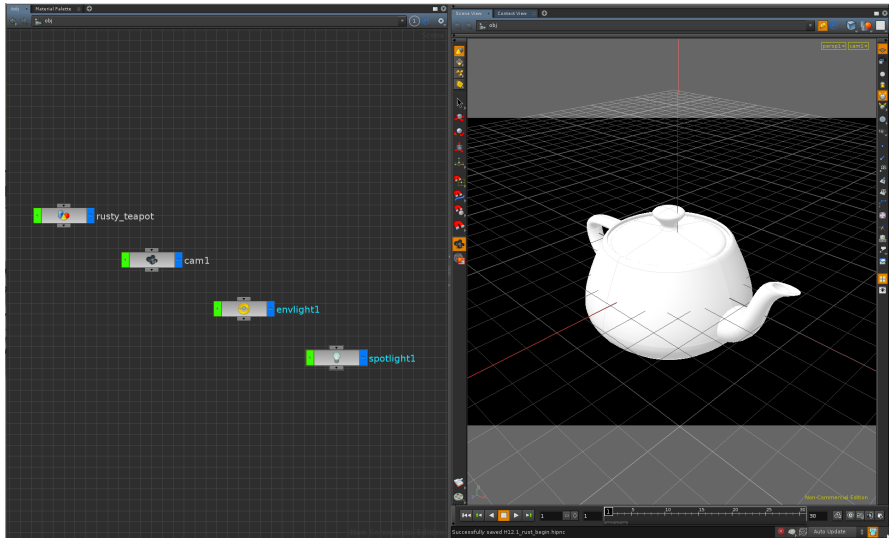
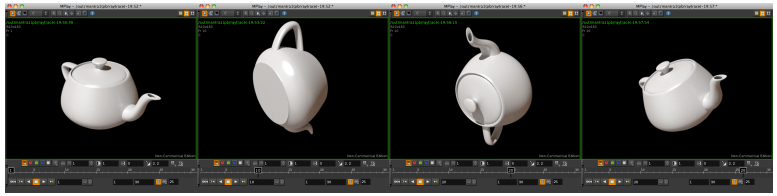
This shader writing lesson will look at the configuration of a rusty teapot Material. Open the scene **H14_rust_begin.hipnc**.



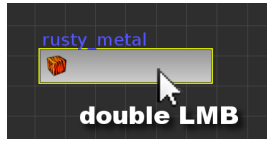
This scene contains a simple animated teapot with a **default Surface Model Material** assigned to it. The scene is lit with an **Environment Light & Map** plus a **Spotlight**. A **Mantra PBR ROP** node has been configured to **Render 30 frames**.



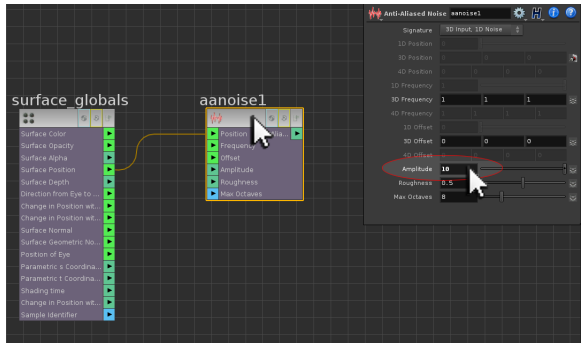
CREATING A RUST PATTERN MASK

When developing custom materials it is useful to understand the differences between a surface aesthetic and its surface aesthetic shape. For a rusty teapot, a good place to start is the generation of a B/W rust pattern mask that will simply define surface aesthetic shape areas of rust and non-rust.

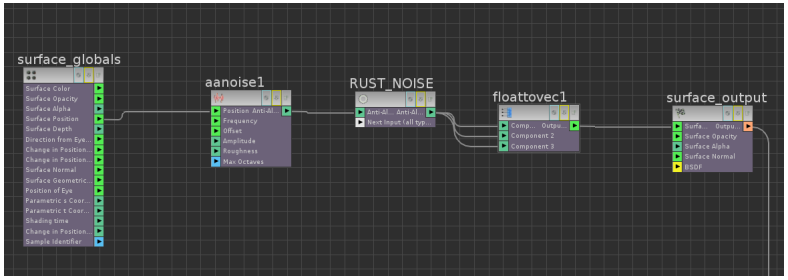
Switch to **SHOP Level** and **double LMB** on the **Mantra Shader Builder** node to go inside it.



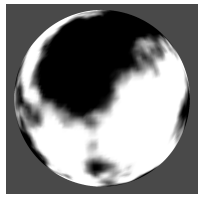
At **Shader Level**, create an **Anti-Aliased Noise VOP**, and wire the **Surface Globals > Surface Position** output (**P**) as its first input. In the **parameters** for the **Anti-Aliased Noise VOP**, specify an **Amplitude** of **10**.



Append a **Null VOP** to the **Anti-Aliased Noise VOP**, and rename it to **RUST_NOISE**.

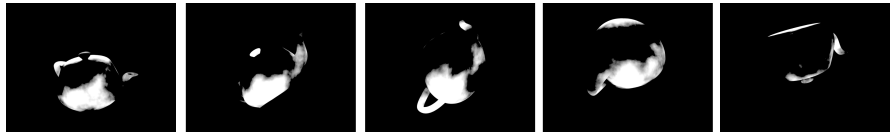


To see the effect of the noise on the surface geometry, the output of the **Null VOP** can be wired into a **Float to Vector VOP**. This can in turn be passed into the **Surface Color input** of the **Surface Output VOP** to see it previewed on the Shader Ball. This pattern will form the basis of the rust regions for the teapot.



NOTE: Temporarily disconnect the Surface Model VOP from the Surface Output VOP.

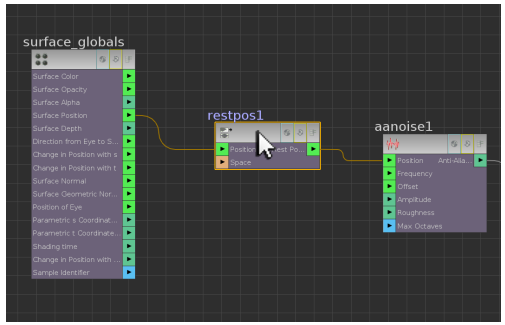
From the main **Render Menu** choose **Edit Render Node > Mantra1**. From the resulting Render Options set the **Rendering Engine** parameter to **MicroPolygon Rendering**. This will **temporarily deactivate Physically Based Rendering**, allowing for examination of black and white rust noise mask.



When the scene is rendered however, the noise swims through the animated teapot.

THE REST POSITION VOP

In order to fix the noise pattern to the surface of the teapot, a Rest Position VOP can be used.



RMB on the **Surface Position output** of the **Surface Globals VOP** to insert a **Rest Position SOP**. In its **parameters** specify:

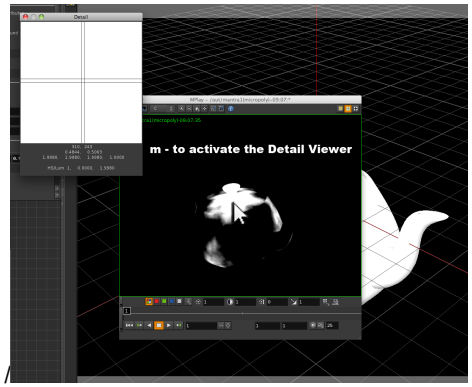
Space **Object**

This will convert the incoming geometry position data to **Object Space**, ensuring that the World Space animation of the teapot is not factored in its rendering.

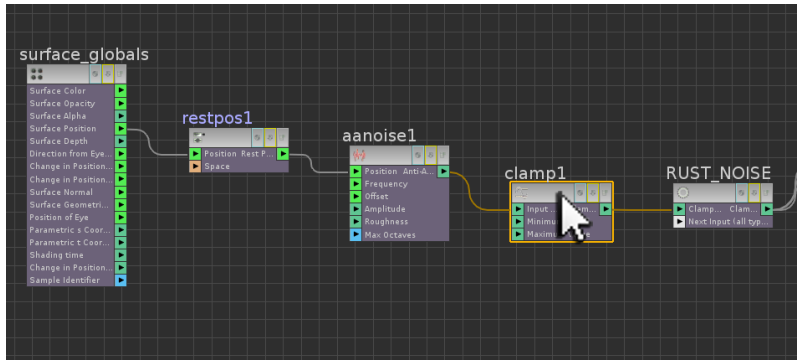
When the scene is rendered again, the noise sticks to the surface of the teapot. This black and white noise information can be used as a mask determining rusty and non-rusty areas of the teapot.

CLAMPING VALUES

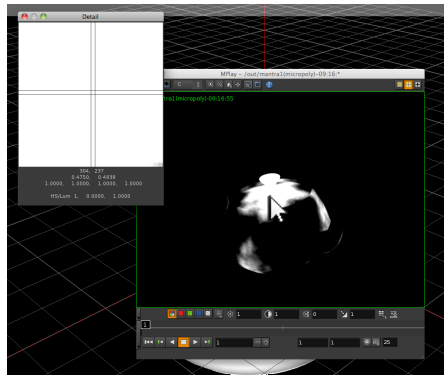
With an **MPlay** render window active, **m** can be pressed to activate the magnified **Detail Viewer**. This viewer returns the numeric values of the rendered image.



Examination of these values reveals that the white regions go above a value of 1, and the black regions go below a value of 0. Inserting a **Clamp VOP after** the **Anti-Aliased Noise VOP** will lock the colour range explicitly between 0 and 1.

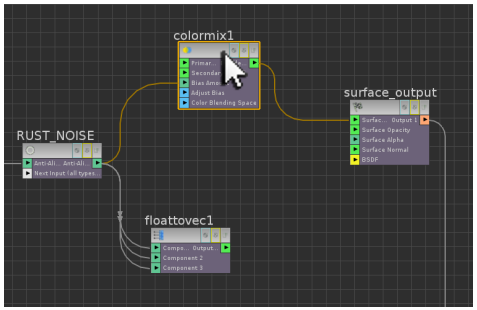


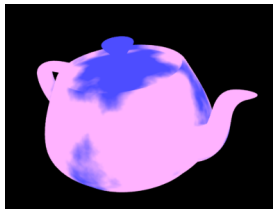
When the scene is rendered again, the values being returned by the Anti-Aliased Noise are now between 0 and 1.



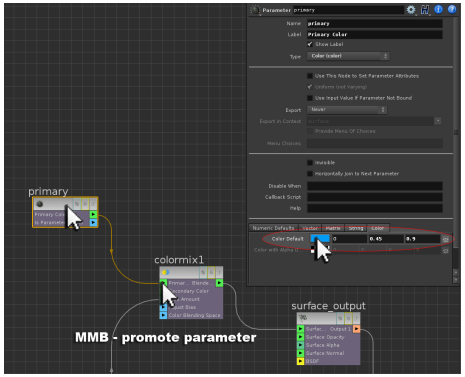
DEFINING THE RUSTY REGIONS

Unwire the **output** of the **Float to Vector VOP** from the **Surface Output** node, and in its place create a **Color Mix VOP**. Wire the **output** of the **Color Mix VOP** as the **Surface Color Output**, and wire the **output** of the **RUST_NOISE Null VOP** as the **Bias Amount input** of the **Color Mix VOP**.



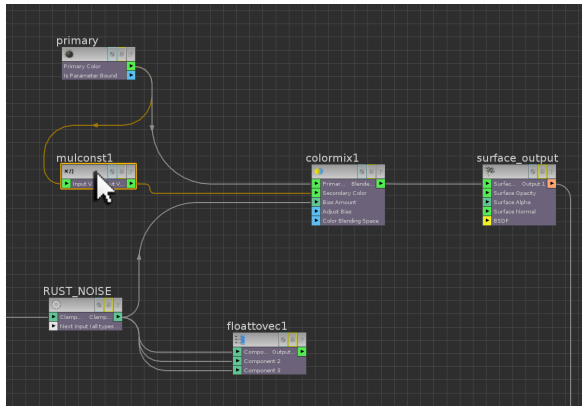


When the scene is rendered again, the noise pattern now controls the distribution of the default Color Mix VOP colours.



MMB on the **Primary Color input** of the **Color Mix VOP** and choose **Promote Parameter**. This will automatically configure a **Parameter VOP** as an end user control (displayed as a parameter nodule). **Double LMB** on the **parameter nodule** to **expose** the promoted **Parameter VOP**, and in its **parameters**, set the primary colour to a **mid blue**.

MMB on the output of the **Primary Color Parameter VOP** and create a **Mulitply Constant VOP** as a new network branch. Wire the **output** of the Multiply Constant VOP as the **Secondary Color input** of the **Color Mix VOP**.



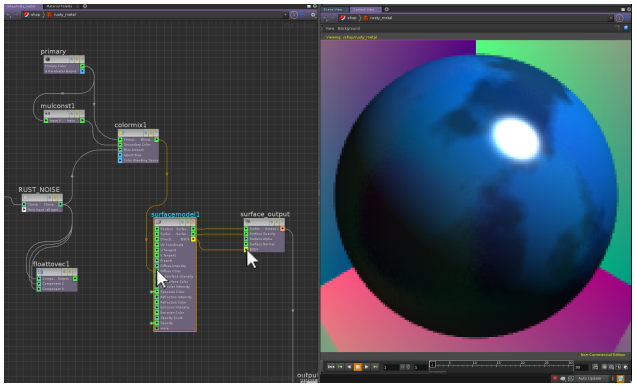
In the **parameters** for the **Multiply Constant VOP** specify:

Multiplier **0.5**

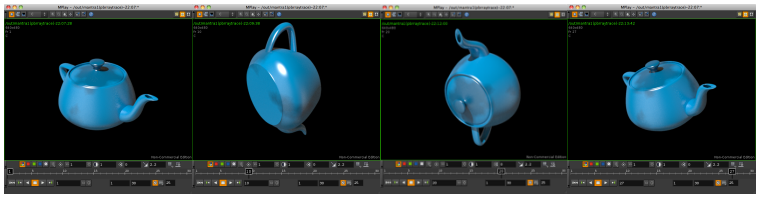
This will darken the original blue in the areas of the teapot affected by rust.

CHECKING THE MATERIAL

Wire the output of the **Color Mix VOP** directly into the **Diffuse Color Input** of the **Surface Model VOP**. This will destroy the default diffuse color parameter nodule.



Reconnect the Surface Model VOP outputs to their corresponding inputs on the Surface Output VOP. Activate the **Rendering Parameters** for the **Mantra1 ROP** and reset the **Rendering Engine** as **Physically Based Rendering**. The sequence can be rendered again to see how it responds as a light responsive Material.

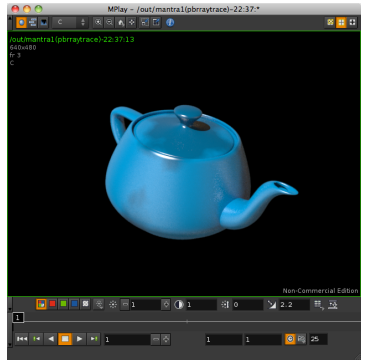


See file H14_rust_stage1.hipnc

IMPORTANT NOTE: The **rusty metal material** will return a **yellow warning error** until the new **Material Level Primary Color** parameter is **channel referenced** into the **Material Level OpenGL > ogl diffuse parameter** to replace the destroyed diffuse default parameter channel reference.

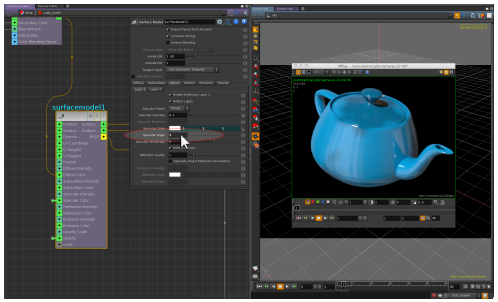
CONTROLLING THE SURFACE AESTHETIC

When a render of the scene is generated, it becomes evident that the specular properties of the surface also need modulating by the B/W rust pattern mask; so that the darker regions do not return any reflections or specular highlights.

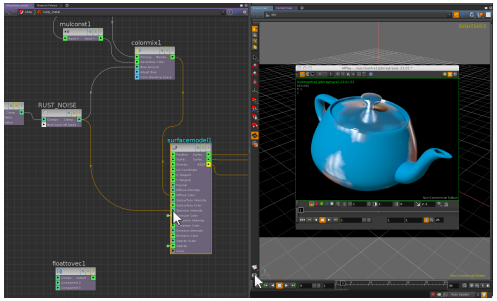


Similarly, the reflective properties of the surface can be enhanced in areas of the surface not yet affected by rust. Back inside the Material at **VOP Level**, **activate** the **parameters** for the **Surface Model VOP**. These are the **same parameters** as on a standard Mantra Surface Material, and can be used to create the lighting model for the rusty metal material.

Set the **Reflect > Layer 1 > Specular Angle** parameter to **3**. This will create a much sharper set of reflections in the surface of the teapot, as well as much sharper specular highlights.

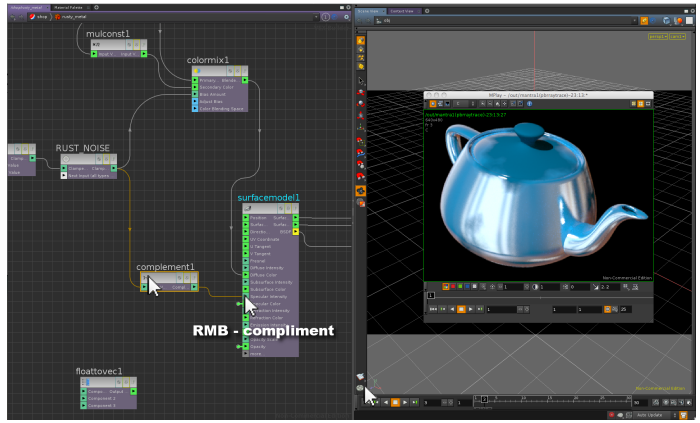


Wire the **output** of the **RUST_NOISE Null VOP** into the **Specular Intensity input** of the **Surface Model VOP**.



When the **scene** is **rendered**, the reflections and specular are modulated by the RUST NOISE mask; however are in the darker regions instead of the lighter regions.

RMB on the **Specular Intensity input** of the **Surface Model VOP** and **insert** a **Compliment VOP**. This will invert the B/W rust noise pattern.

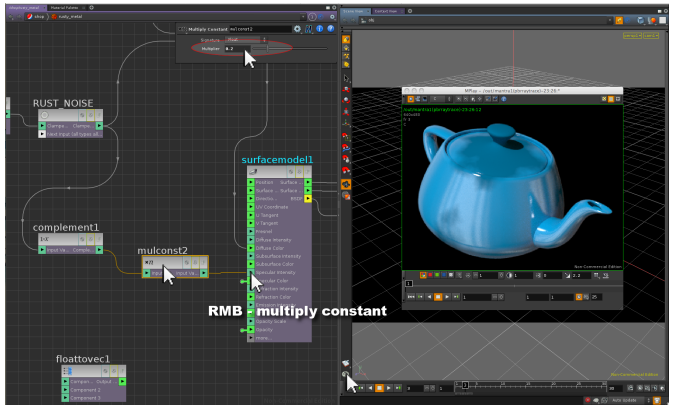


When the scene is rendered again, the highly reflective surface now resides in the correct regions. The reflections are however too bright. **NOTE: When diffuse values are low, and reflectivity high, mirrored surfaces such as chrome can be achieved.**

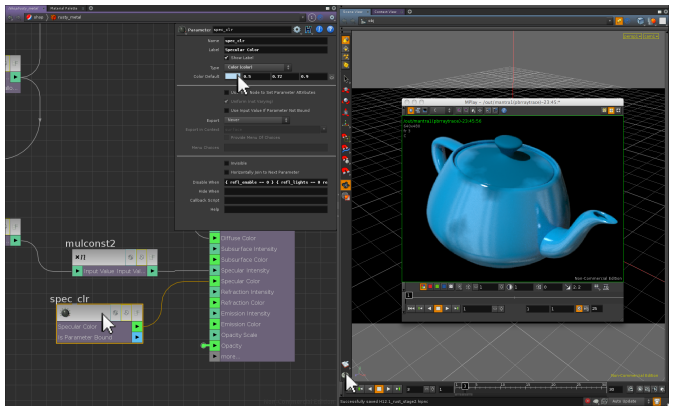
Insert a **Mulitply Constant VOP after** the Complement VOP. In its **parameters** specify:

Multiplier **0.2**

This will reduce the intensity of the reflections down to something more appropriate.



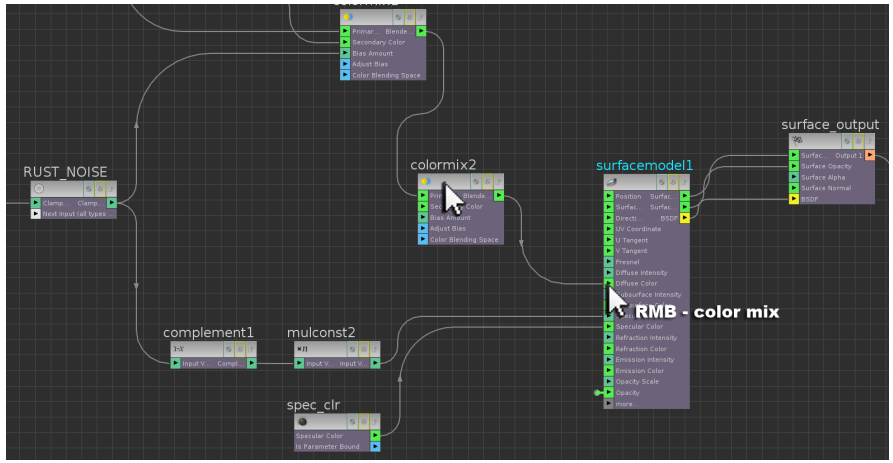
The Specular Color parameter nodule can also be exposed and the Color parameter adjusted to again modulate the specular and reflections slightly. Setting a light blue helps diminish the stronger yellows from the reflection.



See file H14_rust_stage2.hipnc

DEFINING THE RUST REGIONS

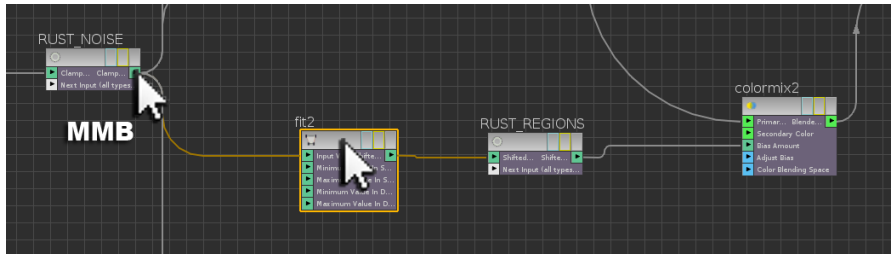
Currently the dull areas of the teapot do not contain any proper rust regions. The RUST_NOISE Null VOP pattern can be modified to create tighter rust regions inside the dull areas. **RMB** on the **Diffuse Color input** of the **Surface Model VOP** to insert a second **Color Mix VOP** into the network.



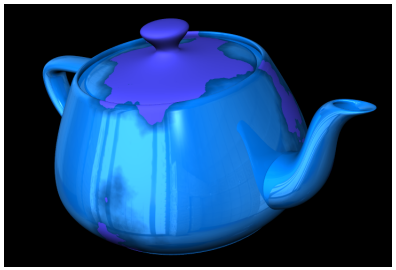
This can be used to mix between the current teapot surface, and some tighter defined rust regions. **MMB** on the **output** of the **RUST_NOISE Null VOP** to create a **Fit Range VOP** as a new network branch. In the **parameters** for the **Fit Range VOP** specify:

1D Defaults	
1D Source Min	**0.7**
1D Source Max	**0.725**

This will limit the incoming noise pattern to just this range, creating a 0-1 output; visually resulting in tighter rust islands than the previous noise pattern was generating.



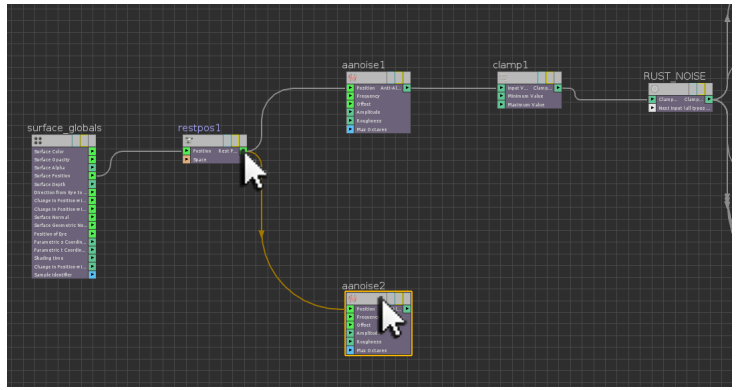
Append to the **Fit Range VOP** a second **Null VOP**, and **rename** the NULL VOP to **RUST_REGIONS**. This output can now be used as the **Bias Amount** for the **second Color Mix VOP**.



When the scene is rendered again, tighter purple rust regions now sit inside the rust noise areas. **See file H14_rust_stage3.hipnc**

CREATING THE RUST PATTERN

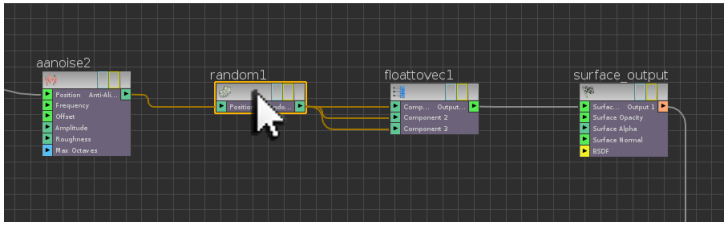
Now the rust regions have been formally defined using the second Color Mix VOP; the purple secondary colour can be replaced by a rust pattern.



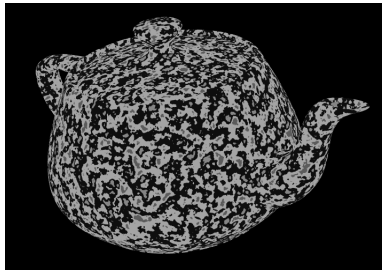
MMB on the **output** of the **Rest Position VOP** generating the RUST_NOISE pattern, and create a **second Anti-Aliased Noise VOP** as a **new network branch**. Utilizing the output of the Rest Position VOP will ensure this new noise pattern sticks to the surface of the teapot. In the parameters for this **second Anti-Aliased Noise VOP** specify**:**

3D Frequency	**30**	**30**	**30**
Amplitude	**5**		

Append a **Random VOP** to the output of this second Anti-Aliased Noise VOP. This will convert the noise into 0-1 random numbers based on the noise pattern input.

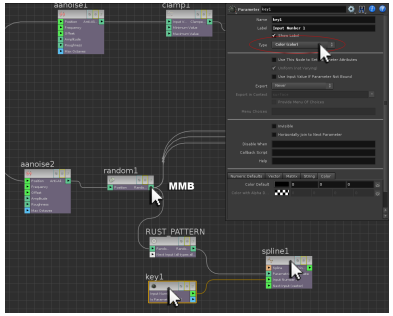


A Float to Vector VOP can be used to temporarily wire the output of the Random VOP into the Surface Output of the shader.

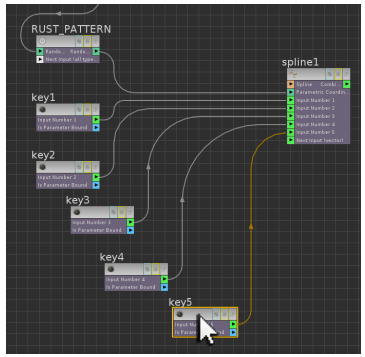


When a **default Mantra render** of the scene is generated (as opposed to the Mantra PBR render output), the teapot is covered by a black and white rusty noise pattern.

MMB on the **output** of the **Random VOP** and append a **Null VOP** as a new network branch. Rename this Null VOP to **RUST_PATTERN**. Create a **Spline VOP** as a separate node, and wire the **output** of the **RUST_PATTERN Null VOP** as its **Parametric Coordinate input**. The Spline VOP can be used to assign colour to the rust pattern.



MMB on the **Input Number 1 input** of the **Spline VOP** to **promote the parameter**. **Double LMB** on the resulting **nodule** to **expose** the newly created **key1 Parameter VOP**. In the **parameters** for the **key1 Parameter VOP** specify the parameter Type as **Color**. This will convert all key inputs of the Spline VOP to a vector type.

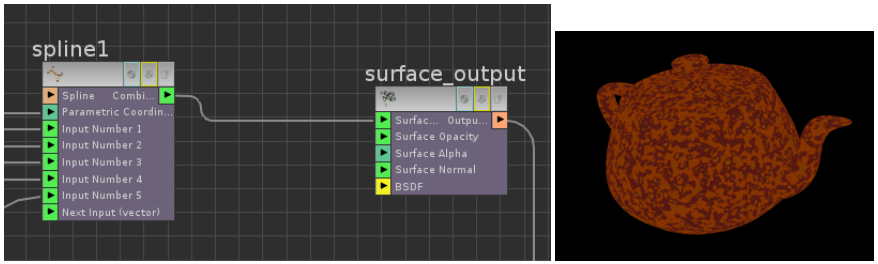


Continue to **promote the Input parameters** for the **Spline VOP**, creating a total of **5 key Parameter VOPs**. In the parameters for each new key, specify a **parameter type** of **Color**.

Set the **colour** values of each key to the following colours:

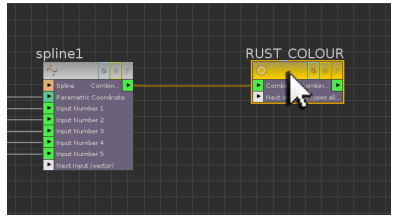
Key 1	**0**	**0**	**0**
Key 2	**0.075**	**0.04**	**0.02**
Key 3	**0.2**	**0.075**	**0**
Key 4	**0.2**	**0.025**	**0**
Key 5	**0**	**0**	**0**

NOTE: The first and last key colour will not appear in the render.

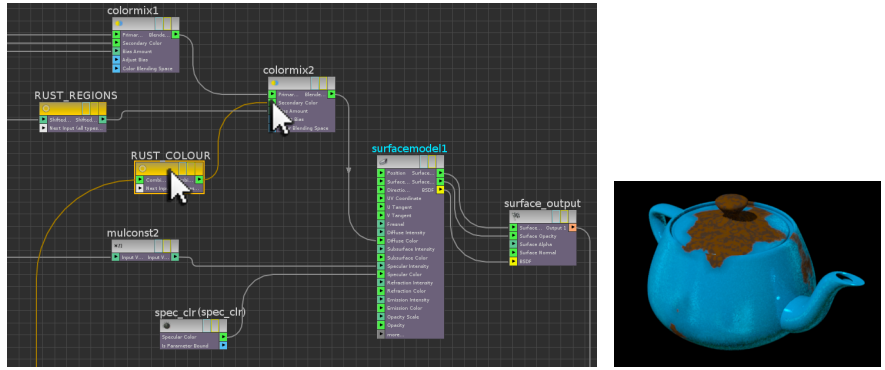


The **output** of the **Spline VOP** can now be temporarily wired into the **Surface Output** in order to see the rust colour pattern (use a standard Mantra Render rather than a Mantra1 PBR Render).

As a final step, a Null VOP renamed to RUST_COLOUR can be created for the output of the Spline VOP.



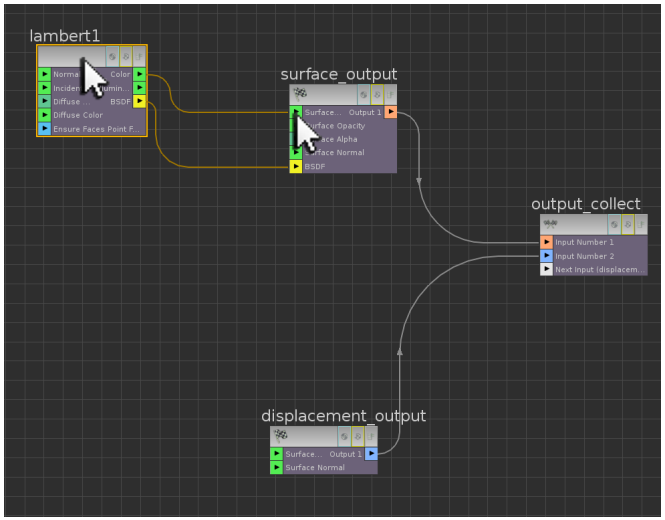
The output of the RUST_COLOUR Null VOP can then be used as the secondary colour input of the second Color Mix VOP. This will colour the RUST_REGIONS with the RUST_COLOUR.



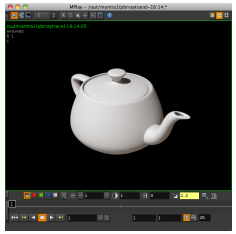
See file H14_rust_stage4.hipnc

ADDING DISPLACEMENT

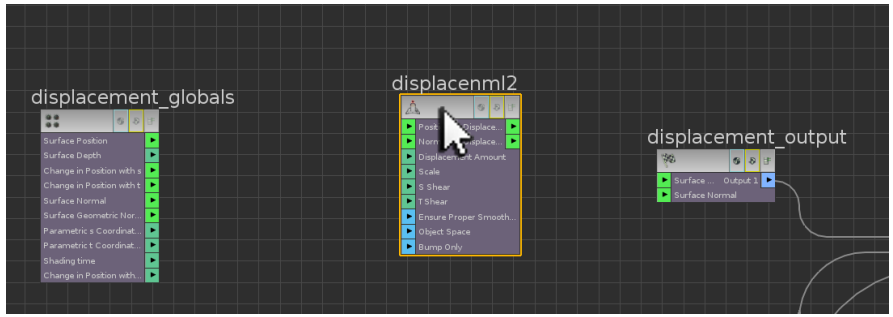
In order to get the rust island regions to retreat from the overall surface of the teapot, displacement can be used. Displacement is different from Bump Displacement, as the shape of the geometry is modified (rather than just the surface normal information). As all of the rust pattern information has been generated for the surface colour of the rusty teapot, this same pattern information can also be used to drive displacement of the surface.



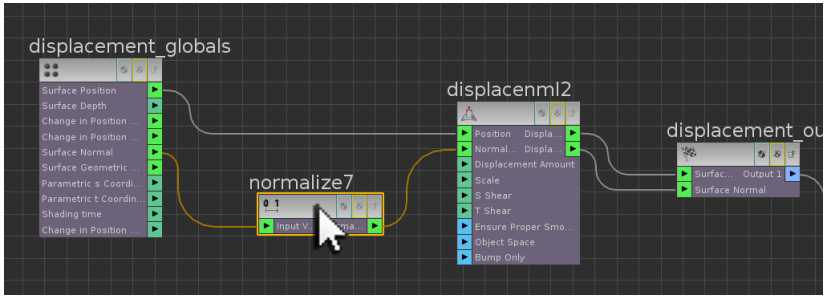
Disconnect the output of the Surface Model VOP from the Surface Output VOP, and in its place create a Lambert VOP. This will act as a temporary neutral surface for the teapot while the displacement is being configured.



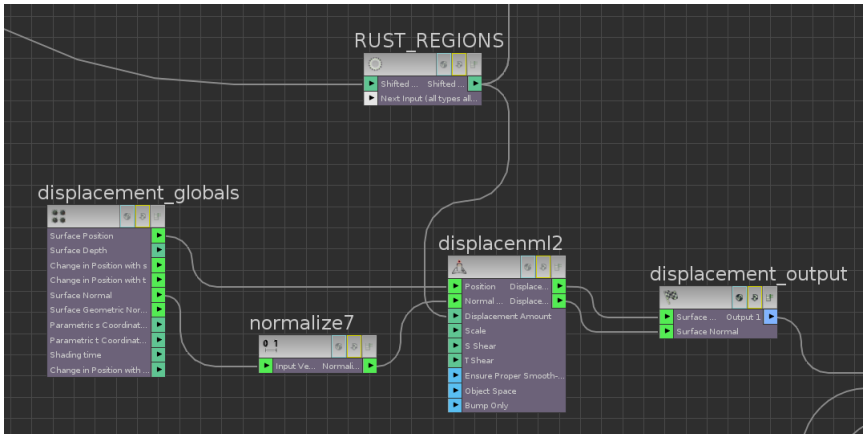
In the **Network Editor**, locate the **displacement_globals VOP** and the **displacement_output VOP**. Between these two nodes, create a **Displace Along Normal VOP**.



Using the **displacement_globals VOP** wire the **Position** and **Normalized Normal information** into the **Displace Along Normal VOP**.



Wire the **two outputs** of the **Displace Along Normal VOP** into the **displacement_output VOP**. While this alone is not enough to cause displacement, the rust regions can now be used to generate displacement.



Wire the output of the **RUST_REGIONS Null VOP** into the **Displacement Amount input** of the **Displace Along Normal VOP**.

In the **parameters** of the Displace Along Normal VOP, specify:

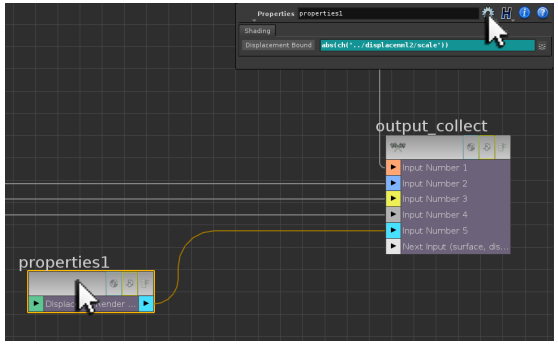
Scale **-0.0125**

When the scene is rendered, the rust regions now displace inwards the areas of the teapot affected by the rust; however displacement tearing is occurring.

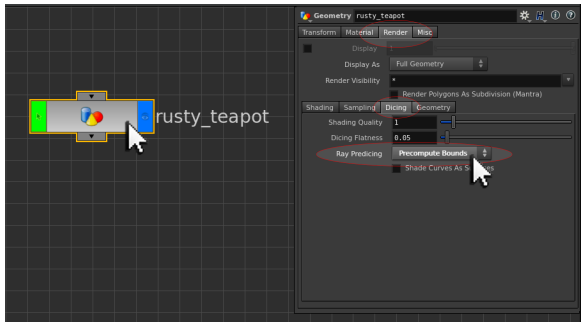
To prevent displacement tearing, Mantra must be informed that displacement is taking place. This can be done by the addition of a **Properties VOP**. Doing this will also correct any displacement render artefacts occuring.

In the **Network Editor**, locate the **output_collect VOP** and alongside it create a **Properties VOP**. Wire the **output** of the **Properties VOP** into the **Next Input** of the **output_collect VOP**.

Using the **Cog Menu** of the **Properties parameters**, activate the **Edit Rendering Parameters** window, and port across the hidden **Mantra > Shading > Displacement Bound parameter**.



With this hidden parameter is activated, a **channel reference** to the **Displacement Scale parameter** of the Displace Along Normal VOP creating the displacement can be made. **Wrap** this **channel reference** in an **absolute function** in order to achieve only positive values.

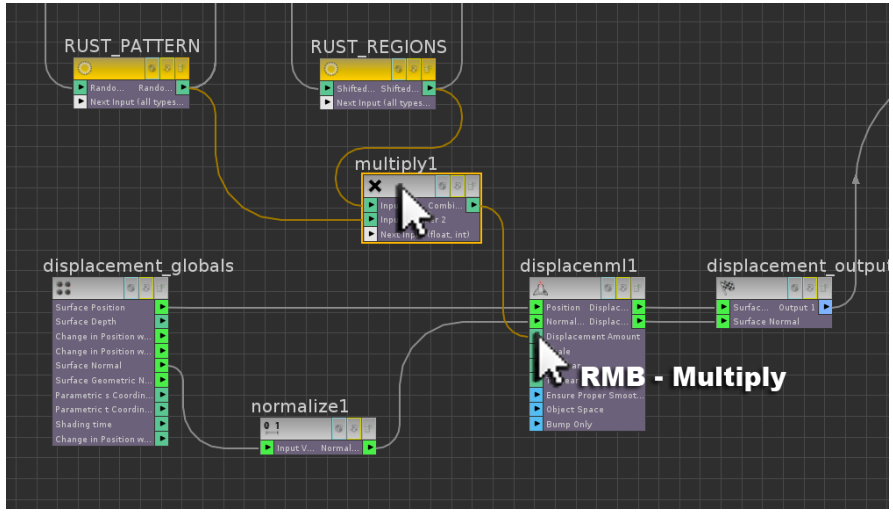


As a final step, set the r**usty_teapot's Object Level Render > Dicing > Ray Predicing parameter** to **Precompute Bounds**.

INTEGRATING THE RUST PATTERN INTO THE DISPLACEMENT

Currently only the RUST_REGIONS are being defined as displacement. The RUST_PATTERN can also be incorporated into the displacement effect by multiplying it with the RUST_REGIONS before they are fed into the Displacement Amount parameter of the Displace Along Normals VOP.

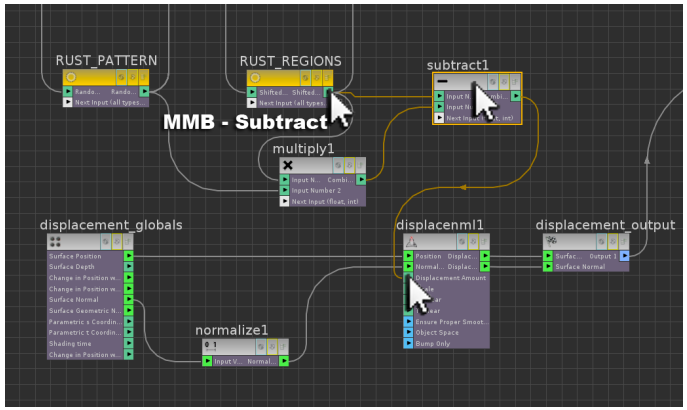
RMB on the **Displacement Amount** input of the Displace Along Normals VOP to **insert** a **Multiply VOP** after the RUST_REGIONS Null VOP.



The **RUST_PATTERN Null VOP** can then be wired as the **second input**. When the scene is rendered, the rust pattern is now evident as displacement within the rust regions.

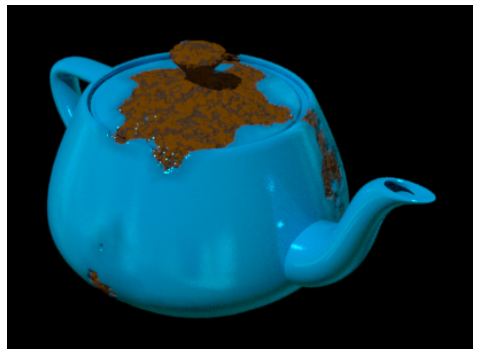
While doing this has created the overall displacement effect, the sense of imprinting of the RUST_REGIONS into the surface of the teapot has been lost. This imprinting can be achieved by subtracting the combined multiplication of the RUST_REGIONS and RUST_PATTERN from the original RUST_REGIONS Null SOP.

MMB on the output of the **RUST_REGIONS Null SOP** to create a **Subract VOP** as a **new network branch**. Wire the **output** of the **Multiply VOP** as its **second input**.



The **output** of the **Subtract VOP** can then be fed into the **Displacement Amount** input of the Displace Along Normals VOP. When the scene is rendered again, the effect of both the RUST_REGIONS and RUST_PATTERN can be seen in the displacement effect.

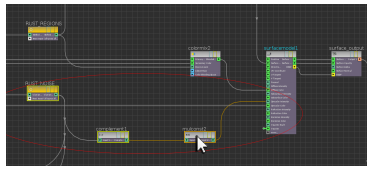
As a final step, the temporary **Lambert VOP** can be **deleted**, and the **output of the Surface Model VOP** can be reconnected to the **Surface Output VOP**.



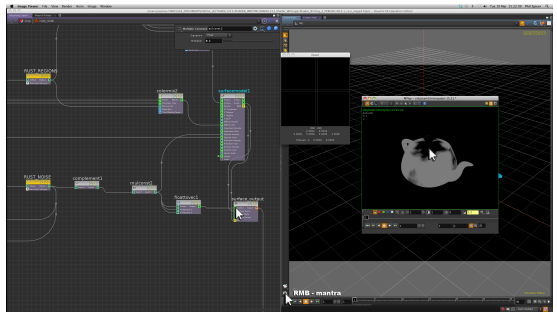
See file **H14_rust_stage5.hipnc**

FIXING THE SPECULAR HIGHLIGHT

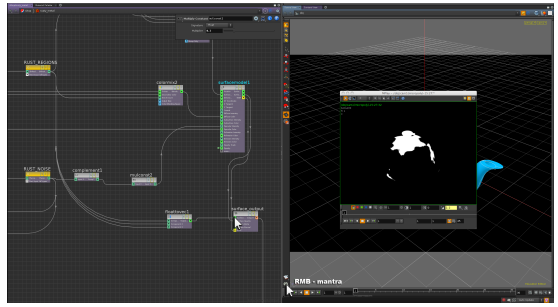
The Specular Intensity input of the Surface Model VOP is currently being driven by the RUST_NOISE Null VOP (appended with a Compliment VOP and a Multiply Constant VOP).



This **Specular Intensity value** also needs to factor the **RUST_REGIONS Null VOP** so that highlights or reflections are not returned inside any rust regions.

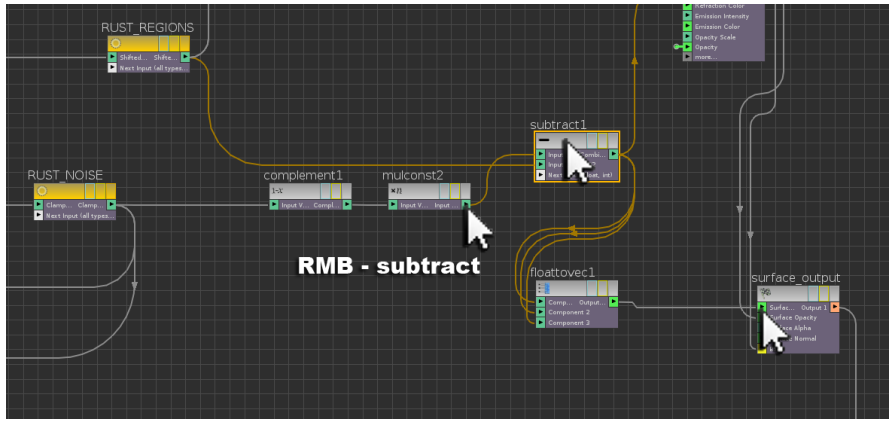


When this Specular Intensity input is fed into the Surface Output VOP and a standard mantra render is activated, the reason for the specular return becomes apparent, as the edges around black areas are not quite returning a full black as per the rust region areas. When the output of the RUST_REGIONS is examined through the Surface Output VOP, a clear B/W mask of the rust regions is apparent.

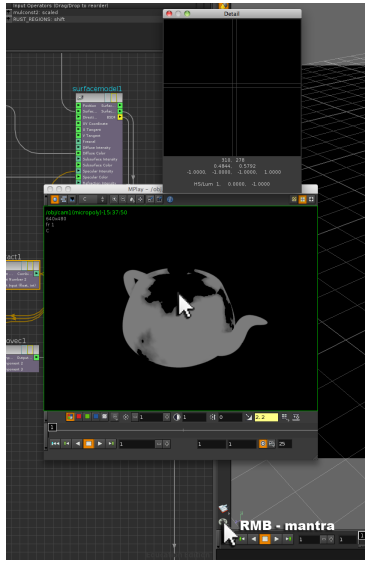


This RUST REGIONS output can therefore be subtracted from the current Specular Intensity network in order to fix the specular return so that it only occurs in the non-rusty regions of the teapot.

RMB on the output of the **Multiply Constant VOP** of the **specular intensity network** and **insert** a **Subtract VOP**; wiring the **output** of the **RUST_REGIONS Null VOP** as the **second input**.

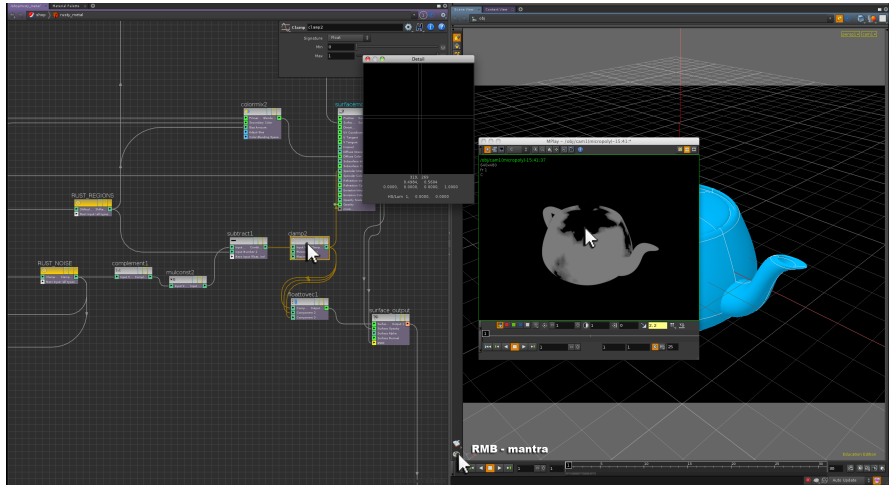


When a default Mantra render is activated, a modified specular intensity image is revealed.

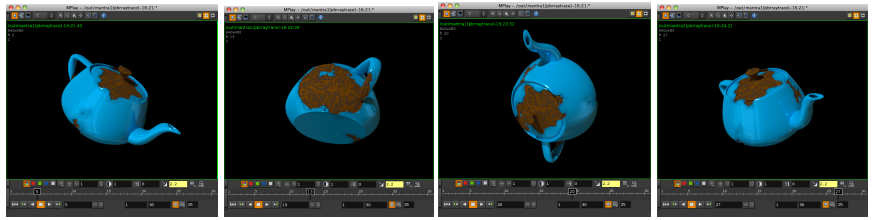


When the render is checked with a Details View (m with the mouse over mplay), the black regions are now returning negative values as a result of the subtraction. This can be corrected by inserting a Clamp VOP after the Subtract VOP.

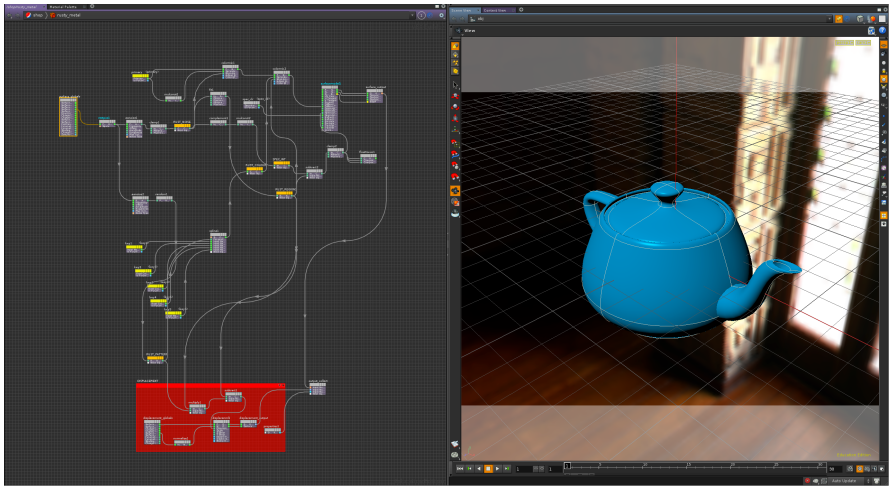
RMB on the **output** of the **Subtract VOP** to create a **Clamp VOP**. The **output** of the Clamp VOP can be fed both into the **Surface Output VOP** (for testing) and into the **Specular Intensity input** of the **Surface Model VOP**. When a **default Mantra render** is created, and **checked** with a **detail view**; all of the black regions are returning a flat value of zero.



With the specular intensity network fixed, the output of the Surface Model VOP can be fed into the Surface Output VOP and a formal render of the sequence can be generated as the specular detail errors observed have now been fixed.



The shader network can also be tidied up and annotated using Net Boxes (CTRL + n with nodes selected), and specific end user parameters for the shader can be generated and tested.

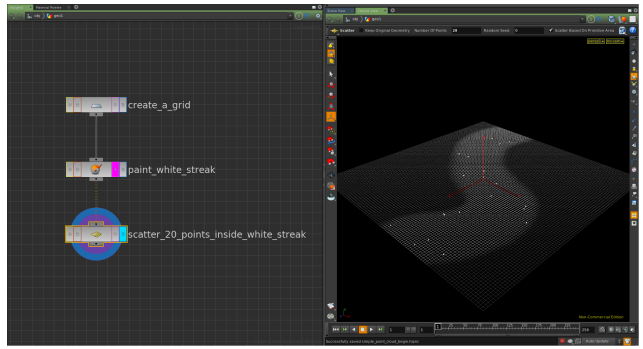


With the initial engineering of the shader complete, it can now be tested relative to production needs, and if necessary parts of the shader can be modified to a higher-level of finish.

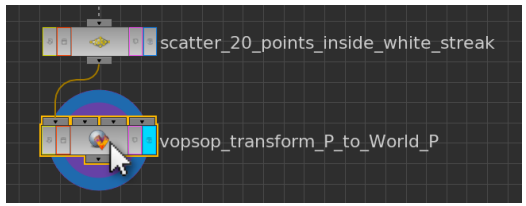
See file H14_rust_end.hipnc

SIMPLE POINT CLOUDS

Point Clouds are **scattered points** over geometry that can be referenced to **create shading tricks**. Point Clouds can be invoked either in a **Material shading context** or within a Geometry Level **VOP SOP** or **Volume VOP SOP context**. **Open the scene simple_point_cloud_begin.hipnc**.



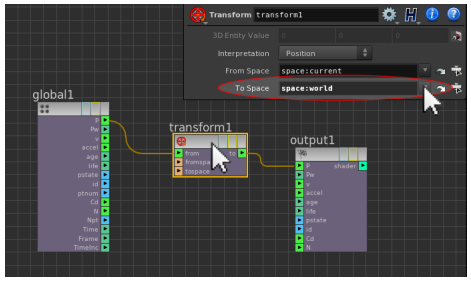
This scene contains **20 points scattered over a grid**. The **placement** of the **scatter** is **controlled** by the **Cd** (colour) attribute created by a **Paint SOP**. This example will look at the basic mechanism of a Point Cloud setup.



Append a **Point VOP SOP** to the Scatter SOP.

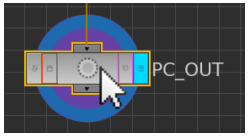
Inside the **VOP SOP**, create a **Transform VOP wiring it between** the **Global P Input** and **Output**. In the **parameters** for the **Transform VOP** specify using the drop down menu:

To Space **space:world**

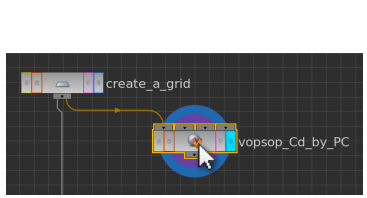


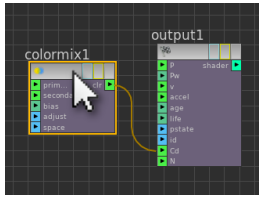
In order for Point Cloud data to be calculated correctly in whichever context is opted for, its **position data** should be **transformed** into global **World Space** rather than the local Object Space inside each object. This will help **ensure** that any Point Cloud **calculations work** as expected.

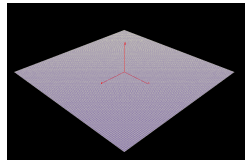
Append a **Null SOP** to the **VOP SOP** and **rename** it to **PC_OUT**. This node can either be **saved to disk** using a **ROP Output Driver**, or **referenced live** into the Point Cloud VOP network using an **expression**.



MMB append a second **Point VOP SOP** to the **Grid SOP**. This node can be used to add custom colour to the grid based upon the scattered points.

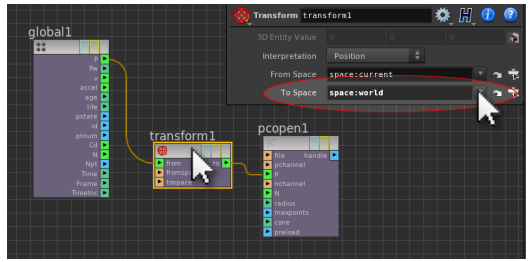






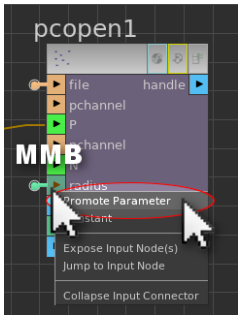
Inside the **VOP SOP**, create a **Colour Mix VOP** and **wire** its **output** into the **Cd input** of the **Output VOP**. This will colour the grid lilac (a 50% mix between the default pink and the purple primary and secondary colours specified by the Colour Mix VOP).

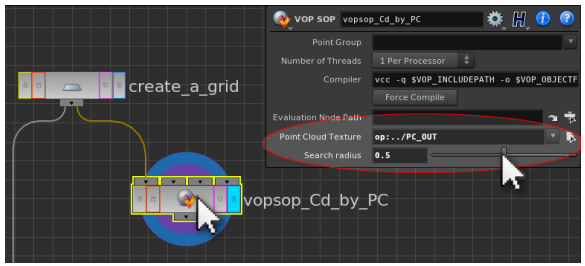
As a **new node** create a **Point Cloud Open VOP**. Wire its **P input** to the **Global VOP P output**.



In-between this wiring **insert** a **Transform VOP**, once again activating **World Space**. This will convert the current point of the grid into World Space (matching the scattered points).

On the **Point Cloud Open VOP**, **MMB** on **both** the **file** and **radius inputs** to **Promote** their **Parameters** as end user controls. At VOP SOP level, these promoted parameters can be seen.





In the promoted **VOP SOP parameters** specify:

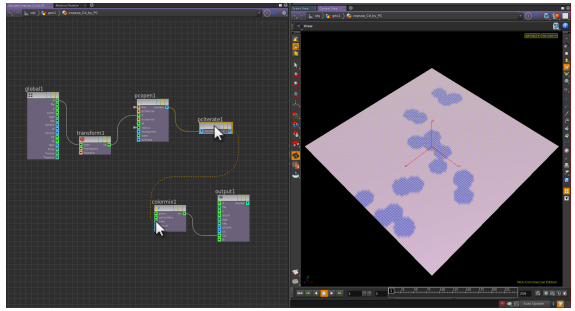
Point Cloud Texture	**op:../PC_OUT**
Search Radius	**0.5**

This will create a **live link between** the **Point Cloud Open VOP** and the **PC_OUT Null VOP** and increase the search radius around each scattered point.

NOTE: Alternatively the Point Cloud Texture parameter could read in point cloud geometry saved to disk.

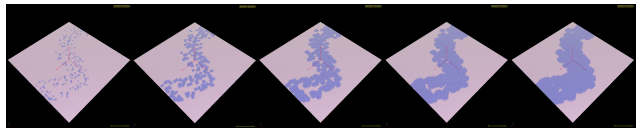
Back **inside** the **VOP SOP**, **append** a **Point Cloud Iterate VOP** to the **output** of the **Point Cloud Open VOP**. This will **evaluate each scattered point in turn**, **returning** a **value of 1** if a scattered point is found.

When the **output** of the **Point Cloud Iterate VOP** is **wired** into the **bias input** of the **Color Mix VOP**, the grid is coloured by the scattered points.



NOTE: A **warning error** is produced by the **VOP SOP** citing that the **Point Cloud Iterate loop** is never completed. In this simple context, this warning does not matter as all the scattered points can be visually verified as found.

The **colour radius** of the **scattered points** can further be adjusted by modifying the **Search Radius parameter** of the **VOP SOP**. Similarly, the **number of points scattered** on the grid can also be **increased**. When these parameters are **animated**, interesting **colour based growth effects** can be achieved. Such colour growth effects can then be used to **drive other effects**.



See file simple_point_cloud_end.hipnc