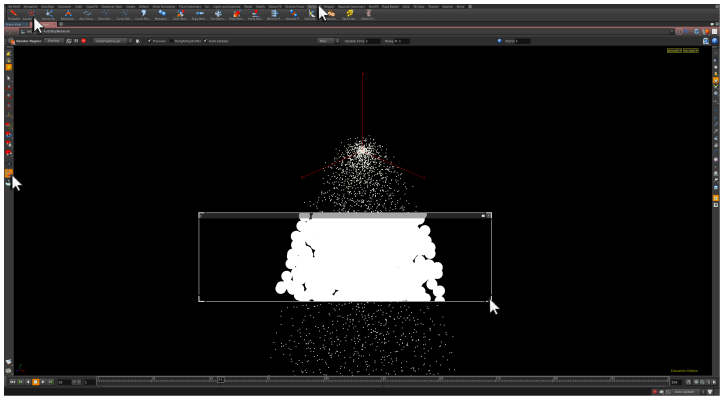
PARTICLE SPRITES

By default particles will render as spheres. The size of the particle spheres can be controlled using the **Point SOP** by activating the **pscale attribute**, and determining a value for their size. The **Copy SOP** also allows for any custom geometry to be copied onto particles for the purposes of rendering. **Sprites** are an alternate method of rendering particles, and can offer simple but effective solutions for effects such as water, spray, smoke and fire. A **sprite** is a **2D square** image that is automatically instanced onto each particle. The sprite square will **always face forward** towards the camera, and can be thought of as a **2D animation window**. Sprites are very **efficient to render**. As sprites are reliant on an image assigned to the particle, **Material properties** such as **Alpha** and **Specular Maps** should be utilised to disguise the appearance of the square based sprite.

In a **new Houdini scene**, **maximise** the **Scene Viewer** and activate the **Shelves**. From the **Particles Shelf**, **LMB** activate the **Location Particle Emitter** button. With the mouse over the Viewer, press **ENTER** to confirm the emitter location.

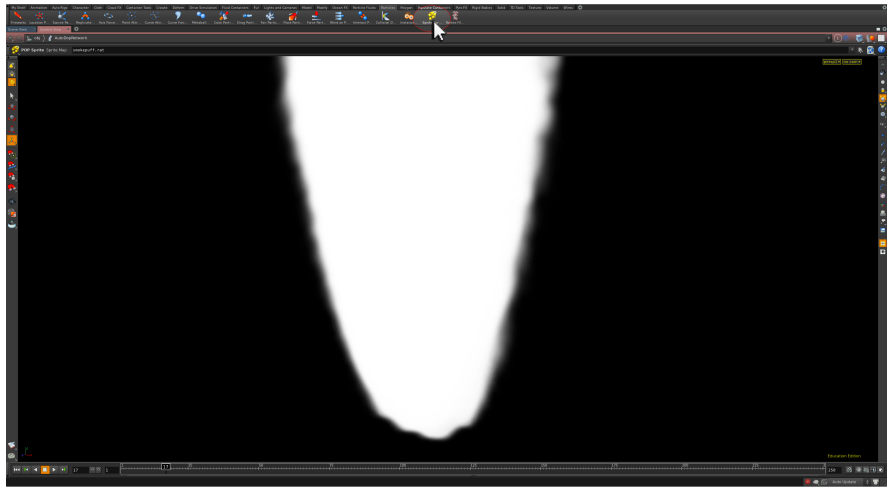


When **PLAY** is pressed, the particles emit from their birth location and fall downwards under the influence of gravity. A **Render Region Preview** reveals the **default spheres** assigned to particles at render time.

In the **Viewer**, **select** the **particles**, and from the **Particles Shelf** activate the **Force Particles button**. In the **Viewer parameters** for this tool, specify:

Force	**0**	**20**	**0**

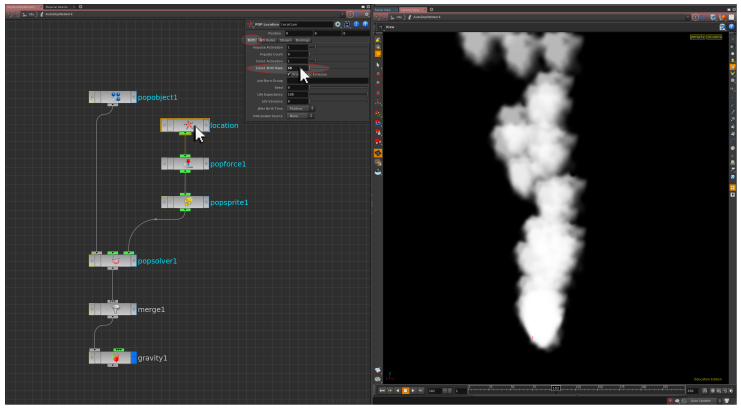
This will cause the particles to rise upwards instead of falling downwards.



With the particles selected, activate the Sprite Particles button. This will assign a **smokepuff.rat** image as a forward facing 2D square to each particle. When **PLAY** is pressed, a column of white smoke is created.

Inside the **AutoDopNetwork**, locate the **POP Location DOP** creating the particles, and in its **Birth parameters** specify:

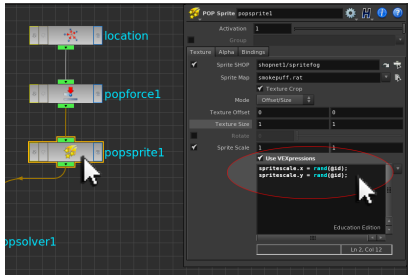
Const. Birth Rate **50**

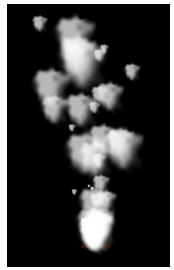


Reducing the number of particles being constantly birthed will make the effect of the sprites easier to see. Whilst the **smokepuff.rat** image does a create a semblance of basic smoke, the aesthetic of each sprite is however identical. Therefore good sprite work requires the modification of sprites on a per particle basis in order to achieve higher-level aesthetics from them.

In the parameters for the **POP Sprite DOP**, activate the **Use VEXpressions** option, and in the resulting Window enter the following commands:

spritescale.x = rand(@id) * 5;
spritescale.y = rand(@id) * 5;





This will override the default Sprite Scale parameter (internal name spritescalex, spritescaley) with random sprite sizes based upon the @id attribute (the unique particle number). When **PLAY** is pressed, each sprite now has its own random size. **Other POP Sprite DOP parameters** can be also overridden in this way to create further randomization and naturalization.

CONTROLLING SPRITE IMAGES

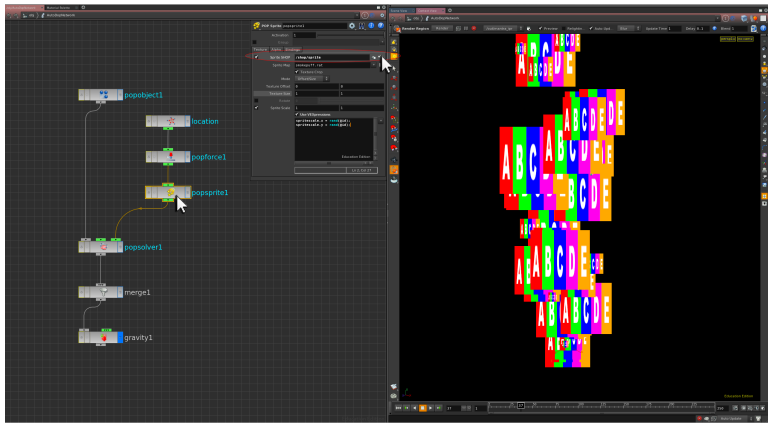
By default the **POP Sprite DOP** references **an internal sprite material**. As part of the Shelf Tool Sprite setup, an **end user sprite material** was created at **SHOP Level**. At **SHOP Level**, locate the **sprite Material**, and in its **parameters** specify:

Base Color Map **$HIP/ALL.rat**

This image is a ‘sprite sheet’ of five letters that can be used to demonstrate the principles of sprite texture manipulation.

Return back inside the **AutoDopNetwork** and in the **parameters** for the **POP Sprite DOP** specify:

Sprite SHOP **/shop/sprite**

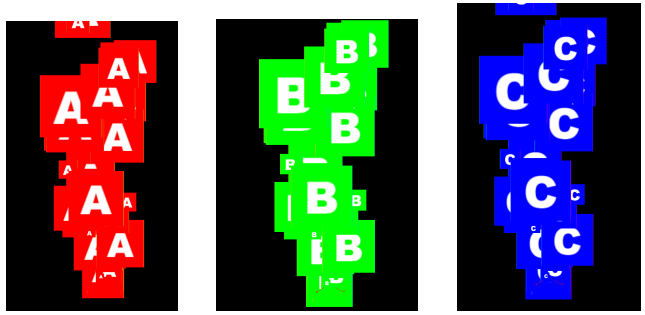


When **PLAY** is pressed, the **All.rat image** appears as **sprites** both in the Viewer and in a render.

In the **parameters** for the **POP Sprite DOP** specify:

Mode	**Sprite Sheet**	
Rows and Columns	**1**	**5**

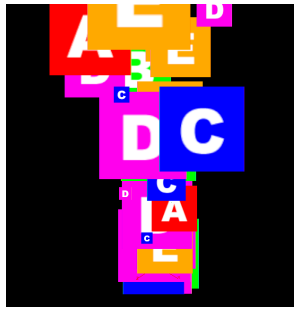
This will tell Houdini that a specific sized sprite sheet is being used. When **PLAY** is pressed, the **letter A** appears on each of the sprites.



Increasing the **Sprite Index parameter** will cycle through each section of the sprite sheet in turn. This parameter can also be overridden so that each sprite is assigned its own letter. In the **VEXpressions** window enter the following command:

textureindex = @id;

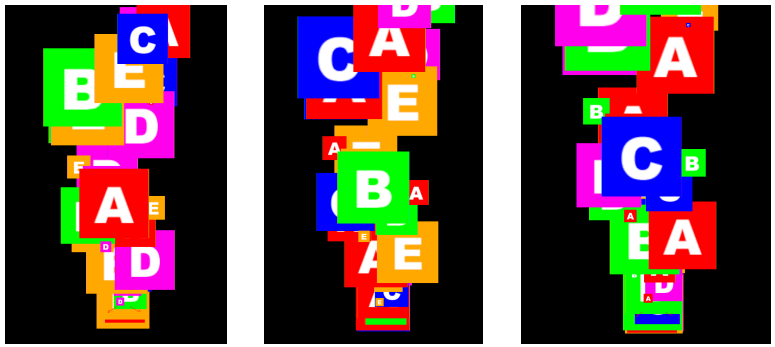
This will assign each particles unique number to the Sprite Index parameter (internal name textureindex). When **PLAY** is pressed, each sprite is now assigned its own letter.



This effect can further be enhanced by assigning scrolling animation to the texture index so that each sprite is assigned its own letter, but also cycles through each other letter in turn. In the **VEXpressions** window modify the above command to:

textureindex = @Frame + @id;

When **PLAY** is pressed, each particle scrolls through each image of the sprite sheet.

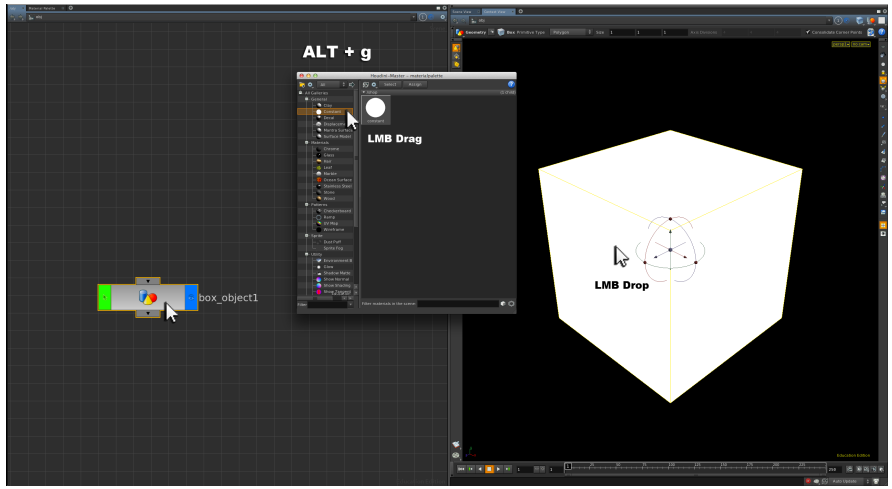


This effect could for example be used for a flock of background birds, where each sprite sheet image was a different part of a bird wing animation cycle.

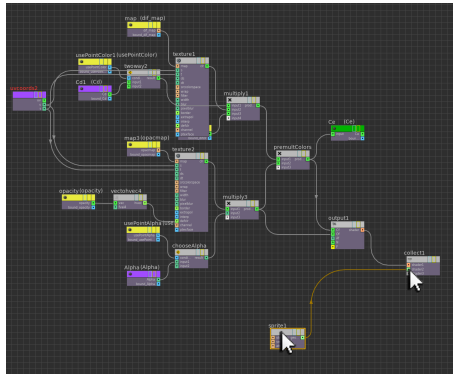
See file **H15_simple_sprites.hipnc**

CUSTOM SPRITES

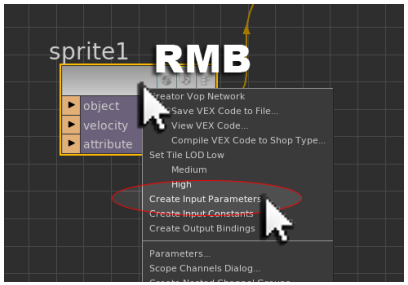
Sprites can also be assigned to any point based geometry, and can read in image sequences rather than sprite sheets. In a **new Houdini scene** create a **Box Object**. Press **ALT + g** to activate the **Material Palette**, **drag** and **drop** a **Constant Material** onto the Box Object.

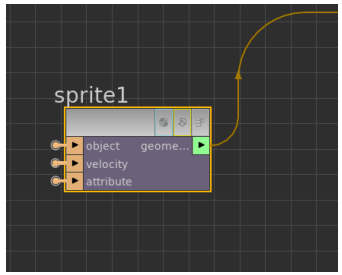


At **SHOP Level**, locate the **Constant Material**, and **double-click** on it to **go inside it**. In the **VOP Shader Network**, create a **Sprites Procedural VOP** and wire its **output** into the **second input** of the final **Collect VOP** of the shader network chain. This node will explicitly tell Houdini to render this material as sprites rather than as normal geometry.



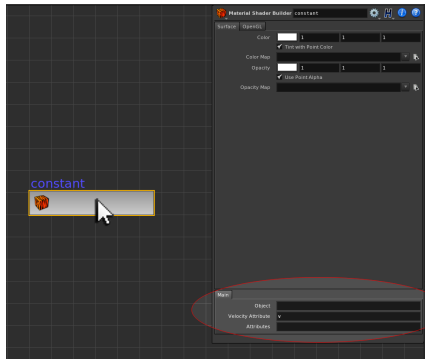
RMB on the **Sprites Procedural VOP** and from the resulting menu choose **VEX/VOP Options> Create Input Parameters**. This will create input nodules on the node.

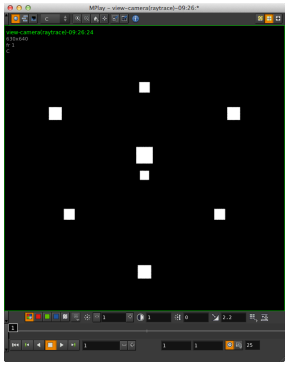




This nodules are hidden nodes **porting the parameters** of the Sprites Procedural VOP up to the **Constant Material's SHOP Level parameters**. The **Attribute parameter** will be used for declaring to Mantra any custom attributes specific for the sprite setup (such as sprite colours and textures).

When the **Constant Material** is examined at **SHOP Level**, the ported parameters from the Sprites Procedural VOP can be seen.





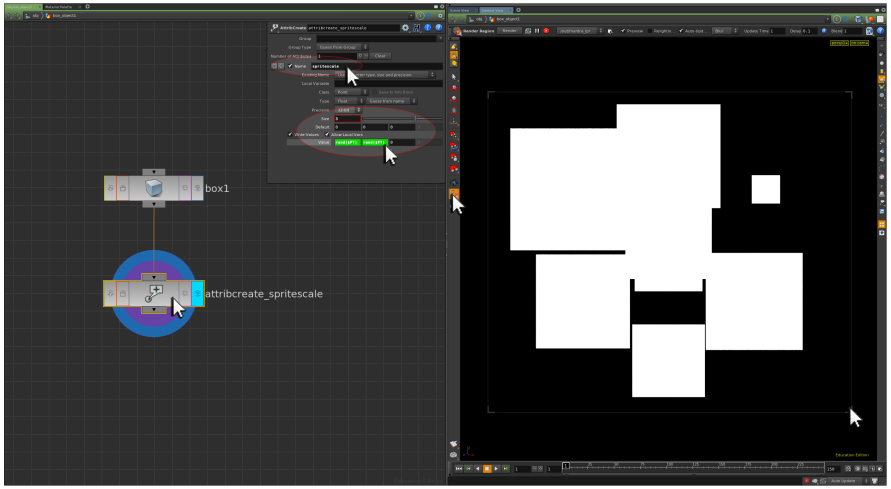
When the box object is rendered, its points now appear as sprites.

CREATING SPRITE ATTRIBUTES

At **Geometry Level** for the box object, **append** an **Attribute Create SOP** to the Box SOP. In the **parameters** for the **Attribute Create SOP** specify:

Name	**spritescale**		
Class	**Point**		
Type	**Float**		
Size	**3**		
Value	**rand($PT)**	**rand($PT)**	**0**

When a Render Region Preview is activated over the viewer, the box sprites now have random sizes.

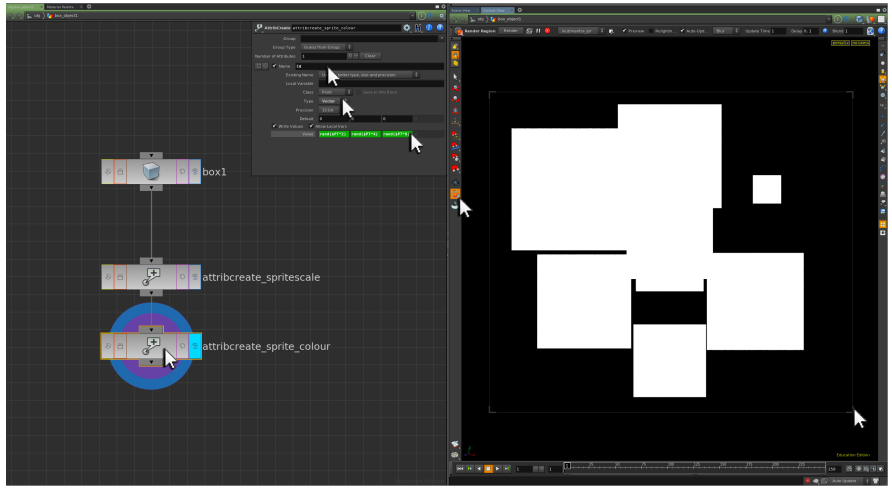


NOTE: The attribute **spritescale** is a default attribute created by the **POP Sprite DOP**. In this example however, it is being created by **Attribute Create SOP**. As Houdini has been told to render the geometry material as sprites (the Sprite Procedural VOP), Houdini will automatically implement any default sprite based Point Attributes it finds.

ASSIGNING CUSTOM SPRITE ATTRIBUTES

At **Geometry Level** append to the network a second **Attribute Create SOP**. This will be used for assigning a different colour to each sprite. In the **parameters** for the second Attribute Create SOP specify:

Name	**Cd**		
Type	**Vector**		
Value	**rand($PT*2)**	**rand($PT*4)**	**rand($PT*8)**



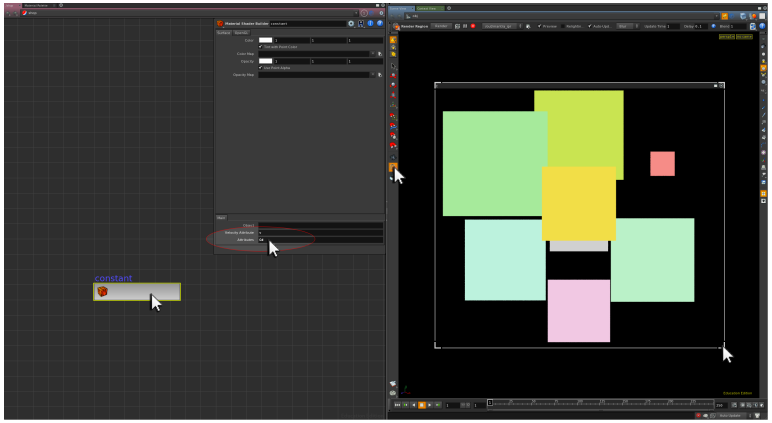
If the box geometry is rendered however, no sprite colour will appear. This is because Cd is not a Point Attribute anticipated in sprite based rendering, and as a result it is ignored.

INFORMING SPRITES ABOUT CUSTOM ATTRIBUTES

In order for Cd to be recognised as a sprite attribute for rendering, it must be declared to Houdini. This is done in the (VOP promoted) **Attribute Parameter** on the **Constant Material**. Return back **SHOP Level**, and in the **parameters** for the **Constant Material** specify:

Attributes	**Cd**

When the scene is rendered, the random colour appears on the sprites.

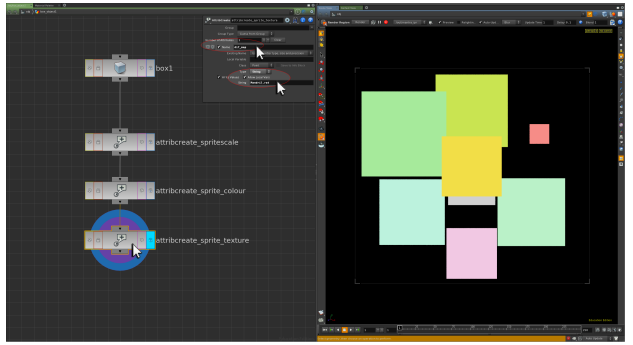


CREATING INDIVIDUAL SPRITE TEXTURES

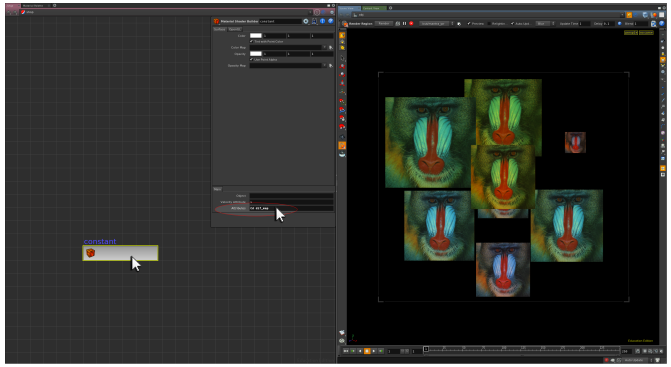
At **SHOP Level** examine the **parameters** for the **Constant Material**. Note that the **internal name** given to the **Color Map parameter** is called **dif_map**. Return to **Geometry Level** and create a **third Attribute Create SOP**. In the parameters for this node specify:

Name	**dif_map**
Class	**Point**
Type	**String**
String	**Mandril.rat**

This will create a String (text) Attribute for each point on the geometry reading in an image. This custom attribute must however be declared in the **Attributes Parameter** found on the **Constant Material** before it will appear in the render.



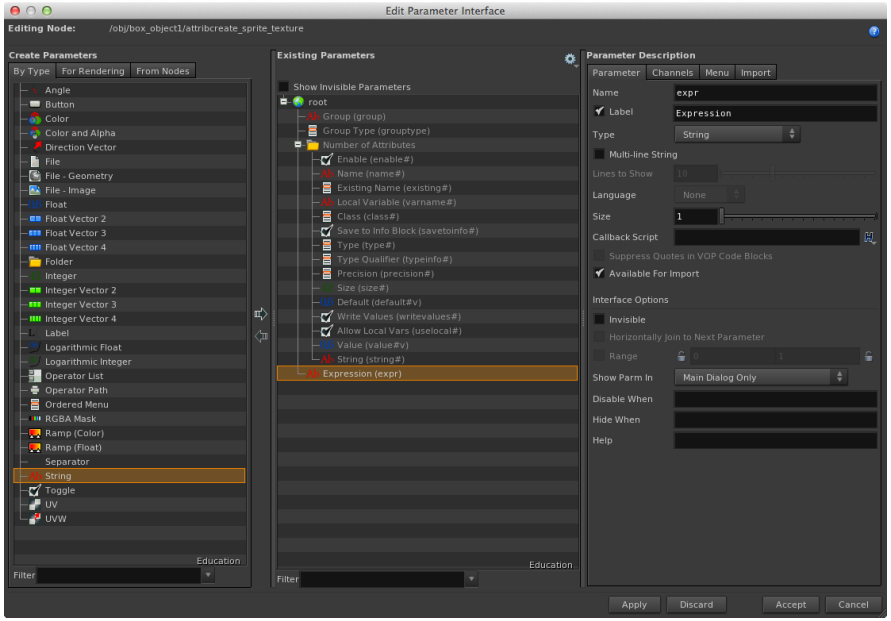
This can either be done by **adding the name dif_map** to the **Attributes listing** or by using the **wildcard *** to instruct the renderer to use all custom attributes it finds on the geometry when rendering the sprites. Care should however be taken when using the wildcard * in case attributes are evaluated unnecessarily.



Now when the scene is rendered, all of the sprites render with the Mandril.rat image, multiplied by the sprites own individual random colour.

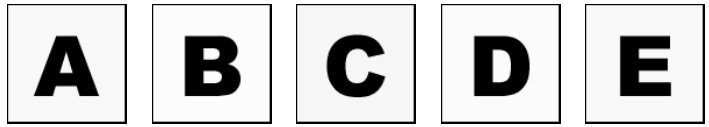
SETTING A FRAME SEQUENCE

Activate the **Edit Parameter Interface window** for the **sprite texture Attribute Create SOP** (the Cog button found on the parameters).



From the **Create Parameters By Type** list select a **String parameter type** and **port it across** to the **Existing Parameters** list. Set the **Name** for this string parameter as **expr**, and the **Label** as **Expression**. Press the **Accept** button to activate this additional parameter. This custom parameter will be used to control image behavior on each sprite.

The following image sequence can be used for testing custom textures on sprites:

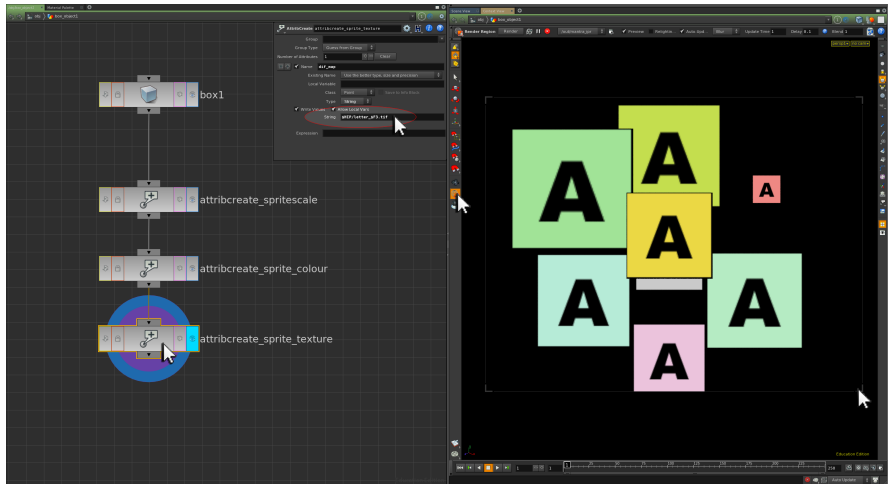


letter_001.tif letter_002.tif letter_003.tif letter_004.tif letter_005.tif

Save the current scene in the **same location** as this **letter image sequence**. In the **parameters** for the **sprite texture Attribute Create SOP** specify:

String **$HIP/letter_$F3.tif**

When the scene is rendered out each sprite will be assigned the letter A if the timeline is set to frame 1.



In the **parameters** for the **sprite texture Attribute Create SOP** specify:

String **$HIP/letter_`chs("expr")`.tif**
Expression **$F3**

NOTE: Back-ticks(`) must be used around the chs() function in order for it to be evaluated as an expression rather than simply text. The chs() function will return the value of any parameter in a string based form.

The scene will render out as before; however modifying the expression controlling the image now becomes easier to manage.

THE PADZERO() FUNCTION

As mathematical modification of the current frame can only be done with $F, the **padzero() function** can be utilised to convert the value returned by $F into its padded frame number equivalent. In the **parameters** for the **sprite texture Attribute Create SOP** specify:

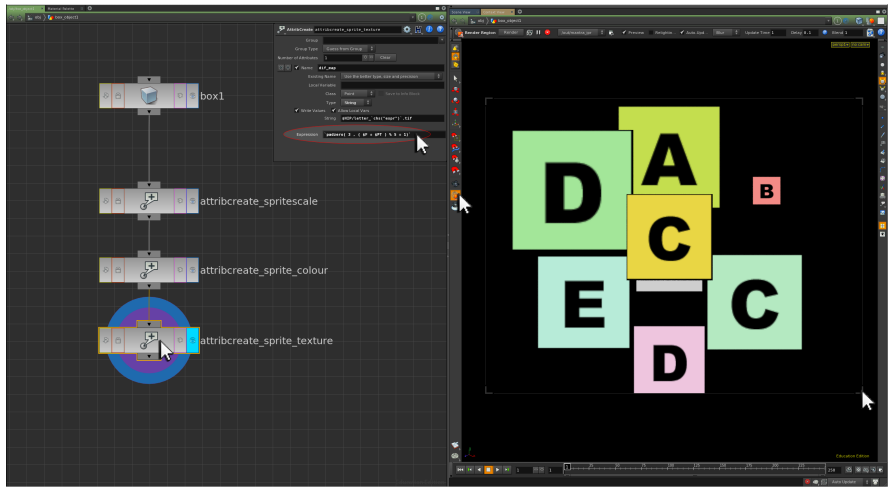
Expression **`padzero(3,$F)`**

The scene will still render as it did before; however now further mathematical modification can be done to $F to assign an individual frame to each sprite.

NOTE: As the Expression parameter is a String Type, back-ticks (`) must be used in order to evaluate the contents of the parameter as an expression rather than as a line of text.

In the **parameters** for the **sprite texture Attribute Create SOP** specify:

Expression **`padzero( 3 , ( $F + $PT ) % 5 + 1)`**



This will assign an individual frame to each sprite dependant upon the current frame + the current point being evaluated. This can be verified when the scene is rendered. **See file H15_custom_sprites.hipnc**

SPRITES IN PRODUCTION

Sprites are a very versatile and efficient tool, and can very often provide effective solutions at a low production cost.

In **Superman Returns (2006)**, sprites were used to generate a field of corn that a young Clark Kent leaps over whilst discovering his super powers. This use of sprites saved creating a massive amount of corn plant geometry.

In **Lord of the Rings – Fellowship of the Ring (2001)**, sprites were used to create the fire monster Balrog.

This production decision was taken after attempts with higher-level fluid simulations failed to create cost effective fire relative to production. Many months were taken attempting to create fluid fire; however the realization of this scene using sprites took a matter of weeks.