sometimes also called U) coordinate of the surface being shaded.
t	float	The parametric t (sometimes called V) coordinate of the surface being shaded.
dPds, dPdt	vector	The change in position with respect to the parametric coordinates s, t
N	vector	The shading normal for the surface.
Ng	vector	The geometric normal for the surface. This normal represents the "true" normal of the surface being shaded. For example, with Phong shading, the N variable represents the interpolated normal, while the Ng variable represents the true polygon normal.
Cl	vector	The Light Colour.

L	vector	The vector from the point on the surface to the light source. The length of this vector represents the distance to the light source. Please see *Initialization of the L Variable* p. 205 for further information.
Lz	vector	The z-axis in the space of the light. This is a unit vector.

3.3 GLOBAL VARIABLE ACCESS

Each context is responsible for setting or modifying different variables. For example, the displacement context can modify the position of the surface being rendered, while the light context is responsible for setting the color of the light source. The Read/Write access of the variables for each context can be described below.

Variable	**Displace**	**Surface**	**Light**	**Shadow**	**Fog**
Cf	-	Write	-	-	Read/ Write
Of	-	Write	-	-	Read/ Write
Af	-	Write	-	-	Read/ Write
P	Read/ Write	Read/ Write *	Read	Read	Read
Pz	Read	Read	-	-	Read
Ps	-	-	Read	Read	-
I	-	Read	Read	Read	Read
Eye	-	Read	Read	Read	Read
s	Read	Read	Read	Read	Read
t	Read	Read	Read	Read	Read
dPds, dPdt	Read	Read	Read	Read	Read
N	Read/ Write	Read/ Write	Read	Read	Read
Ng	Read				Read
Cl	-	Read/ Write	Write	Read/ Write	Read/ Write
L	-	Read	Read/ Write	Read	Read
Lz	-	-	Read	Read	-

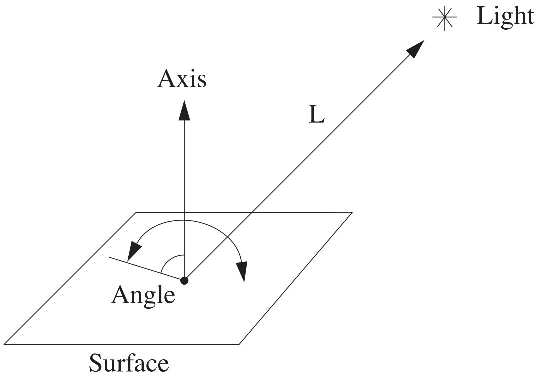
** Although P is modifiable in the surface context, at the current time, changing P will not affect rendering of the surface, only shading.*

3.4 SURFACE SHADING CONTEXT

The surface shading context's purpose is to set the final colour, opacity and alpha of the surface being rendered. If the Of and Af varaibles are not set, they will default to 1. If the Af variable is not set, it will resolve to average of the componets (see the Average VOP). It is possible to set the Af variable to any arbitrary value, making it possible to build matte/cutout shaders.

Inside the surface context, it is possible to loop through all light sources in the scene computing their illumination and shadows. This is done using the illuminance() construct. This creates will loop through all light sources, calling the light shader for each light source to ensure that the Cl and L variables are set properly. The shadow shader will not be called unless specifically requested. However, once the shadow shader has been called, the value of Cl will be changed for the duration of the surface shader. The shadow shader is automatically called when using any of the built-in lighting calls.

ILLUMINANCE



The default value for the axis is the surface normal. The default value for the axis is PI/2.The default value for the light mask is LIGHT_DIFFUSE | LIGHT_SPECULAR (please see *shading.h* for the light definitions). The Illuminance Loop VOP will loop through all light sources which meet the criteria dot(L, axis) > cos(angle).

3.5 DISPLACEMENT SHADING CONTEXT

Displacement shading can be used to move the position of the surface before the surface gets rendered. It is intended as a mechanism to add fine detail to a surface, not as a modelling technique. After moving a surface, you must typically re-compute the normals of the surface. For surfaces which have interpolated normals (i.e. polygons which are smooth shaded), the Shading Normal VOP can be used to approximate what the 'smooth' displaced normal will be.

3.6 LIGHT SHADING CONTEXT

Light shaders will get called from surface or fog shaders to compute the illumination from a given light source. The light shader can be invoked using the Illuminance Loop VOP.

3.7 SHADOW SHADING CONTEXT

Shadow shaders will get called from surface or fog shaders to occlude the illumination from a given light source. The light will already have been computed by calling the light shader. The function of a shadow shader is to modify the Cl variable. Typically, the light will be occluded, causing the Cl variable to decrease in intensity. However, it is possible to create ‘negative’ shadows, and increase the illumination due to occlusion.

The shadow context is typically used to separate illumination from expensive raytracing calls (like the Fast Shadow and Filter Shadow VOPs). This allows fog shaders (or surface shaders) to bypass shadow testing.

3.8 FOG SHADING CONTEXT

A fog shader is responsible for modifying the Cf, Of or Af variables after the surface shader has completed its shading. It is possible to use Illuminance Loop VOP inside of fog shaders.

3.9 INITIALIZATION OF THE L VARIABLE

In the Light and Shadow contexts, the P variable represents the position of the light source and the Ps variable represents the position of the surface point being shaded.

In Houdini, lights can have either perspective or orthographic projections. An orthographic light in Houdini is used to represent an infinite (or very distant) light source which has all the light raysparallel with each other. A perspective light acts more as a point light source.

When a perspective light shader runs, the L variable is initialized as follows:

```
L = P - Ps;
```

Orthographic lights, on the other hand, are initialized so that the direction of L is the same for each light ray eminating from the light source.

```
L = Lz * dot(Lz, P - Ps);
```

Where Lz represents the normalized "z-axis" of the light source (i.e. a unit vector pointing down the z-axis in the space of the light). Thus, the scale of the L variable will be the orthographic distance from the plane of the light source and the surface point being shaded.

In a light shader, it is possible to change the L variable to any value you choose. However, this is how the L variable is initialized.

3.10 SPECIAL VARIABLES

There are several ‘special’ variables used in the shading contexts. These are specified as parameters to shaders. Typically, these are export parameters. You can find reference to these in: *VEX Surface Shader:VOPs > Shading > Lighting Model > Diffuse; Specular.*

3.11 TRANSFORM SPACES

One of the key features of VEX shading is the concept of spaces. There are at least two or three transform spaces defined for each context. Often, shading it is important to transform a position or vector in one space to a different space. This is done using the Space Change VOP (among others). However, it is important to know what space variables are in and to which space the transform functions will take a variable.

In all VEX contexts, global variables are in "world" space. In the mantra renderer, this space is defined with the camera at the origin pointing down the positive Z axis. Each shader has a local "object" space associated with it. This is the space which defined as the object at the origin of the space. Displacement, Surface and Light shaders also have a special "NDC" (Normalised Device Coordinate) space associated with them.

SPACE TYPES

Context	Space	Description
Displacement Surface	World	The space which has the camera at the origin, looking down the positive Z axis.
Displacement Surface	Shader	Often, this is the same as object space. However, in Houdini, it is possible to define different spaces for displacement/surface shading. Typically, this space should be used for shading purposes.
Displacement Surface	Object	This is defined as the space which has the object at its origin. This is basically the inverse of the object's transform.
Displacement Surface	NDC	This is a special "Normal Device Coordinate" space. A position in world space can be converted to NDC space by calling the: *To NDC VOP*. This will transform the X & Y components of the position so that {0,0,z} will be the bottom left hand corner of the image plane as viewed from the camera. The top right hand corner will be represented by {1,1,z}. The Z component of the position remains unchanged by the NDC transformations. Given a point in NDC space, it is possible to find it's world space position by calling the *From NDC* or *UV Spacechange* VOPs.
Light Shadow	World	The space which has the camera at the origin, looking down the positive Z axis.
Light Shadow	Shader	For Light & Shadow contexts, this is the same as object space.
Light Shadow	Object	This is defined as the space which has the light at the origin, pointing down the positive Z axis.
Light Shadow	NDC	Since light sources in Houdini have the same attributes as cameras, NDC space is defined for light sources as well. To transform a position into the NDC space for the light source, the object space position should be used in the toNDC space call. The Z component of the object space position will remain unchanged. Given a point in NDC space, it is possible to find it's world space position by calling the From NDC VOP.
Fog	World	The space which has the camera at the origin, looking down the positive Z axis.
Fog	Shader	For fog shaders, this is the same as object space.

Fog	Object	This is defined as the space which has the fog object at the origin, pointing down the positive Z axis.
Fog	NDC	This is a special "Normal Device Coordinate" space. A position in world space can be converted to NDC space by calling the To NDC VOP. This will transform the X & Y components of the position so that {0,0,z} will be the bottom left hand corner of the image plane as viewed from the camera. The top right hand corner will be represented by {1,1,z}. The Z component of the position remains unchanged by the NDC transformations. Given a point in NDC space, it is possible to find it's world space position by calling the From NDC VOP.

3.12 OPACITY VS ALPHA

In the surface shader context, there are two separate variables to control transparency. Of and Af are related, but represent different quantities. The Of variable represents the opacity of the surface. For example,

```
Of = {1, 0, 0};
```

will make the surface opaque in red, but pass through green and blue light from behind. This will give you a cyan colored filter. The opacity is used when mantra is resolving surface colors.

The Af variable is used to decide what value gets put into the alpha channel of an RGBA image. If the Af variable is not set by a shader, it is automatically assigned by using:

```
Af = avg(Of)          (use an Average / Average Coordinate VOP)
```

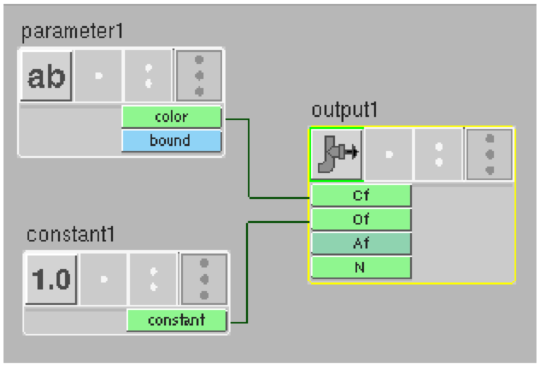
However, because this variable is available, it is possible for a shader to decide what value gets put in the alpha channel of the output image. This makes it possible to write shaders like "matte" or "shadowmatte". See the Matte and Shadowmatte VOPs in the Material toolbar.

EXAMPLE

For example, a VEX shader with the following parameters simulates a blue screen:

```
surface bluescreen(vector clr=0)
{
  Cf = clr;
  Of = 1;
  Af = 0;
}
```

Here is the same shader represented with VOPs:



The colour of the surface is set to black (by default). The opacity is 1, meaning that this surface will occlude other surfaces in the scene. However, the alpha output to the image will be 0. This means that if this image is composited over an other image, the background image will blend appear where this surface was rendered. An example use of this would be to create a window in a wall without having to modify the geometry.

It is also possible to set the Af variable to be something other than 1, meaning that the background image will appear ‘shadowed’ where the matte surface is rendered. This is the theory behind the ‘shadowmatte’ shader.

2 VOPs

1 ABSOLUTE VOP

1.1 DESCRIPTION

This operator computes the absolute value of the argument. It can be used to convert negative values into their positive equivalent.

1.2 SEE ALSO

- Complement
- Multiply

2 ADD VOP

2.1 DESCRIPTION

This operator outputs the sum of its inputs.

The first input can be an integer, float, vector, vector4, matrix3, or matrix. The allowed data types of subsequent inputs depend on the data type of the first input. For example, if the first input is a float, subsequent inputs can be either floats or integers. The output data type is always the same as the data type for the first input.

2.2 INPUTS

INPUT NUMBER 1...N

The input values to be added together.

NEXT INPUT

Where the next input value should be connected.

Any number of inputs can be specified.

2.3 OUTPUTS

COMBINED VALUE

The sum of all the input values.

2.4 SEE ALSO

- Add
- Constant
- Complement
- Divide, Multiply
- Subtract

3 ALIGN VOP

3.1 DESCRIPTION

This operator computes a matrix representing the rotation around the axes normal to two vectors (their cross product), by the angle which is between the two vectors. The resulting matrix maps the first vector onto the second. If both vectors have the same direction, the result will be an identity matrix. If the vectors are opposed, the rotation is undefined.

3.2 SEE ALSO

- Direction
- Space Change
- Look At
- Orient
- Rotate

4 AND VOP

4.1 DESCRIPTION

This operator performs a logical "and" operation between its inputs and returns 1 (if all inputs are non-zero) or 0 (if at least one input is zero).

Typically, "and" is used as an input to conditional operators such as If and While. All inputs and the output data type are integers.

4.2 INPUTS

INPUT NUMBER 1...N

The input values to be combined together.

NEXT INPUT

Where the next input value should be connected.

Any number of inputs can be specified.

4.3 OUTPUTS

COMBINED VALUE

The logical "and" combination of all inputs.

4.4 SEE ALSO

- Or
- Not
- Compare
- If
- While

5 AVG VOP

5.1 DESCRIPTION

This operator outputs the average of its inputs.

The first input can be an integer, float, vector, vector4, matrix3, or matrix. The allowed data types of subsequent inputs depend on the data type of the first input. For example, if the first input is a float, subsequent inputs can be either floats or integers. The output data type is always the same as the data type for the first input.

5.2 INPUTS

INPUT NUMBER 1...N

The input values to be averaged together.

NEXT INPUT

Where the next input value should be connected.

Any number of inputs can be specified.

5.3 OUTPUTS

COMBINED VALUE

The average of all the input values.

5.4 SEE ALSO

- Average Vector Component
- Maximum
- Minimum

6 COMPARE VOP

6.1 DESCRIPTION

This operator compares two values and returns true or false. Both operands must have the same type.

6.2 SEE ALSO

- And
- If
- Or
- Not
- While

7 CONSTANT VOP

7.1 DESCRIPTION

This operator outputs a constant value of any VEX data type.

Use this operator type to create a value that is not going to change between different instantiations of the VEX operator defined by the VOP network. For example, if a shading function required multiplying a value by 2, or finding the cross product of a vector and { 0, 1, 0 }, the values 2 and { 0, 1, 0 } would most easily be represented by a Constant VOP. If the value 2 or { 0, 1, 0 } might change depending on the properties of the material being shaded, a Parameter VOP should be used instead.

Unlike this operator, the Parameter VOP allows the specification of a different value for each instantiation of the OP type that uses this VOP network. For example, if you want the surface color to be accessed by a Surface shader (SHOP) using this VOP network, make the color a Parameter VOP instead of a Constant.

7.2 PARAMETERS

CONSTANT TYPE

Specifies the VEX data type of the value output by this operator. In some cases there is more than one way to represent a VEX data type. For example, the vector type can be represented by 3 float values, or a single color value. This parameter also specifies how the VEX value should be represented. This determines which parameter to use to define the value output from this operator.

CONSTANT NAME

The name of the constant defined by this operator, both in the generated VEX code and on the operator tile's output.

CONSTANT LABEL

The label that pops up when the cursor moves over the operator tile's output.

DEFAULT VALUES

Depending on the constant type selected, it represents the value of the constant in the VEX function.

7.3 OUTPUTS

CONSTANT VALUE

The value specified by the parameters of this operator.

7.4 SEE ALSO

- Global Variables
- Parameter
- Shading Layer Parameter

8 DIVIDE VOP

8.1 DESCRIPTION

This operator outputs the result of dividing each input value by the next.

The first input can be an integer, float, vector, vector4, matrix3, or matrix. The allowed data types of subsequent inputs depend on the data type of the first input. For example, if the first input is a float, subsequent inputs can be either floats or integers. The output data type is always the same as the data type for the first input.

8.2 INPUTS

INPUT NUMBER 1...N

The input values to be divided.

NEXT INPUT

Where the next input value should be connected.

Any number of inputs can be specified.

8.3 OUTPUTS

COMBINED VALUE

The result of dividing each input by the next.

SEE ALSO

- Add
- Complement
- Multiply
- Multiply Constant
- Subtract

9 FORPOINTS VOP

9.1 DESCRIPTION

This operator is only available in Image3D VOP networks. It contains other VOP operators, and executes the code for the contained operators once for each metaball or particle that contains the position passed in.

If the distance is specified, all metaballs and particles within the distance of the point specified will be iterated through. The distance parameter is optional and may result in slower execution of the shader.

Inside the point loop, the Metaball Density and Metaball Attribute operators may be used to query the contribution of the current point instead of getting a blended value. The Metaball Space operator can be used to transform the points into the local space of the metaball.

Any value that you wish to modify inside the point loop must be provided as an input. The outputs of the Point Loop operator will contain the modified versions of the inputs once the loop exits. The actual values wired into the Point Loop inputs are never modified, and so can be connected to other operators in the network; but remember that the values from these operators will always be the values unmodified by the Point Loop operator.

Here is a pseudocode example of how to compute the point color of the metaball which contributes the maximum weight to the point in space:

```
float  crt_distance = 0;
float  max_distance = 0;
vector max_color;

for_all_points( P )
{
    crt_distance = metaball density of P;
    if (crt_distance > max_distance)
    {
        max_color = metaball attribute "Cd" of P;
        max_distance = crt_distance;
    }
}
vector blended_color = max_distance * max_color;
```

9.2 INPUTS

All connected inputs mimic the properties of the output wired into them. They adopt the same help label and name as the output connected to them. If multiple outputs with the same name are connected, the names of the inputs are automatically incremented to make them unique.

POSITION

Vector representing the position of a point in space. If no no input is connected, the default is the global P.

DISTANCE

If specified, all metaballs particles within the distance of the point specified will be iterated through.

NEXT INPUT

Any number of inputs can be connected here. Each time an input is connected, a new input slot is added.

9.3 OUTPUTS

The list of outputs depends on the inputs connected to the Subnet Output operator contained in this operator. The data type and name of each output will match the corresponding input of the Subnet Output operator.

9.4 SEE ALSO

- Global Variables
- Subnet
- Subnet Input
- Subnet Output
- While
- Metaball Attribute
- Metaball Density
- Metaball Space

10 GLOBAL VOP

This operator provides outputs that represent all the global variables for the current VOP network type. There are no inputs to this operator.

The variables available from this operator will almost always include all of the input variables provided in the Output Variables operator. The difference between these two operators is that the Output Variables operator only provides inputs for those global variables that can be changed.

Every VOP network type has a set of global variables associated with it. These are the variables inherent to the VEX context that the VOP network operates in. For shading contexts (such as Surface and Displacement) these will be variables like the position of the point on the geometry being shaded (P), or the normal of the surface at the point being shaded (N). For Sop or Pop Operator networks, these variables include the position of the point being manipulated (P), and the point number of the point being manipulated (ptnum).

10.1 PARAMETERS

OUTPUT A SINGLE VARIABLE

When enabled, it shrinks the number of outputs to one and outputs only the global variable chosen from the menu below. This option is very useful when the operator is part of a subnetwork that needs to be accessed by different contexts.

VARIABLE NAME

Menu of all available global variables.
It works in conjunction with the toggle above it.

10.2 INPUTS

None.

10.3 OUTPUTS

The list of available outputs depends on the current VOP network type. The number of outputs depends on whether "Output A Single Variable" is enabled.

10.4 SEE ALSO

- Constant
- Is Connected
- Parameter
- Output Variables

11 IF VOP

11.1 DESCRIPTION

This operator contains other VOP operators. The code for the contained operators is executed only if the value fed into the first input matches the criterion specified by the Condition parameter. At least one input must be connected to this operator.

Use this operator when you want something to happen only if a certain condition is met. The Condition Value is usually the output of a Compare operator, but any integer value can be used.

Any value that you wish to modify inside the If operator must be provided as an input. The outputs of the If operator will contain the modified versions of the inputs (if the condition was met) or the unmodified version of the inputs (if the condition was not met). The actual values wired into the If operator inputs are never modified, and so can be connected to other operators in the network, but remember that the values from these operators will always be the values unmodified by the If operator.

There is no place in this operator to specify what should happen if the condition is not met (i.e. an Else block). To achieve the same effect as an if-then-else statement, use two If operators. Connect the same output value to both Condition Value inputs of the two If operators. Then set the Condition parameters of the two If operators to be different. At this point one If operator represents the "if" block, and the other represents the "else" block. Then connect your inputs to the "if" block operator. Then connect all but the Condition Value output from the "if" block into the inputs of the "else" block. Now the outputs of the "else" block are going to be what you would expect from an if-then-else statement. For simple if-then-else constructs, use the Two Way Switch operator instead.

11.2 PARAMETERS

CONDITION

Specifies the condition that must be met by the first input for the contained code to be executed. Either the input value must be False (equal to 0) or True (not equal to zero).

11.3 INPUTS

CONDITION VALUE

This integer input must be connected. The value connected to this input is compared to the requirements of the Condition parameter. Usually this input will be connected to the output of a Compare operator, or one of the logical operators (And and Or).

NEXT INPUT

Any number of inputs can be connected here. Each time an input is connected, a new input slot is added. All connected inputs mimic the properties of the output wired into them. They adopt the same help label and name as the output connected to them. If multiple outputs with the same name are connected, the names of the inputs are automatically incremented to make them unique.

11.4 OUTPUTS

The list of outputs depends on the inputs connected to the Subnet Output operator contained in this operator. The data type and name of each output will match the corresponding input of the Subnet Output operator.

11.5 SEE ALSO

- And
- Compare
- Or
- Subnet
- Subnet Input
- Subnet Output
- Switch
- Two Way Switch
- While

12 ILLUMINANCE VOP

12.1 DESCRIPTION

This operator is only available in Surface VOP networks. It contains other VOP operators, and executes the code for the contained operators once for every light that affects the surface being shaded.

This operator should only be used by advanced users with a good understanding of shading and rendering. For most purposes, the Lighting Model operator will provide all the functionality necessary to generate convincing and realistic materials. Within the illuminance loop, the Global Variables operator will change slightly. Two additional outputs will be available: the Color of the current light (Cl) and the direction from the point on the surface being shaded to the light (L). As is always the case within an illuminance loop, the values of these global variables are reset for each iteration through the code.

Any value that you wish to modify inside the illuminance loop must be provided as an input. The outputs of the Illuminance Loop operator will contain the modified versions of the inputs once the loop exits. The actual values wired into the Illuminance Loop inputs are never modified, and so can be connected to other operators in the network; but remember that the values from these operators will always be the values unmodified by the Illuminance Loop operator.

12.2 INPUTS

All connected inputs mimic the properties of the output wired into them. They adopt the same help label and name as the output connected to them. If multiple outputs with the same name are connected, the names of the inputs are automatically incremented to make them unique.

SURFACE POSITION

The position of the point on the surface being shaded. If no input is connected, the default is the global P.

SURFACE NORMAL

The normal of the point on the surface being shaded. If no input is connected, the default is the global N.

PERMITTED LIGHTING ANGLE

The range of angle (in radians) away from the Surface Normal from which lights can influence the surface. Any light outside the cone defined by this value and the Surface normal is not part of the illuminance loop. If no input is connected, the default is PI/2 (i.e. 90 degrees in radians).

LIGHT TYPE MASK

Specifies the types of lights that are allowed to influence the surface. This is a bit mask, where:

```
LIGHT_AMBIENT  = 0x01
LIGHT_DIFFUSE  = 0x02
LIGHT_SPECULAR = 0x04
```

If no input is connected, the default is LIGHT_DIFFUSE | LIGHT_SPECULAR.

LIGHT MASK

Specifies a string listing names of light objects that can influence the surface. If no input is specified, the light mask specified in the parameters for the object being rendered is used.

NEXT INPUT

Any number of inputs can be connected here. Each time an input is connected, a new input slot is added.

12.3 OUTPUTS

The list of outputs depends on the inputs connected to the Subnet Output operator contained in this operator. The data type and name of each output will match the corresponding input of the Subnet Output operator.

12.4 SEE ALSO

- Global Variables
- Shadow
- Subnet
- Subnet Input
- Subnet Output
- While
- Lighting Model

13 MAX VOP

13.1 DESCRIPTION

This operator outputs the maximum value from its inputs.
All inputs and the output are float values.

13.2 INPUTS

Any number of inputs can be specified.

INPUT NUMBER 1...N

The input values to be compared.

NEXT INPUT

Where the next input value should be connected.

13.3 OUTPUTS

COMBINED VALUE

The maximum of all the input values.

13.4 SEE ALSO

- Average
- Ceiling
- Clamp
- Max Vector Component
- Minimum

14 MIN VOP

14.1 DESCRIPTION

This operator outputs the minimum value from its inputs. All inputs and the output are float values.

14.2 INPUTS

Any number of inputs can be specified.

INPUT NUMBER 1...N

The input values to be compared.

NEXT INPUT

Where the next input value should be connected.

14.3 OUTPUTS

COMBINED VALUE

The minimum of all the input values.

14.4 SEE ALSO

- Average
- Clamp
- Floor
- Maximum
- Min Vector Component

15 MULTIPLY VOP

15.1 DESCRIPTION

This operator outputs the product of its inputs.

The first input can be an integer, float, vector, vector4, matrix3, or matrix. The allowed data types of subsequent inputs depend on the data type of the first input. For example, if the first input is a float, subsequent inputs can be either floats or integers. The output data type is always the same as the data type for the first input.

15.2 INPUTS

Any number of inputs can be specified.

INPUT NUMBER 1...N

The input values to be multiplied together.

NEXT INPUT

Where the next input value should be connected.

15.3 OUTPUTS

COMBINED VALUE

The product of all the input values.

15.4 SEE ALSO

- Add
- Divide
- Multiply Constant
- Subtract

16 OR VOP

16.1 DESCRIPTION

This operator performs a logical "or" operation between its inputs and returns 1 (if at least one input is non-zero) or 0 (if all inputs are zero).

Typically, "or" is used as an input to conditional operators such as If and While. All inputs and the output data type are integers.

16.2 INPUTS

Any number of inputs can be specified.

INPUT NUMBER 1...N

The input values to be combined together.

NEXT INPUT

Where the next input value should be connected.

16.3 OUTPUTS

COMBINED VALUE

The logical "or" combination of all inputs.

16.4 SEE ALSO

- And
- Not
- Compare
- If
- While

17 OUTPUT VOP

17.1 DESCRIPTION

Every VOP network requires one of these.

This operator provides inputs that represent all the global variables for the current VOP network type that can be modified. There are no parameters and no outputs to this operator. Every VOP network contains exactly one of this operator type. The Output Variables operator cannot be deleted except by deleting the parent VOP network. It is also not possible to create a second Output Variables operator within a VOP network (even within a subnet).

This operator drives the VEX code generation process for a VOP network. When generating code, only operators that are directly or indirectly connected to an input of the Output Variables operator will actually generate any code. This operator is also always the last operator to output its code. It simply goes through all of its inputs, and sets the global variable corresponding to each connected input to the value that is wired into it.

The variables available from this operator will be a subset of the variables available as outputs from the Global Variables operator. The difference between these two operators is that the Output Variables operator only provides inputs for those global variables that can be changed.

Every VOP network type has a set of global variables associated with it. These are the variables inherent to the VEX context that the VOP network operates in. For shading contexts (such as Surface and Displacement) these will be variables like the position of the point on the geometry being shaded (P), or the normal of the surface at the point being shaded (N). For Sop or Pop Operator networks, these variables include the position of the point being manipulated (P), and the point number of the point being manipulated (ptnum).

17.2 INPUTS

The list of available inputs depends on the current VOP network type.

17.3 SEE ALSO

- Global Variables
- Parameter

18 PARAMETER VOP

18.1 DESCRIPTION

This operator creates a parameter to appear in the signature of the VEX function defined by the VOP network (VOPNET) and in the interface of any OP type that uses the VOPNET.

By providing VEX parameters, it becomes possible to customize the operation of the VOPNET for each instance of the operator using the VOPNET. For example, if a Surface VOPNET is used in a surface shader for a variety of colored baloons, the balloon color should be defined by a Parameter VOP. This way there may be 100 individually colored balloons, with a single VOP network defining the surface shader (i.e. the SHOP) for all of them.

18.2 PARAMETERS

PARAMETER TYPE

Specifies the VEX data type of the new parameter. It can also specify how the parameter should be represented in an OP dialog. For example, a VEX vector can be viewed as 3 float values or as a color value.

PARAMETER NAME

The name of the new parameter, both in the VEX function declaration, the definition of any OP type that uses the VOPNET, and the VOP tile's output. If a parameter by this name already exists, this operator will reference it and will disable most of of its own fields, such as the Parameter Type and Parameter Label.

PARAMETER LABEL

The label that appears in the dialog of the OP type that uses the VOPNET, and in the baloon help over the VOP tile output.

USE INPUT VALUE IF PARAMETER NOT BOUND

If true, the VEX function parameter is checked to see if it is bound to an attribute in the current VEX context (e.g. point color). If not bound, the input value is is assigned to the VEX parameter. In a Surface VOP context, a VEX parameter is bound if the geometry being shaded has an attribute with the VEX parameter name.

EXPORT PARAMETER

Specifies whether the new parameter can be exported to other contexts (written to as well as read from). If true, this operator gets an input. The value wired into this input is then assigned to the exported parameter. In a Surface network, exported parameters can be used to create deep rasters. In SOP and POP networks, the exported parameters create new geometry attributes.

PROVIDE MENU OF CHOICES

If the parameter type is String or Integer, check this toggle to provide a menu of choices instead of a string entry field or an integer slider.

MENU CHOICES

A list of pairs of strings. The first string in each pair is the parameter value if that menu item is chosen. The second string is the label that appears in the dialog for that menu item.

DEFAULT VALUES

Depending on the parameter type selected, it represents the default value of the parameter in the VEX function and in the OP type that uses this VOP network.

18.3 INPUTS

Unbound or Export Value - The value assigned to the new parameter if the "Export Parameter" toggle is checked or if the parameter is not bound when "Use Input Value If Parameter Not Bound" is checked.

18.4 OUTPUTS

OUTPUT PARAMETER

The value of this VEX function parameter. Is Parameter Bound - True is the new parameter is bound to an attribute.

SEE ALSO

- Constant
- Global Variables
- Output Variables
- Shading Layer Parameter

19 PRINT VOP

This operator is used to generate a formatted text string. It can also output this text to the console, usually for debugging purposes.

Any number of inputs can be connected to this operator. These inputs can be integer, float, vector, vector4, matrix3, matrix, or string data. By default, each input value will be printed with the name of the output connected to the input, followed by the actual value. the values are separated by tabs. So connecting a vector input "P" and a float input "alpha" would result in string that looked like this:

```
P: { 0.0, 0.0, 0.0 }     alpha: 1.0
```

This operator can be placed anywhere in the VOP network and does not have to be part of the node chain that ultimately connects to the "Output Variables" operator.

19.1 PARAMETERS

FORMAT STRING

This string specifies how the input values will be formatted and displayed. Any text can be entered here. To specify that the value from a particular input should be displayed, use "%". The inputs will be displayed in the order in which they are connected. To display a "%" character, use "\%". To display a "\", use "\\". To specify the start of a new line, use "\n". A new line character is automatically put at the end of the string when it is output to the console. To place a TAB in the text, use "\t". The double-quotes are not actually needed anywhere in the string. Any input value that does not explicitly appear in the Format String is appended to the end of the string using the format described above for the default behaviour.

OUTPUT TEXT TO CONSOLE

Toggle this parameter on to have the formatted output string displayed in the console window.

INPUTS

Any number of inputs can be specified.

INPUT NUMBER 1...N

The input values to be put in the output string.

NEXT INPUT

Where the next input value should be connected.

19.2 OUTPUTS

FORMATTED OUTPUT TEXT

The string created by formatting the input values according to the details in the Format String.

19.3 EXAMPLE

To generate a more easily readable string than the default, you might set the Format String to:

```
The noise value is % with displacement %.
The color is %.
```

Or to generate the name of a texture map file based on an integer input, you could set the Format String to:

```
$HIP/map/Texture%.rat
```

20 SHADING LAYER VOP

20.1 DESCRIPTION

This operator creates a parameter to appear in the signature of the VEX function defined by the VOP network (VOPNET). The new parameter is named according to the chosen geometry attribute and the specified layer.

Unlike the Parameter operator, which creates parameters of any name and type, Shading Layer computes the parameter name from the name of the geometry attribute and the specified layer. For example, choosing "UV Coordinates" from the attribute menu will generate "uv, "uv2", "uv3" when the layer is 1 and respectively 2 and 3. The parameter type matches the type of the selected geometry attribute.

This operator always checks if the geometry attribute is bound and, if so, assigns its value to the new parameter. Thus, the new parameter will be hidden in the operator type defined by the VOP network.

One of the most common uses of this operator is the creation of layered textures, which requires different sets of texture coordinates for each Texture operator. To accomplish this task, choose "UV Coordinates" from this operator's attribute menu and set the appropriate layer. Pipe the "uv" output into a Vector To Float operator to extract the "s" and "t" as individual float values, then wire "s" and "t" into the Texture operator. Repeat this operation for all the layers, then add or multiply the all Texture outputs using the Add or Multiply operator respectively.

20.2 PARAMETERS

GEOMETRY ATTRIBUTE

The attribute to be searched for in the given layer. The following attributes are available in this operator:

Label	Name	Type	Default If not Bound
V Coordinates	uv	3 floats	{ global s, global t, .5 }
RGB Color	Cd	3 floats	{ 1, 1, 1} (i.e. white)
Alpha	Alpha	1 float	(i.e. opaque)

Geometry attributes are created in the Geometry module using a variety of SOP operators such as: AttribCreate, Attribute, Point, UV Project, UV Texture, UV Unwrap, etc.

ATTRIBUTE LAYER

Attribute layer where to search for the geo attribute. Layer 1 is the default layer. Attributes in this layer do not have a suffix. Layers are created in the Geometry module using the Layer SOP.

EXPORT PARAMETER

Specifies whether the new parameter can be exported to other contexts (written to as well as read from). If true, this operator gets an input. The value wired into this input is then assigned to the exported parameter. In a Surface network, for example, exported parameters can be used to create deep rasters.

20.3 INPUTS

EXPORT VALUE

The value assigned to the new parameter if the "Export Parameter" toggle is checked.

20.4 OUTPUTS

OUTPUT PARAMETER

The value of this VEX function parameter.

IS PARAMETER BOUND

True is the new parameter is bound to an attribute.

20.5 SEE ALSO

- Constant
- Global Variables
- Parameter
- UV Space Change

21 SPLINE VOP

21.1 DESCRIPTION

This operator computes either a Catmull-Rom (Cardinal) spline or a Linear spline between the specified key points, given an interpolant (u) in the domain of the spline. The result is a 1D, 3D, or 4D value representing u's image on the spline.

The Cardinal keys are uniformly spaced over the range 0 to 1. Because of the nature of the Cardinal spline, the value associated with the first and last keys will never be returned. However, these keys are used to determine the shape of the curve on entry and exit. If you want the spline to reach its end points, connect the first key and last key twice. See the Fire operator for one such example.

If you are blending only two vector types using the Cardinal interpolant, the Color Mix operator offers the convenience of fewer connections. Make sure, however, that you choose the "Smooth With Cardinal Spline" option to obtain the exact same results as with this operator.

The Linear spline interpolation is equivalent to a sequence of Linear Interpolation operators. Consider the simpler Mix operator if only two inputs are required.

21.2 PARAMETERS

SPLINE

Catmull-Rom (Cardinal) or Linear.

21.3 INPUTS

Any number of inputs can be specified.

PARAMETRIC COORDINATE

Interpolant (u) representing the parametric location in the domain of the spline where to evaluate the spline.

INPUT NUMBER 1...N

1D, 3D or 4D spline key points to interpolate.

NEXT INPUT

Where the next key value should be connected.

21.4 OUTPUTS

COMBINED VALUE

Interpolated spline location.

21.5 SEE ALSO

- Blend Regions
- Color Mix
- Fire, Mix
- Shift
- Smooth

22 SUBINPUT VOP

22.1 DESCRIPTION

This operator allows the connection of operators outside a subnet to operators inside the subnet. This operator can only be created inside a subnet type operator (If, Illuminance, Subnet, and While).

For every input connected to a subnet type operator, this operator will have an output that corresponds to that output. Connecting the output of this operator to another operator inside the subnet is the same as connecting the operator outside the subnet to the operator inside the subnet. Disconnecting an input from the subnet will delete the corresponding output from this operator. Thus care must be taken when removing inputs from the subnet to check that the connections from this operator are still correct.

22.2 OUTPUTS

The list of outputs depends on the inputs connected to the subnet containing this operator. The data type of each output will match the data type of the corresponding input. The name will be the same, except it will include the prefix "sub".

22.3 SEE ALSO

- If
- Illuminance
- Subnet
- Subnet Output
- While

23 SUBNET VOP

23.1 DESCRIPTION

This operator contains other VOP operators. It generates no code of its own, and simply acts as a container for a block of functionality.

Use this operator to help organize complex VOP networks. If a network becomes very large and difficult to navigate, it often helps to collapse self-contained blocks of functionality into subnets. When doing this, try to isolate and collapse groups of operators that do not involve large numbers of external connection.

23.2 INPUTS

NEXT INPUT

Any number of inputs can be connected here. Each time an input is connected, a new input slot is added.

All connected inputs mimic the properties of the output wired into them. They adopt the same help label and name as the output connected to them. If multiple outputs with the same name are connected, the names of the inputs are automatically incremented to make them unique.

23.3 OUTPUTS

The list of outputs depends on the inputs connected to the Subnet Output operator contained in this operator. The data type and name of each output will match the corresponding input of the Subnet Output operator.

23.4 SEE ALSO

- If
- Subnet Input
- Subnet Output
- While

24 SUBOUTPUT VOP

24.1 DESCRIPTION

This operator allows the connection of operators inside a subnet to operators outside the subnet. This operator can only be created inside a subnet type operator (If, Illuminance, Subnet, and While, for example).

Like the Output Variables operator, one and only one of these operators can exist inside each subnet. This operator cannot be deleted without deleting the containing subnet. It is also not possible to add a second Subnet Output operator. Also like the Output Variables operator, this operator drives the VEX code generation within a subnet. Only operators connected directly or indirectly to it will generate any code within the subnet block. For While operators, it is important to always have some value connected to the first input of this operator (which will correspond to the Condition Value of the While operator). If no value is connected to this input, the Condition Value will never change, and the While loop will continue forever.

For every operator connected to an input of this operator, the containing subnet operator will have an output. Connecting an operator to this subnet output is the same as connecting the operator to the corresponding input of this operator.

24.2 PARAMETERS

INPUT NAME/LABEL

For each input connected to this operator, excluding those inputs that correspond to inputs of the containing subnet, you can specify the name and label for the input. The name and label are also used by the outputs of the containing subnet. To change the name or label for an input, enter the new value directly in the table presented. If an epty value is specified in the table, the name and label will be copied from the operator connected to each input.

24.3 INPUTS

NEXT INPUT

Any number of inputs can be connected here. Each time an input is connected, a new input slot is added. This also results in a new output being created on the containing subnet operator.

For every input connected to the subnet containing this operator, this operator will have an input. This input will be the same name as the corresponding subnet input, with the prefix "sub" added to it. Connections made to the Next Input input of this operator will always appear after the connections corresponding to the containing subnet inputs.

SEE ALSO

- If
- Illuminance
- Output Variables
- Subnet
- Subnet Input
- While

25 SUBTRACT VOP

25.1 DESCRIPTION

This operator outputs the result of subtracting all its inputs.

The first input can be an integer, float, vector, vector4, matrix3, or matrix. The allowed data types of subsequent inputs depend on the data type of the first input. For example, if the first input is a float, subsequent inputs can be either floats or integers. The output data type is always the same as the data type for the first input.

25.2 INPUTS

Any number of inputs can be specified.

INPUT NUMBER 1...N

The input values to be subtracted.

NEXT INPUT

Where the next input value should be connected.

25.3 OUTPUTS

COMBINED VALUE

The result of subtracting all the input values.

SEE ALSO

- Add
- Add Constant
- Complement
- Divide
- Multiply

26 SWITCH VOP

26.1 DESCRIPTION

This operator outputs the value connected to one of its inputs. The first input is always an integer that specifies the index of the value to output. The other inputs can be of any type, but they must all be the same type.

26.2 INPUTS

SWITCHER INDEX

The value connected to this input is used as an index to determine which input value to feed into the output value. For example, a value of 0 will choose input1; a value of 1 will choose input2, and so on.

Usually, the switcher index will be connected from the output of a Compare operator, or one of the logical operators (And and Or).

INPUT NUMBER N

Any number of inputs can be connected here. Each time an input is connected, a new input slot is added.

26.3 OUTPUTS

CHOSEN VALUE

This output will be of the same type as all the inputs.

SEE ALSO

- And, Compare, If, Or, Two Way Switch

27 WHILE VOP

27.1 DESCRIPTION

This operator contains other VOP operators. The code for the contained operators is executed repeatedly in a loop until the input condition is no longer satisfied. At least one input must be connected to this operator.

Use this operator to repeat the same sequence of code many times. Be careful when using this operator to ensure that the condition will eventually fail. Otherwise an endless loop can result. Such endless loops can be interrupted within Houdini, but can result in renders that never finish.

If you know that the contents of the While operator would be executed at least once, it is easiest to use a Constant operator with an integer value set to 1 as the Condition Value input. Then within the While operator, set up the real comparisons and conditions that will lead to exiting the loop. If the While operator code may need to be skipped over entirely, there may be some duplication of operators inside and outside the While operator.

Any value that you wish to modify inside the While operator must be provided as an input. The outputs of the While operator will contain the modified versions of the inputs once the loop exits. The actual values wired into the While operator inputs are never modified, and so can be connected to other operators in the network, but remember that the values from these operators will always be the values unmodified by the While operator.

For those familiar with VEX programming, it may be noted that there is no "For" operator. This is because a While operator can do anything that a For operator would be able to do, and provides a simpler framework in which to implement the loop.

27.2 PARAMETERS

CONDITION

Specifies the condition that must be met by the first input for the contained code to be executed. Either the input value must be False (equal to 0) or True (not equal to zero).

INPUTS

All connected inputs mimic the properties of the output wired into them. They adopt the same help label and name as the output connected to them. If multiple outputs with the same name are connected, the names of the inputs are automatically incremented to make them unique.

CONDITION VALUE

This integer input must be connected. The value connected to this input is compared to the requirements of the Condition parameter. Usually this input will be connected to the output of a Compare operator, or one of the logical operators (And and Or).

NEXT INPUT

Any number of inputs can be connected here. Each time an input is connected, a new input slot is added.

27.3 OUTPUTS

The list of outputs depends on the inputs connected to the Subnet Output operator contained in this operator. The data type and name of each output will match the corresponding input of the Subnet Output operator.

SEE ALSO

- And
- Compare
- If
- Or
- Subnet
- Subnet Input
- Subnet Output