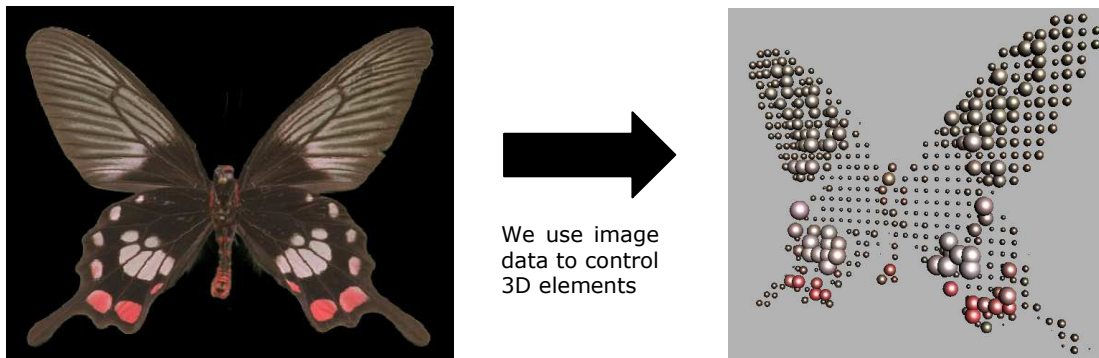


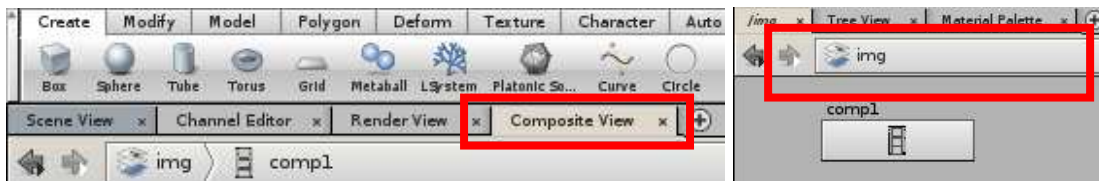
Workshop 04: Image-driven Animation



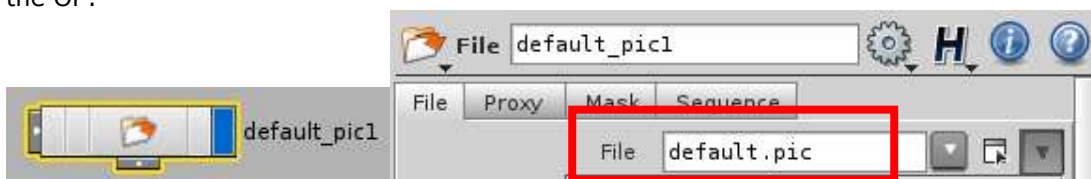
In this week we will look at several examples on how to create image-driven animations.

Image-driven animation

Step 1) First you need an image network. Go to the **/img** category. By default, Houdini creates an empty Image Network called "comp1" for you. You can use it, or you can create your own one. Go into the Image Network by double-clicking it. We are going to construct an Image Network (called a COP2 network) to manipulate the image. You also need to choose a **Composite View** to view the 2D image processing result (instead of using a Scene View):



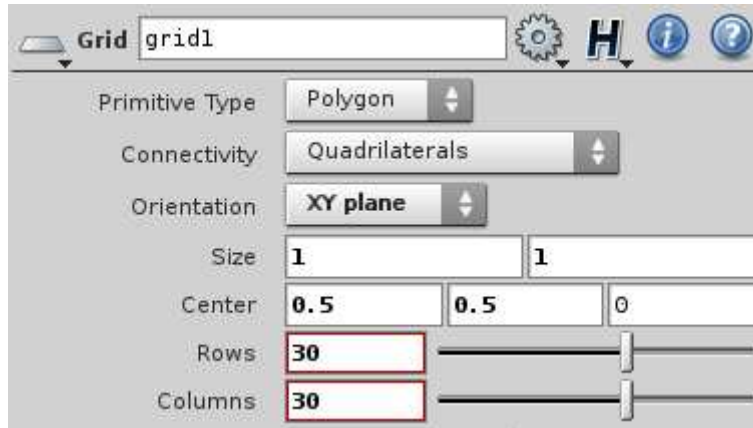
Press TAB and create a **FILE OP**. You can choose your own image, or you can use the default "butterfly" image (it is called *default.pic*). Don't forget turning on the blue-flag of the OP:



Note that the name of the FILE OP will be changed based on what your image file name is. Suppose you use the default image, then the name of this OP is called **/img/comp1/default_pic**. You will need this name in the later steps.

You can further process the image using other OPs in this network, say, **Invert** or **Bright** or others. However, let's use this simple image first.

Step 2) Now, go back to **/obj**. Create a **GEOMETRY** and go into it. I start from a **GRID** (I will explain why I set the grid's parameters in such a way soon):

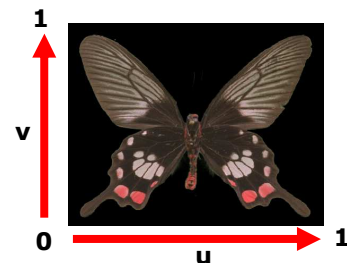


Step 3) (Slower version) whenever you want to grab the pixel data to control a parameter, you can give this expression to the parameter:

pic(a, u, v, d)

where

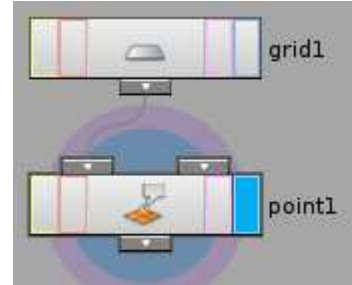
- a is the name of OP in the Image Network;
- u and v specify the location of the pixel on the 2D image (in the range [0-1]);
- d is one of the following: D_CR, D_CG, D_CB, D_CA, D_CHUE, D_CSAT, D_CVAL, D_CLUM for the red, green, blue, alpha, hue, saturation, value, or luminance of the pixel.



For example, let's say you want to use the luminance of a pixel to modify the grid point's z-position. To modify a point's attribute, we can use a **POINT OP**. In Houdini, POINT is a useful OP: it can be used to modify points' position, color, alpha, normal, etc. Let's try this: in the z-position of the POINT OP, type the following:

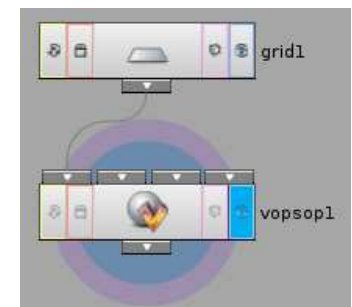
pic("/img/comp1/default_pic", \$TX, \$TY, D_CLUM)

The \$TX and \$TY are two internal variables available in POINT: \$TX is the x-position of the point, and \$TY is the y-position of the point, both of them are within [0,1] because of our GRID's setting in step 2. Therefore, they are suitable for the pic() function.

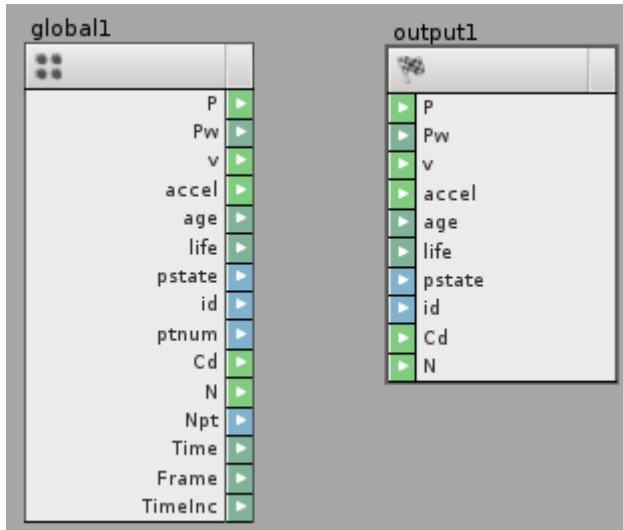


Step 3) (Faster version) plug the GRID into the first input of a **VOP SOP**. The VOP SOP will take the points coming down from above, and *execute a script* to modify the points' attributes. But don't worry: we don't need to write the script line-by-line. We can construct the script by using the "connecting OPs together" workflow.

Double-click the VOP SOP to go into it. If you press TAB you can see a different set of OPs available, which are used to construct the script. In this workshop I don't want to go deep into it. I will just cover a few OPs which can help us to finish the setup.

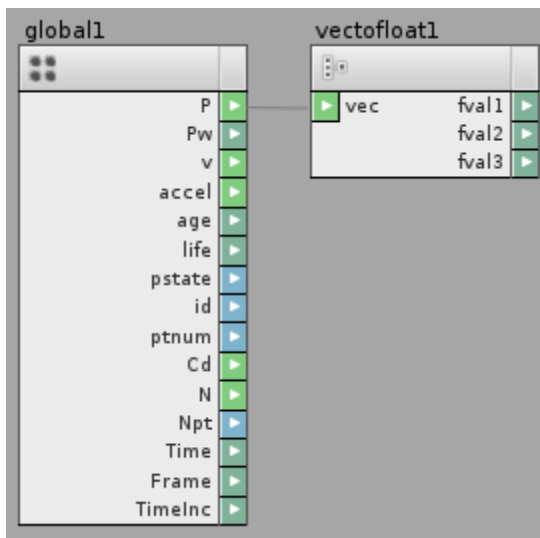


All the OPs in VOP SOP have their input on the left side, and output on the right side. By default, there are two OPs created already: the *global1* gives you the point's attributes, and the *output1* let you assign the point's attributes:



Let's try a simple example: directly connect the "P" (position) of *global1* to the "Cd" (color) of *output1*. This means "using the point's position as its color: x-position value is assigned to the red color, y-position value is assigned to the green color, and z-position value is assigned to the blue color". Try it and see what happen.

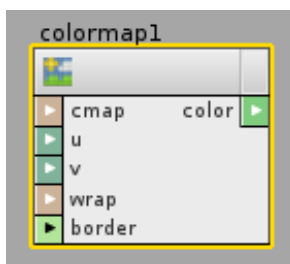
Now we are going to include our "butterfly" image into this network. First, let's separate the position data into 3 channels (x, y and z), using a **Vector To Float**:



The *fval1* is the point's x position; *fval2* is the y position, and *fval3* is the z position.

And we create a **Color MAP** to read an image. We can give a file name to it, but we can also refer to an OP inside a COP2 network by using the syntax:

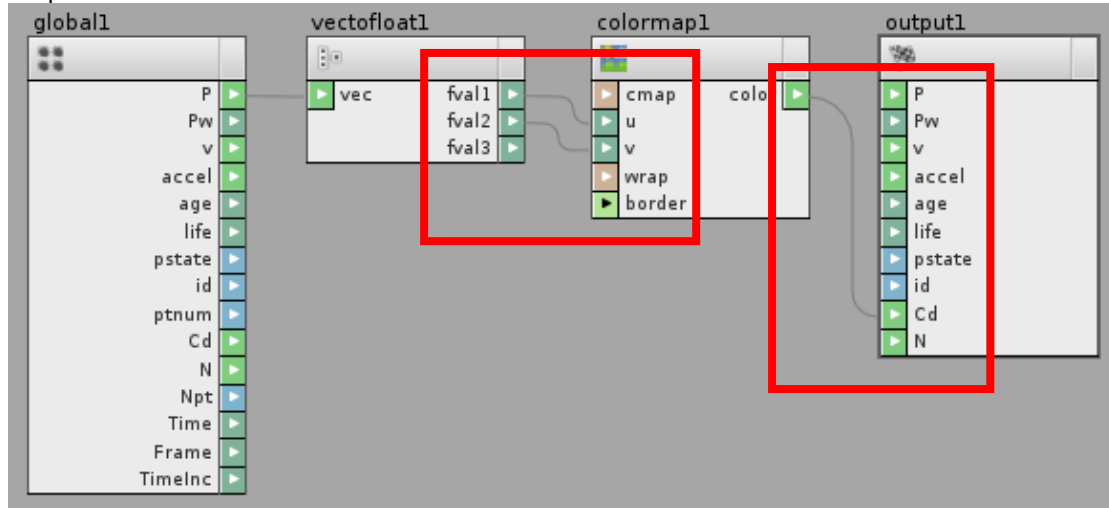
op:/img/comp1/default_pic



Referring to an OP inside a COP2 network has an advantage: we can do other 2D image processing on the images before we use it by this Color Map, or we can also load an image sequence into a COP2 network and use it.

Then we can use the x-position of the points as the **u** of the Color Map (the horizontal position on the image), and y-position of the points as the **v** of the Color Map (the vertical position on the image), like what we did in the pic() expression.

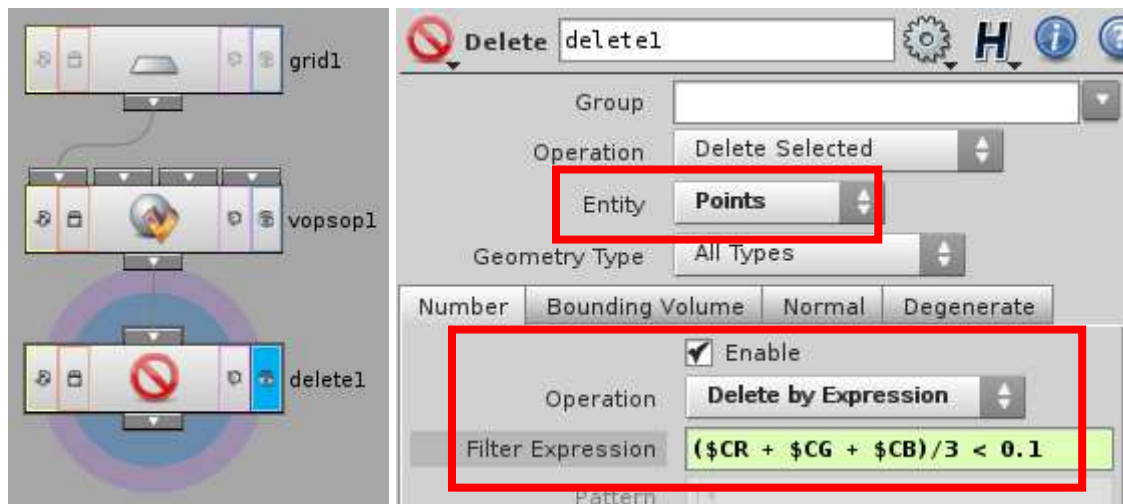
Lastly, we plug the output of the Color Map (i.e. the image pixel's color) to the Cd of output1:



One limitation in this setup is that we are assigning the image pixel color directly to the point color. Please read the following remark on how to assign the image pixel color to an arbitrary attribute.

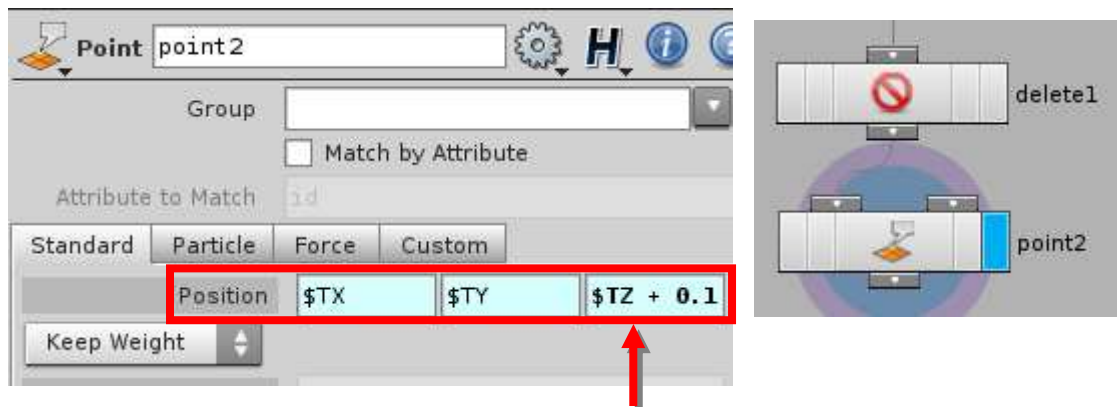
Step 4) Now we can go back to the geometry level. We have assigned color to the points, based on an image's pixel color. Of course we don't want to stop at such a "texture mapping" effect. We can go further, using whatever OPs provided by Houdini.

Let me give you some examples: suppose you want to delete all points with dark color. You may use a **DELETE**. To use the DELETE OP, you have to specify which points you want to delete. Say, we can use an expression to specify that "delete all points with dark luminance", or, put it in a more technical term, " $(\$CR + \$CG + \$CB)/3 < 0.1$ ":



Note that in this example, the **\$CR**, **\$CG** and **\$CB** are internal variables of the red, green and blue color of the points coming down (which has been assigned by the VOP SOP).

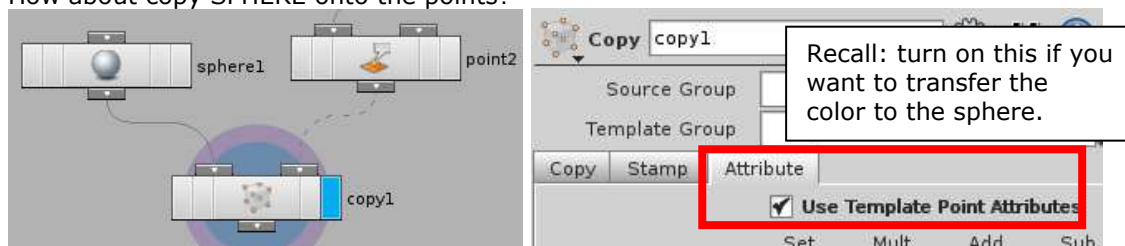
As another example, let's try to move the point's z-position based on the pixel's luminance again. We can use POINT:



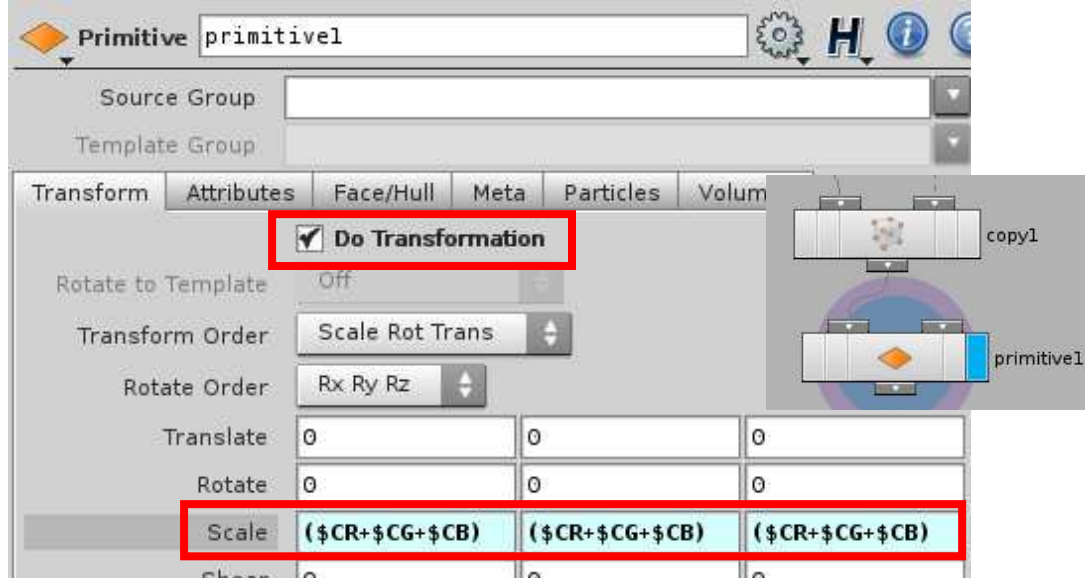
$$\$TZ + 0.1 * (\$CR + \$CG + \$CB)/3$$

The "\$TZ +" means adding extra value to the original point's z-position (\$TZ).

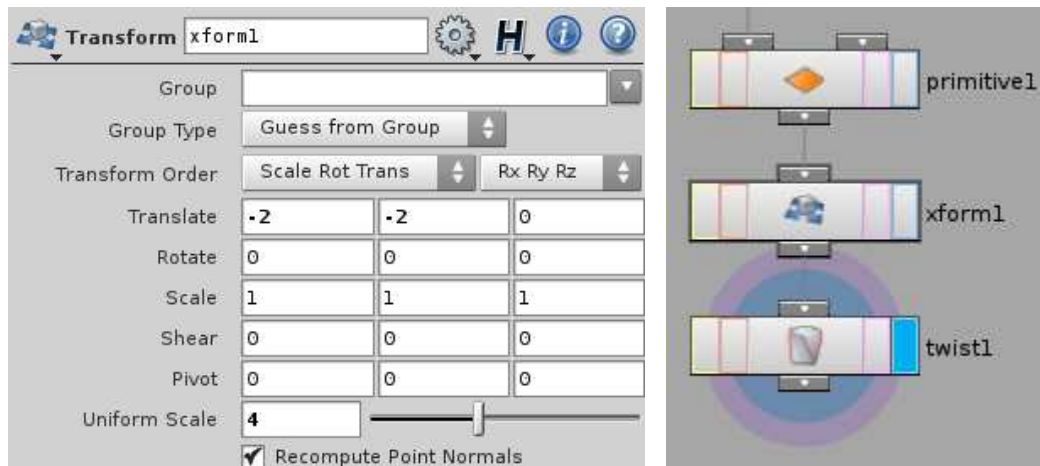
How about copy SPHERE onto the points?



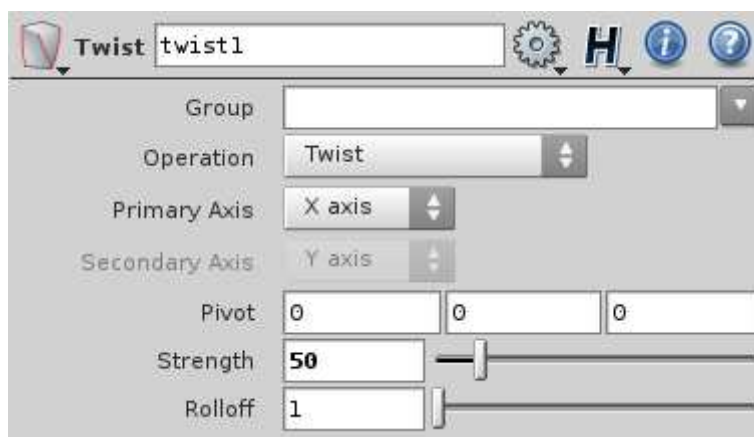
And how about scale the spheres based on the pixel's luminance? You can either use the *stamping* technique; or, as the copies of sphere are *primitives*, you can simply use a **PRIMITIVE** afterward to change their transformation attributes. A PRIMITIVE is quite similar to POINT; it can modify different attributes of the primitives coming down:



Note that we have used the point's x-position and y-position to access the (u,v) position of the image pixel color. The **u** and **v** of **Color Map** should be in the range [0,1], and that's why I setup the GRID in such a manner so that all the point's x and y position are within 0 and 1. After we have grabbed the pixel's information in VOP SOP, we can freely scale or transform the whole model. For example, you can attach a TRANSFORM to transform it;



And, how about adding TWIST to twist it?



NULL OP

I will suggest in the Image Network adding a **NULL OP** after your FILE OP, and then modify the file name from:

op:/img/comp1/default_pic

to

op:/img/comp1/null1



In our example, after adding a NULL OP you can insert any number of OP between the FILE and NULL, for doing some 2D image processing, without need to modify your Color Map in the VOP SOP. For example, you can insert an INVERT between the FILE and the NULL to invert the color of the image.

Exercise 1

Note that you can also use a video-clip. However, Houdini does not read video file directly. You have to convert it into an image sequence using other software, such as Adobe Premiere. In this exercise, you are given an image sequence for testing; they are put under the folder **image_sequence**. Note also the naming of those files.

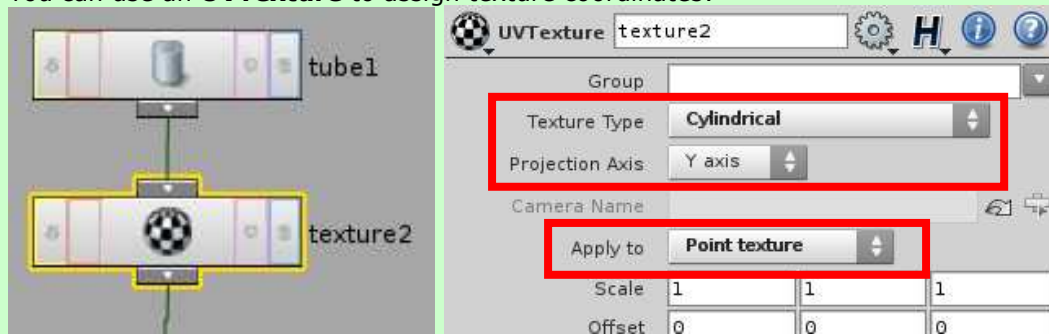
Using the image sequence, try to simulate the *Wooden Mirror* effect using Houdini. You can copy boxes onto a grid, and the rotation of boxes should be based on the pixel color from the given image sequence.

Remark beyond this class: assigning texture coordinates

Using the (x, y) position of a point to access the (u, v) of an image may not be a good idea: you can only start from a grid (with size 1x1). A more general approach is to assign *texture coordinates* to the object, and use the texture coordinates to access the (u, v) on an image.

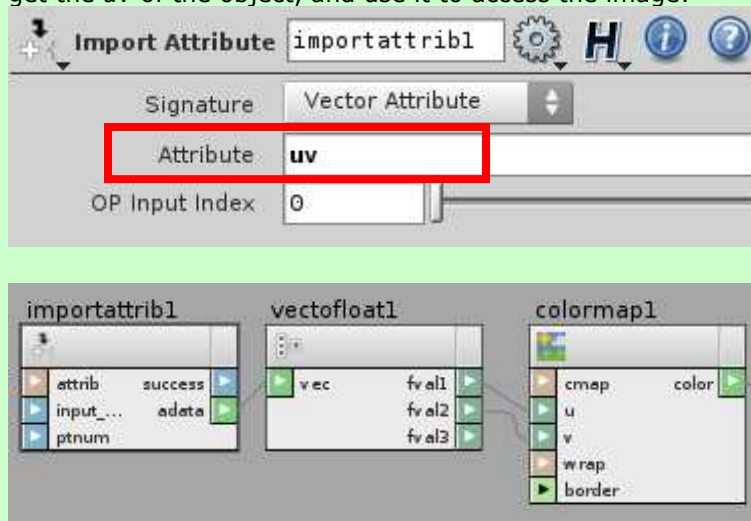
Since assigning texture coordinates to an object is not the major focus of this course, I will only show you the simplest way here.

You can use an **UVTexture** to assign texture coordinates:



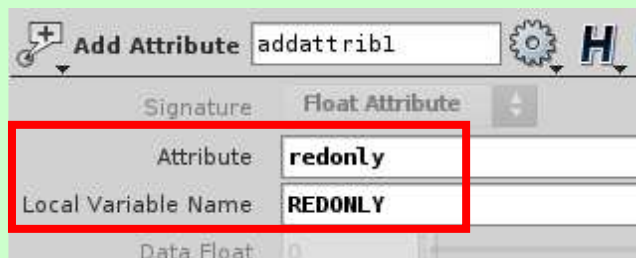
You can try different combination of **Texture Type** and **Projection Axis** for different object type. For example, *Polar* is suitable for a sphere, and *Orthographic* (with a suitable Projection Axis) is good for a grid.

Middle-click on the UVTexture, you can see that a new point attribute called *uv* has been assigned to the object. Inside the VOP SOP, you can use the Import Attribute to get the *uv* of the object, and use it to access the image:



Remark beyond this class: adding new attribute by VOP SOP

In our previous example, we assign the pixel's color to the *color* attribute of the object. In some situations you may want to assign value to other (new) attributes of the object. For example, inside VOP SOP you can extract the red color of the pixel, and assign it to a new attribute of the object (let's call it *redonly*), using the **Add Attribute**:



Go back to the geometry level, we can see that a new attribute called *redonly* is added to the object. We can use this value afterward, using the variable symbol \$REDONLY.



For example, we can use \$REDONLY here to get the *redonly* value.

Cache the intermediate results

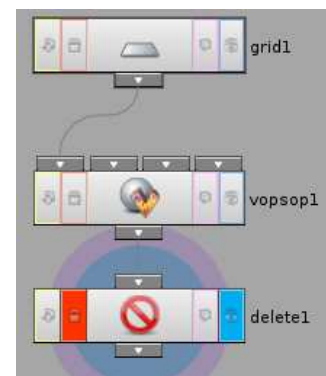
In case you still think that the VOP SOP is a little bit slow, especially when the GRID has a lot of points, you may consider caching the intermediate result (point positions and colors). This is useful especially when you seldom change the setting of GRID and VOP SOP.

There are two ways to do that: (1) to "lock" the OP; or (2) to save the results to disk as external files, and read them back later.

Let's say we want to cache the intermediate results at the DELETE OP. You can:

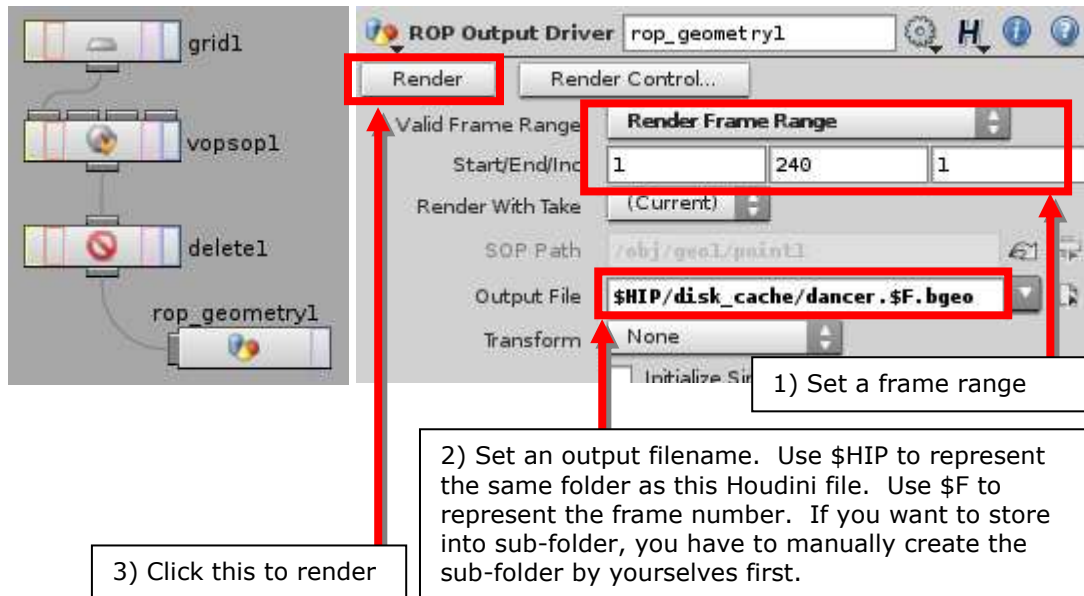
Method 1) Lock an OP

You can turn on the red flag of any OP to cache the intermediate results. The results will be saved together with the file (i.e. the file size will be larger), but the operations of GRID, VOPSOP and DELETE will not be evaluated again until you turn the red flag off – to "unlock" the OP.



Method 2) Save to external files

You can also use the **ROP OUTPUT DRIVER** to save the intermediate results to external files:



The intermediate results of DELETE (i.e. the vertices, edges, faces and point color) are then saved onto the hard-disk as a sequence of geometry file (the format is called BGeo).

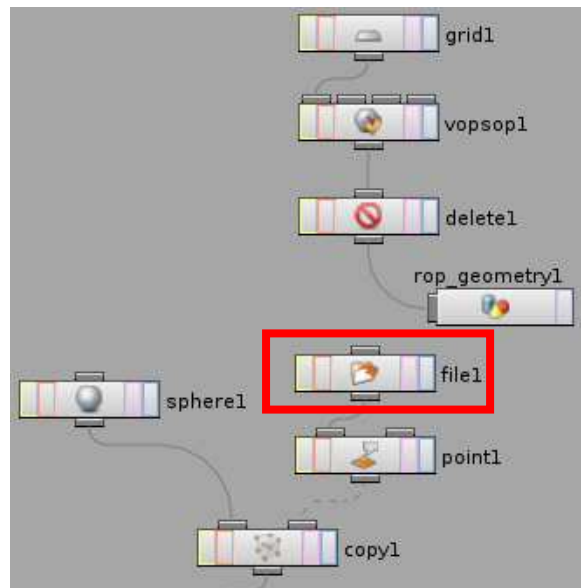
Now you can read it back using **FILE**, with the same filename

\$HIP/disk_cache/dancer.\$F.bgeo

(Note that the filename is *dancer.\$F.bgeo*, which means that at frame#1, it loads the file *dancer.1.bgeo*, and at frame#2, it loads the file *dancer.2.bgeo*, etc.)

The advantage of this is obvious: reading the *bgeo* files from the hard-disk is much faster than re-compute the OPs above the DELETE again and again.

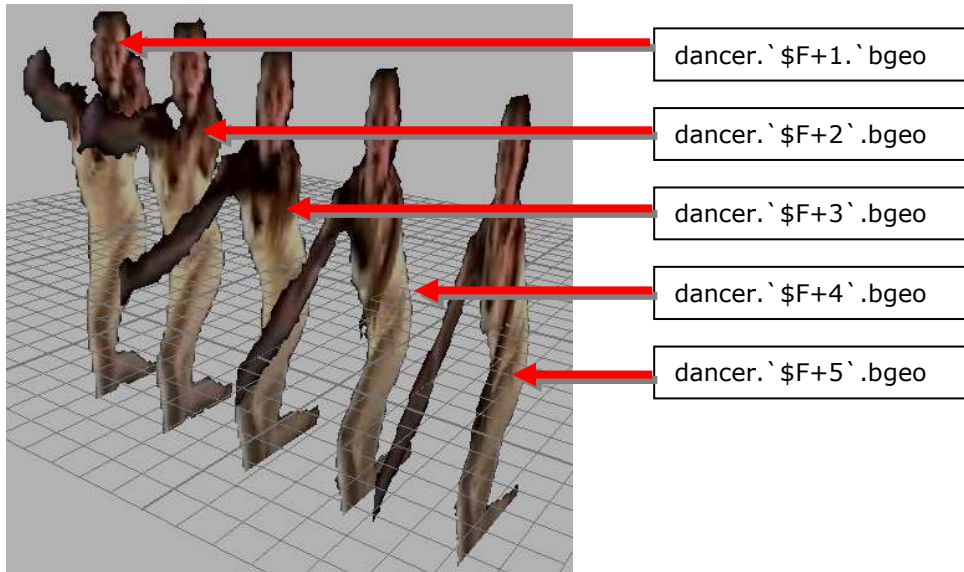
Latter on, if you have changed the GRID's setting or the VOP SOP's parameters, you can always go back to this ROP OUTPUT DRIVER, click the RENDER button, and re-generates the disk-cache again.



Time-shifted image sequence

If you have saved onto the hard-disk the intermediate results of geometry for all the frames, there is one more funny option that you can play with: the file index you load is not necessary to be equals to the current frame number. Therefore, at frame#1 it is not necessary to load the file *dancer.1.bgeo*. I can "shift" the time a little bit.

Assume that I want to load 5 grids at a time, each with different time-shifted. For example, at frame number \$F I want to have:



This can be achieved by creating 5 FILES, and putting a simple expression inside the filename parameter of the FILES:

dancer.`\$F + 1`.bgeo for file1
dancer.`\$F + 2`.bgeo for file2
dancer.`\$F + 3`.bgeo for file3
 ...

And you have learnt stamp(), so you can use the COPY and stamping technique instead of using several FILES:

Filename **dancer.`\$F + stamp("...", "fnum", 0)`.bgeo**

A grid with Row=5 and Columns=1

Create a stamp variable **fnum** with value **\$PT**

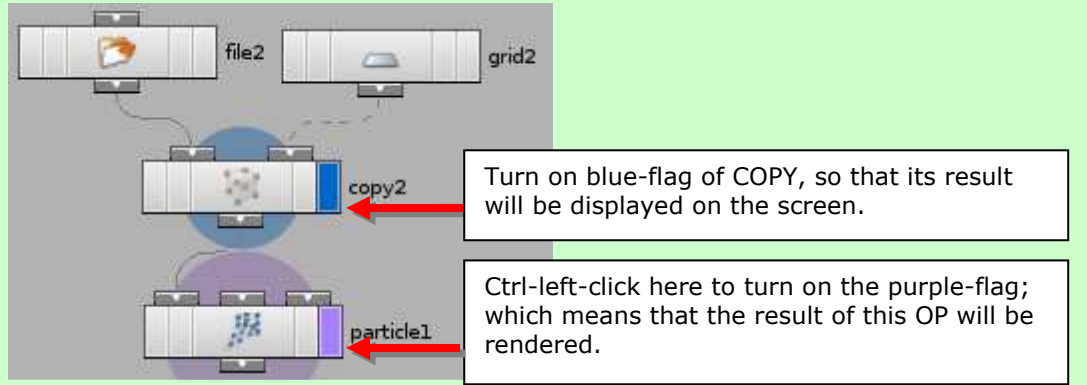
Generate particles

Lastly, don't forget that we have tried PARTICLE. You can generate particles from those grid points:

Add some *Turbulence* too!

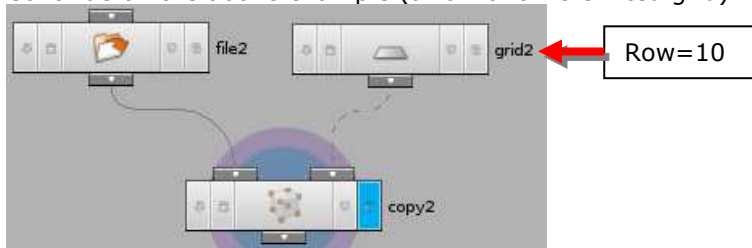
For example, generate large number of short-life particle

Remark: you may want to render the results of the particles, but display the result of COPY in order to speed-up the on-screen display. In Houdini, it can be done by:

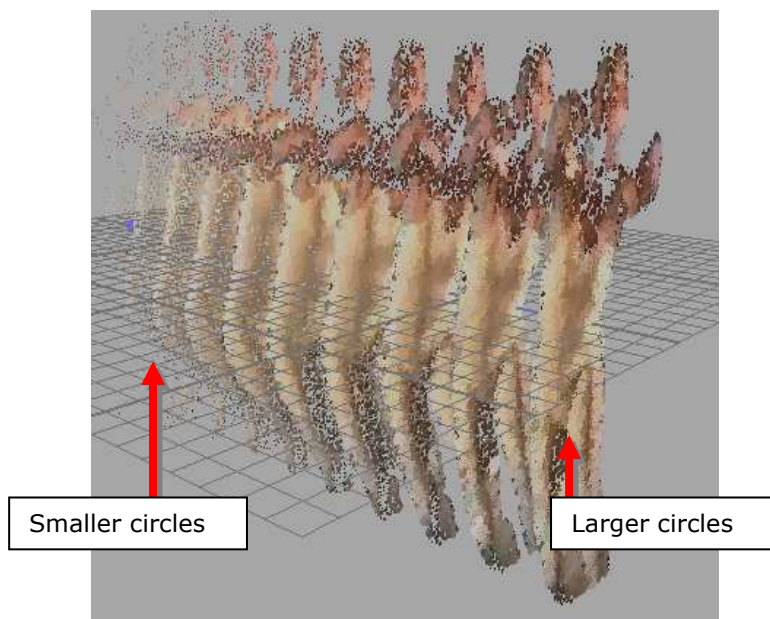


Exercise 2

Continue on the above example (a row of time-shifted grid):



1. Try to copy circle onto the points, where the size of the circle will depends on the luminance of the point.
2. Try to make the circles at the beginning of the row to be smaller, and the size becomes larger larger when it is at the end of the row:



**** Week 04 END ****