AE500013

Exploring Stylized Looks with Arnold Toon Shader, 3ds Max, and OSL

Ciro Cardoso
Hayes Davidson

Learning Objectives

- Develop different NPR styles with the Arnold Toon Shader
- Create hatching and linework styles with OSL nodes
- Implement Arnold Operators to apply different styles
- Develop and integrate traditional illustration styles into your workflow

Description

Arnold is a powerful production renderer used across the industry for realistic imagery. However, you can use Arnold for much more than photorealistic rendering. The film Spider-Man: Into the Spider-Verse is a good example of Arnold software's versatility. NPR (non-photorealistic rendering) can be a solution for fast-turnaround visualization projects, such as initial briefs or competitions. In this class, we will explore how to use traditional 2D illustration techniques in 3ds Max software with the powerful and versatile Arnold Toon Shader, with the help of some OSL nodes. To streamline the process, you will learn how to use Arnold Operators to create templates and efficiently apply different illustration styles that are perfect for fast-paced projects. From adjusting linework quality to hatching styles, we will cover tips and tricks to start implementing NPR and stylized looks into your workflow.

Speaker(s)

Ciro Cardoso is an author and a self-taught 3D artist with over 16 years of production experience in the animation and visualization industry. His skillset encompasses a wide range of productions and DCCs. When he's not working as a Senior Artist at Hayes Davidson pushing R&D, Ciro can be found exploring new techniques on lighting, rendering and looks to push realism in CGI.

If you have any questions related to this topic, feel free to ping me on my LinkedIn profile:

https://www.linkedin.com/in/cirocardoso/

Or on my social media accounts: @Cyrus3v

A quick intro to NPR

NPR stands for Non-Photorealistic Rendering, and this nomenclature refers to CGI that follows a traditional style (cell shading, hand drawing, 2D illustration) in contrast with a photorealistic 3D rendering. Some perfect examples of this can be seen in recent movies, like Spider-Man: Into Spider-Verse (2018), The Mitchells vs the Machines or the Star Wars Visions series. The movie Spider-Man: Into Spider-Verse is also a great example of traditional 2D comic drawing techniques being applied into a 3D scene, such as halftone and shading techniques.

The studios behind these movies have complex workflows and shaders to produce such results. However, getting similar styles is possible due to the powerful Arnold Toon Shader. Even if you aren't trying to create the next Spider-Man movie, you can apply these techniques for scientific, engineering and architectural illustrations.

This handout isn't a replacement for the **Arnold Toon Shader** documentation, which explores all the settings available. Instead, it works as a checklist and a summarized version of some of the most practical tools and techniques.

You can find the Arnold Documentation for the Toon shader here:
https://docs.arnoldrenderer.com/display/A5AF3DSUG/Toon

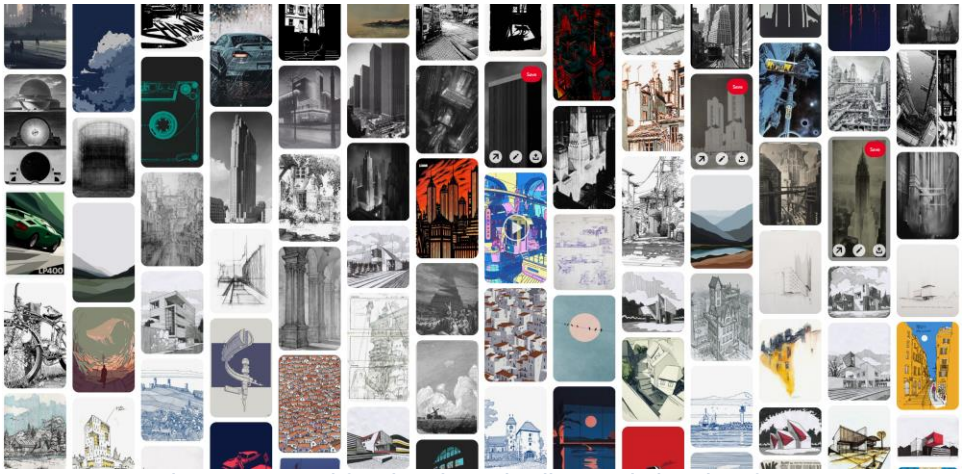
And you can find a series of tutorials with the Toon shader here:

https://docs.arnoldrenderer.com/display/A5AF3DSUG/Toon+Tutorials

What do you need to start exploring the **Arnold Toon Shader** and develop your stylized looks?

References, References, References

Before we move any further and see some of the technical aspects of working with this shader, getting references is essential. These references are a crucial aspect for you to develop your style. Those references will help you identify elements you like and things you want to implement and work as a touching stone along with the project. An excellent way to gather those references is by creating a Pinterest board, similar to what I have here:



My Pinterest Board for drawing, shading and sketch references

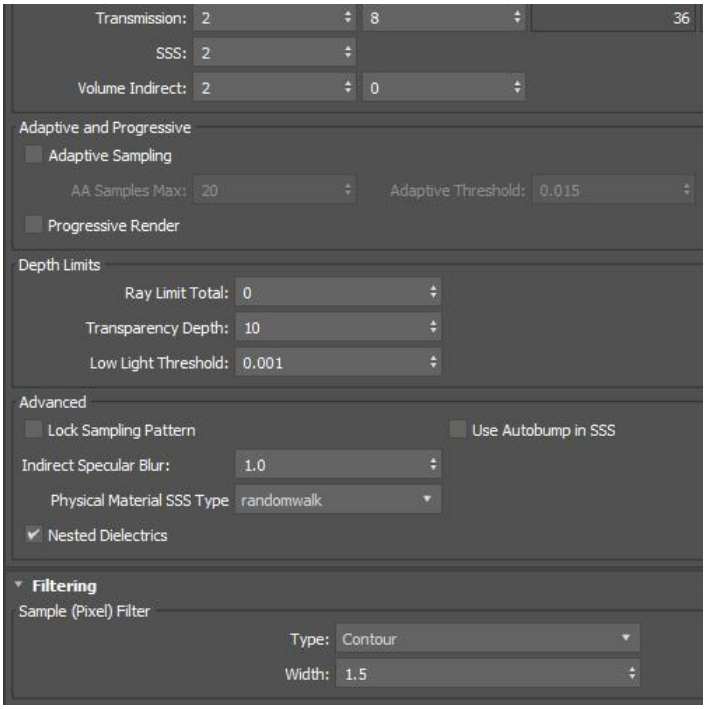
This particular board is 200 plus references, keeps growing each day and is also a great way to find new cool ideas and keep inspired, as Pinterest will keep finding further references for you.

First steps

Let's start with the essential things you need to work with the **Arnold Toon Shader**.

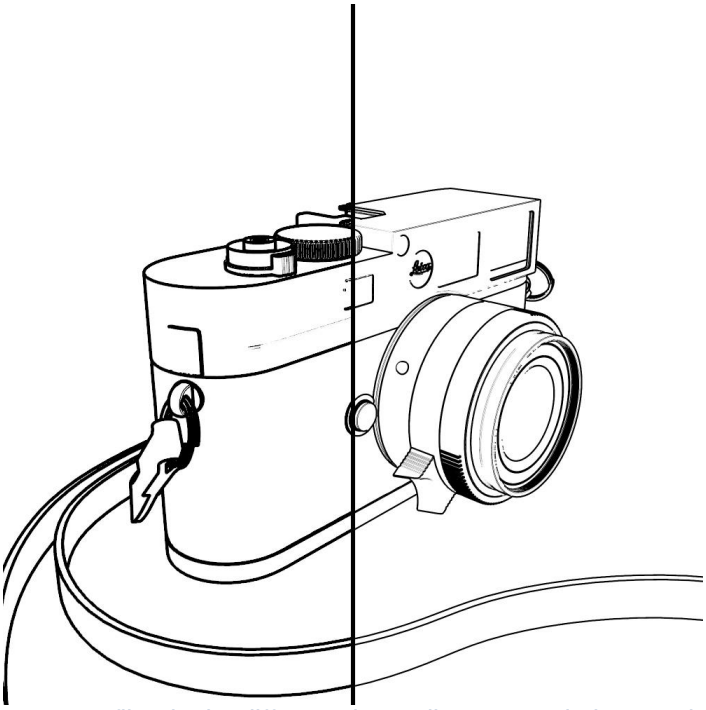
Contour Filter

First, you need to switch the **Filter** from **Gaussian** (Default) to **Contour**. To do that, open the **Render Setup** window (F10), go to the **Arnold Renderer** tab, scroll down until you reach the **Filtering** section.



Contour Filter – essential for the Toon Shader edges

You probably noticed that my **Filter Width** is set to **1.5**. The lowest value you can use is **1**, but you may need to increase this value depending on the project. However, keep in mind that the higher the resolution you use to render, the higher the width's value will have to be. As an example, if you use a **Width** of **5**, the edge's thickness is going to look different if I render a 1K image or if I render a 4K image, and here is an example showing that:



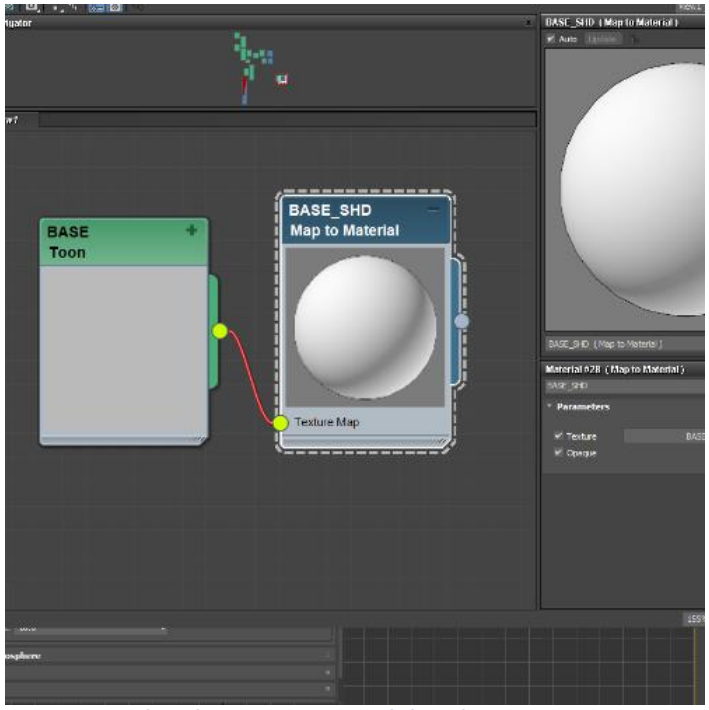
Contour filter looks different depending on resolution used

On the example above, I have the same **Filter Width**, in this case, **5**, but I am rendering two different resolutions. The image is divided in half, and on the left side, the render is *1000x1000 pixels*, and the render on the right side is *2000x2000 pixels*. As you can probably see, the edges on the left side (the 1000x1000 pixels render) are thicker than the edges on the right side (the 2000x2000 pixels render), but we are using the same value for the **Contour filter** for both images. While developing the toon shaders, you need to keep this in mind and try to do so using the final resolution for your project. Increasing the **Contour Filter** will proportionately increase the render time.

Find more information here: https://docs.arnoldrenderer.com/display/A5NodeRef/contour_filter

Toon\Map to Material

Next, you need the **Toon** map (Maps – Arnold – Surface) connected to a **Map to Material** (Materials – Arnold – Utility) node. The **Toon** map is what you use to create the different NPR styles, and the **Map to Material** is what you use to assign that **Toon** map to the geometry. A basic shading network will look something like the example below:



Assign the Map to Material to the geometry

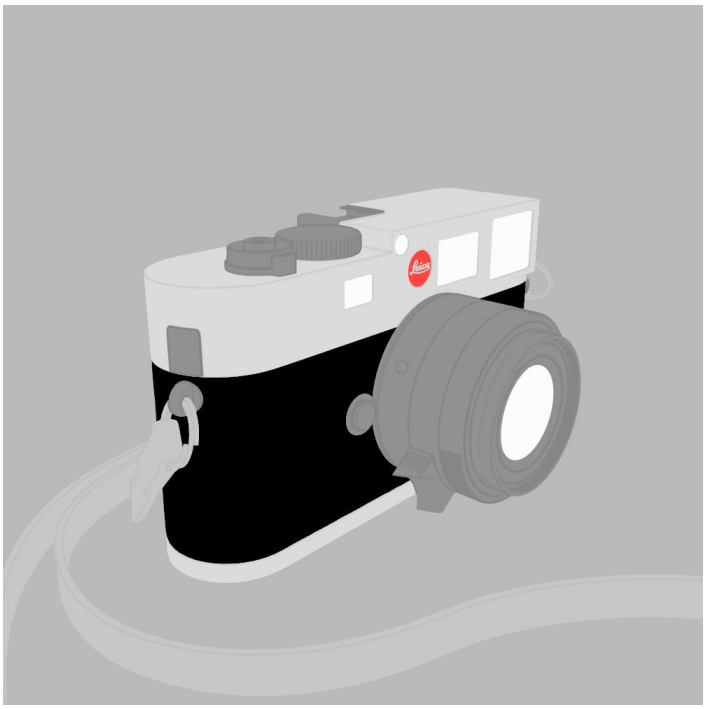
And with this, you are good to go and ready to explore, develop and create different stylized looks using the **Arnold Toon Shader**. I mentioned that you would need references for what you are trying to accomplish in terms of look and aesthetics. Something that may help you with this is by searching for examples of Notan, Chiaroscuro and Linework styles. The 30-minute talk covers in more depth these artistic styles and how you can use the **Arnold Toon Shader** to accomplish it.

Artistic Styles – Mass\Notan

This handout isn't an art class but a quick introduction to different artistic styles and what can be accomplished. **Notan**, for example, is a style that explores the use of mass. **Notan** is a Japanese word that means "light dark harmony" and bellow, you can see an excellent example of that in action on a Japanese artwork:

Old plum by Kano Sansetsu

Applying this technique involves exploring arrangements of light and dark elements and is an excellent way to explore shape and composition. One way to achieve this look with the **Arnold Toon Shader**, is by using the **Emission Color** and **Emission Weight**. Below you can see an example of me using only the **Emission** to achieve this look:



Notan style accomplished using the Emission

The process to create this style is quite simple. You don't need any lights in your scene because each material will emit light. Then for the different local values, I am using multiple **Arnold Toon Shaders** with different **Emission Color** values. The 30-minute talk covers in more depth these artistic styles and how you can use the Arnold Toon shader to accomplish it.

Artistic Styles – Form\Chiaroscuro

Another artistic style that explores the form is called **Chiaroscuro**. This technique focuses on the strong contrast you have between light and shadows and explores the use of depth and tone. For this style, you will need one strong light source to create contrast between light and shadow. You can see that technique being used in this good example painted by the master Caravaggio:

The Calling of St Matthew by Caravaggio

Usually, you have just one intense light source in a scene like the one above, but that does not mean you can only use one light. Looking closer at this painting, you will probably notice that the legs under the table are illuminated. However, in a physically correct world, these would be in complete shadow. Caravaggio made an artistic choice to ignore what would be physically accurate and instead chose to illuminate the legs to improve the composition.

We can achieve something similar with Arnold using light linking. Light Linking is an artistic process in which each light is linked or affects specific objects in your scene. You can find more information on how to do that here:
https://docs.arnoldrenderer.com/display/A5AF3DSUG/Light+Group

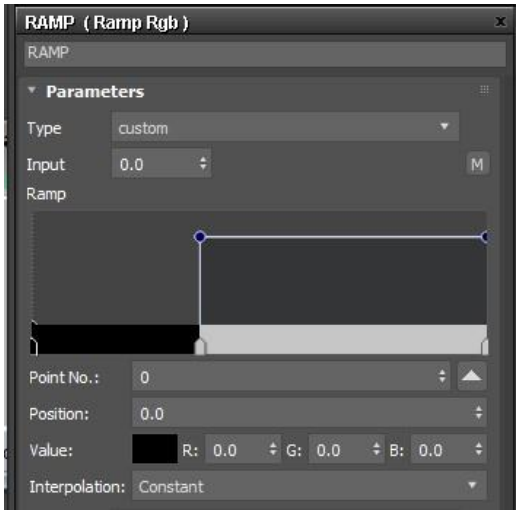
To apply a similar Chiaroscuro style in your project, you need to have a strong light to define the object's form and play with the contrast between light and shadow. Below you have a simple example of using the **Arnold Toon Shader** to render a Chiaroscuro image:

The Leica M9

Achieving something like this is easy. For lighting, I am using a strong directional light and then for shading, I am using a **Utility** node with the **Shade mode** set to **Lambert**.

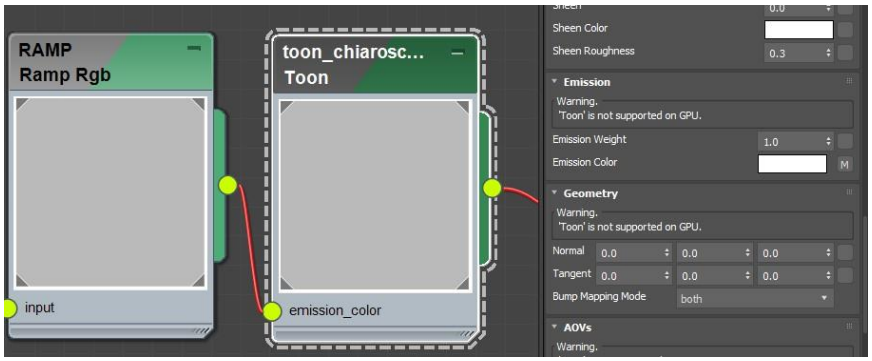
Lambert Shade Mode

I use this **Utility** node to create a mask by connecting it to an **RGB Ramp** set to constant, as you can see below:



Interpolation set to Constant to create a sharp mask

Finally, the **Ramp RGB** is connected to the **Emission Color**, with the **Emission** value set to 1.

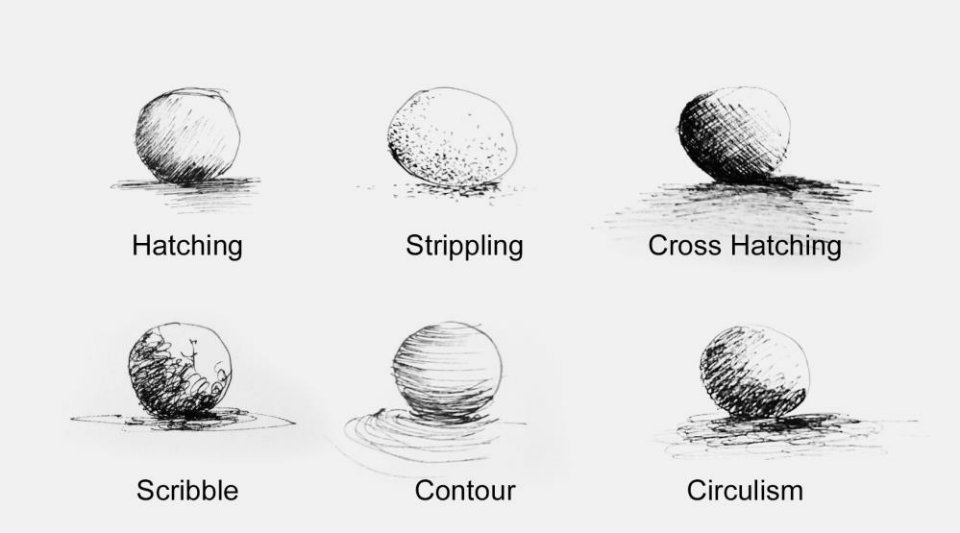


Emission Weight set to 1

If you are following these steps and not getting the same look, the trick is increasing the light's intensity. In my case, the light's **Intensity** is set to 1, the **Exposure** to 1.1 and the light's **Angle** set to **10**, so I can have some soft shadows. You will probably need to increase the light samples to **2** or **3** to have a cleaner render. The 30-minute talk covers in more depth these artistic styles and how you can use the Arnold Toon shader to accomplish it.

Artistic Styles – Line\Texture

A third style is sketching, and this is the one that many of us are more aware. Is when you use a simple pencil or pen to draw, scribble, hatch, and sketch. This technique is the simplest and the most used to express ideas. I asked my good friend Mark Eszlari to draw a couple of examples showing those shading techniques:



Some of the most common shading techniques

If you want to focus only on the linework, without worrying about local values, then set the **Emission Weight** to 1 and the **Emission Color** to white or a light gray. That way, you will be closer to a drawing on a piece of paper. You can create the shading techniques above by using bitmaps or a procedural approach using OSL Noises. We will see how to do that next, but before moving to that topic, let me share something that will help you start exploring and developing your stylized styles.

These three elements we briefly covered, **Mass**, **Form** and **Line**, are primary elements of design. You can mix these three elements, but one must be dominant, while the other two complement the design. Implementing these elements in equal parts will create a "muddy" concept that will not be clear because nothing stands out. Having one element more dominant will help you balance how you approach and develop your stylized looks. However, this isn't an art class, so I highly recommend researching these styles in greater depth and gathering references of what you would like to reproduce.

In the next couple of topics, I will cover some of the techniques that I used while developing stylized looks, and we will focus on creating shading techniques, linework quality, and the use of tonemap.

Create shading and linework styles with OSL nodes

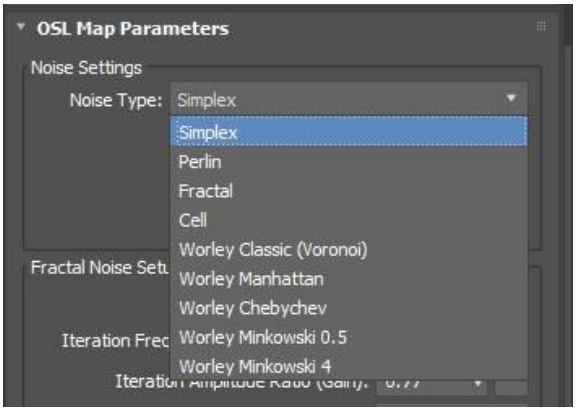
Let me introduce you to the OSL nodes that can help you create customised edges and hatching and surface details, which will be helpful later for shading.

OSL Uber Noise

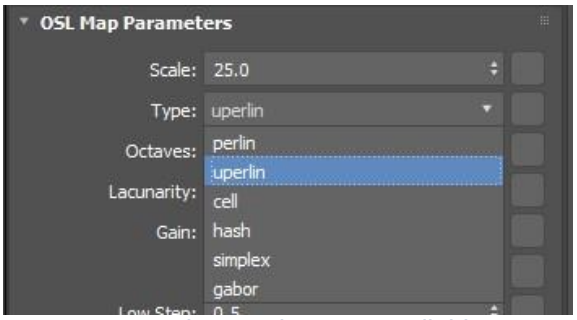
You can use the **OSL Uber Noise** or the **OSL Noise**, and both are useful with some minor differences. I prefer the **OSL Uber Noise** because you have some extra settings, like the **Layer Distortion** and the **Output Normalize**, which adds more detail to the noise.

When adjusting and playing with the settings for these noises, it is better to have a reference in mind. Do a quick search for hand drawing hatching or shading techniques, and you will find some examples of the different styles being used. I encourage you to play with the **OSL Uber Noise** node and explore each setting and what it does. While you do this, you will come across with “happy accidents”. It may not be useful for the style you are developing in that instance, but create a Material library to start saving those happy accidents you may find. But let me give you some tips.

The first thing to explore are the noise options available for both the **OSL Uber Noise** and the **OSL Noise**:

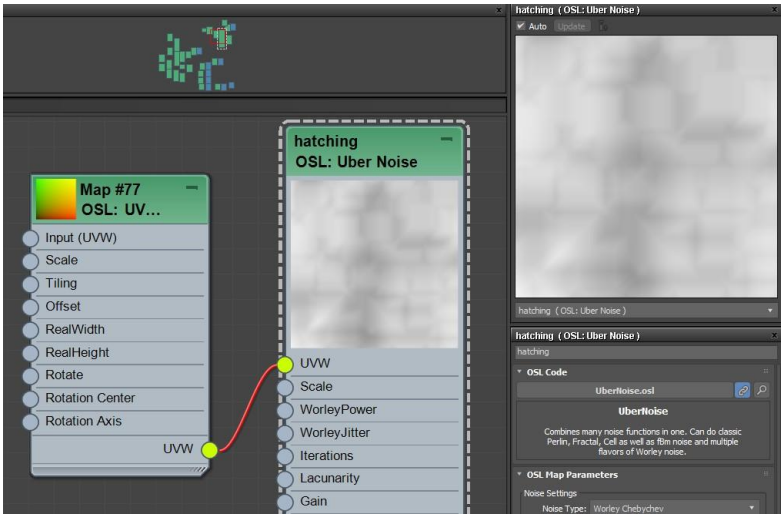


OSL Uber Noise – noise type available



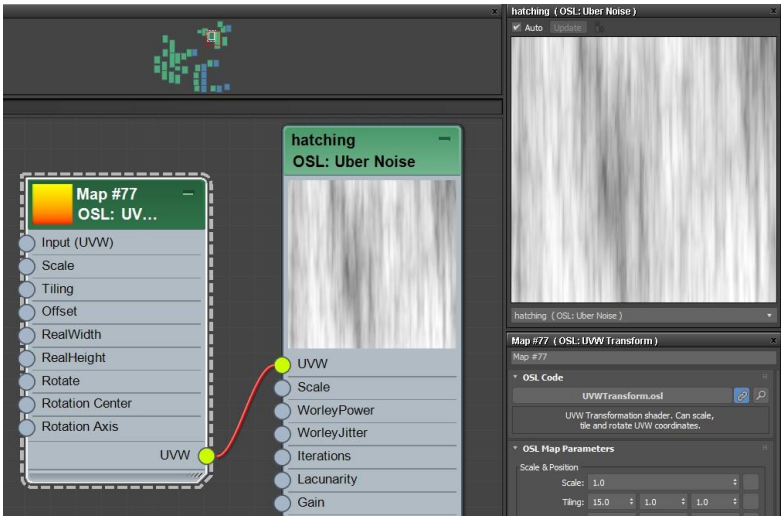
OSL Noise – noise type available

You may be asking, so now what? How do I transform these noises into something useful? Well, before you start playing with the other settings, connect an **OSL UVW Transform** to the UVW slot in the **OSL Uber Noise** so that you can add some tilling in one of the axes. That is when the magic starts, and you go from this:



Without the UVW Tiling

To something like this by increasing the tiling from 1 to 10 in the X axis:

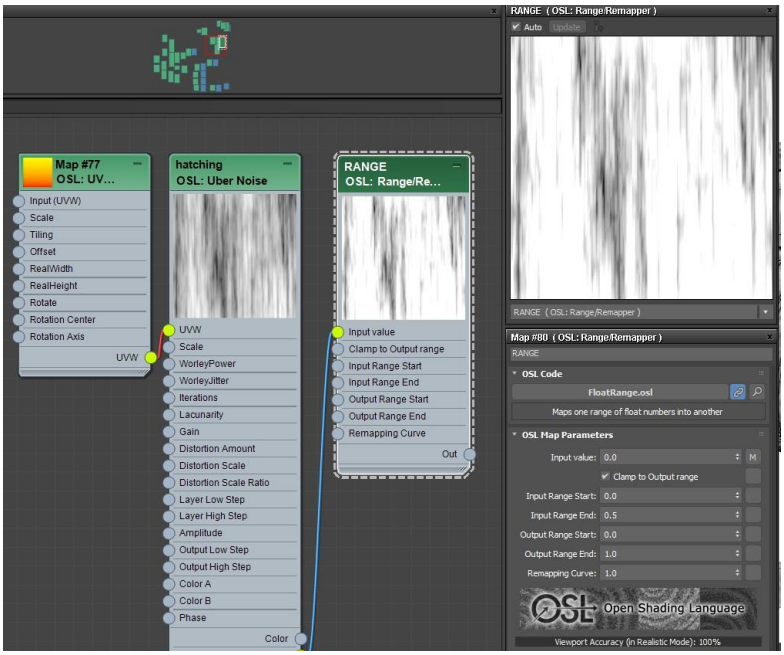


With the UVW Tiling

The **OSL Uber Noise** can create thousands of options for shading, linework and even mimic paper texture. Before we explore how to make those, let me introduce you to some helpful OSL nodes.

OSL Range\Remapper

The **OSL Range\Remapper** node is perfect for adding more contrast and remap values and create masks if necessary. Sometimes, you need to tone down how the **OSL Uber Noise** affects the **Arnold Toon Shader**, and this node is perfect for this.

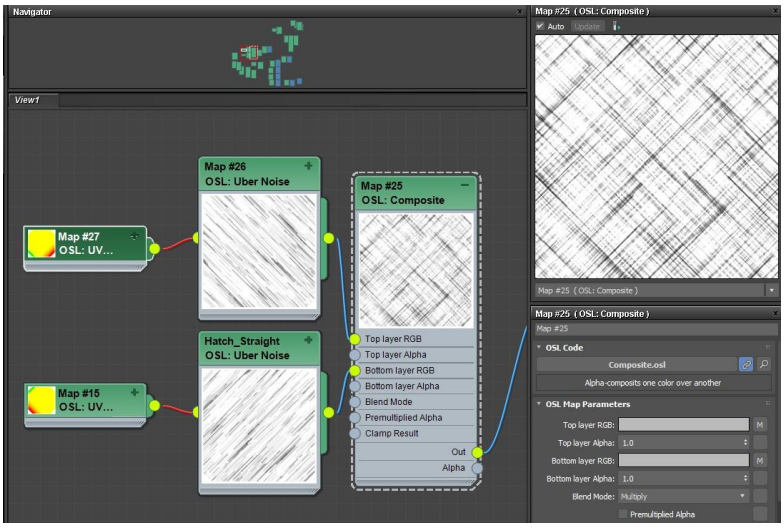


OSL Range\Remapper

Other alternatives are the **OSL Lift\Gamma\Gain** or the **OSL Tweak\Levels**. I prefer the **OSL Range\Remapper** because it already has a float output, and some nodes do require a float value instead of a color value.

OSL Composite

The **OSL Composite** node is very useful for combining different OSL Noises, helping create a hatching effect. Here is an example of that:

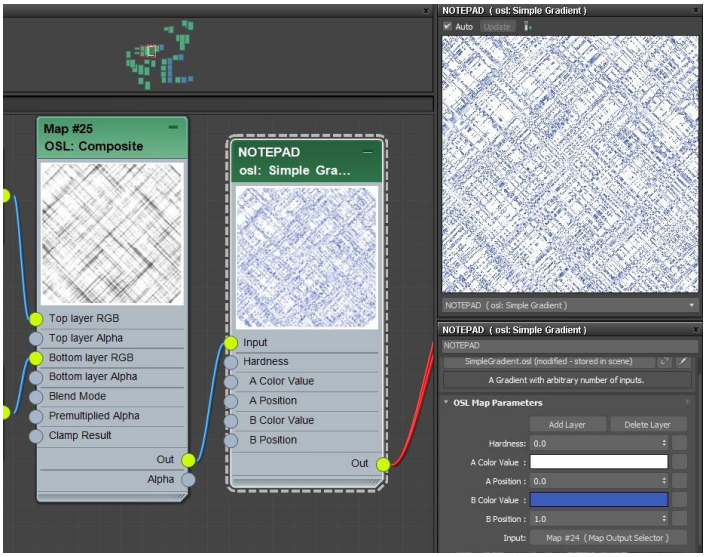


OSL Composite

In the example, above you can see how I used another OSL node, **UVW Transform**, to rotate 45 degrees both noises to create a cross-hatching effect. You have multiple blending modes to multiply, subtract and add values.

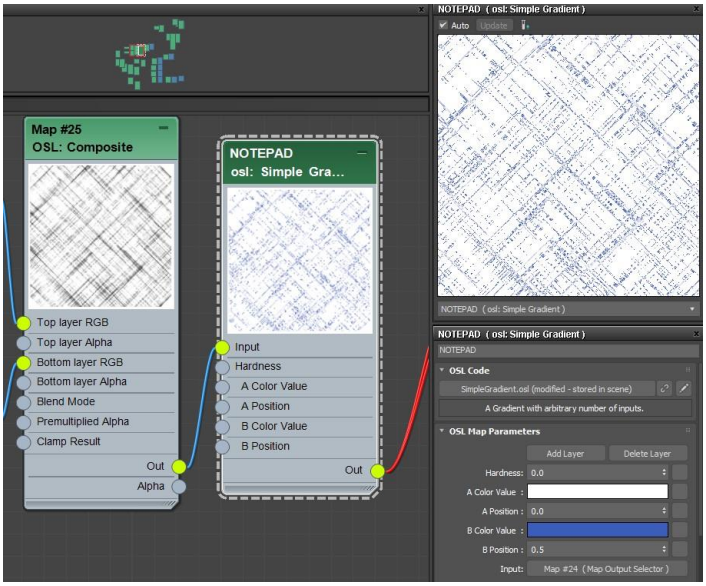
OSL Simple Gradient

The **OSL Simple Gradient** is another useful shader that you can use to color map the noise and control the hatch's intensity. How? Add an **OSL Simple Gradient** and delete the layers until you have only two. Set the second layer's value to 1. You can then use this to add colorize the hatch, like the example below:



OSL Simple Gradient with two layers

See then what happens when you change the **B Position** value from 1 to 0.5, for example.

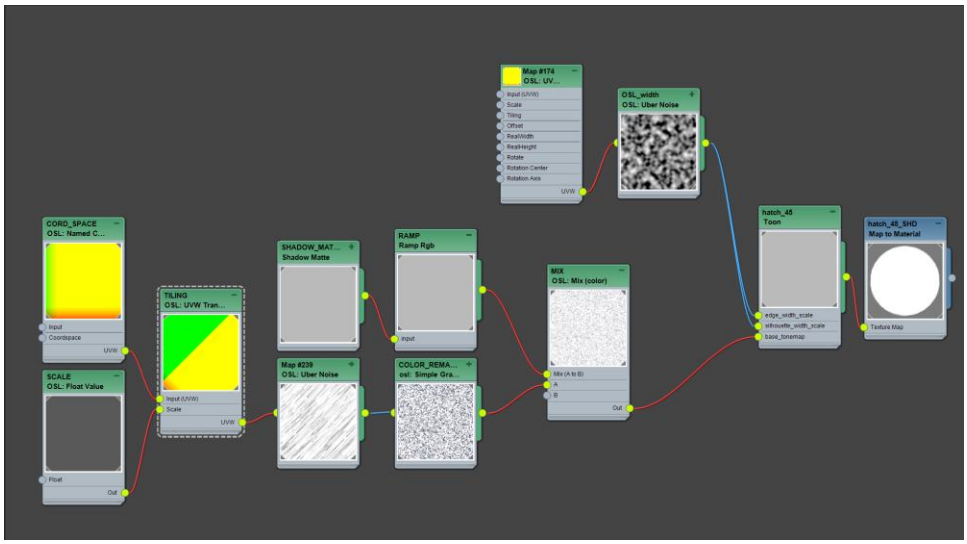


Changing the second layer's value will affect the hatching density.

Changing the **B Position** value will affect the cross-hatching density. This flexibility makes this OSL node a good way to have a second layer of control. You can have one **OSL Uber Noise** creating the hatching and using an **OSL Simple Gradient** to create different options for different assets in your scene.

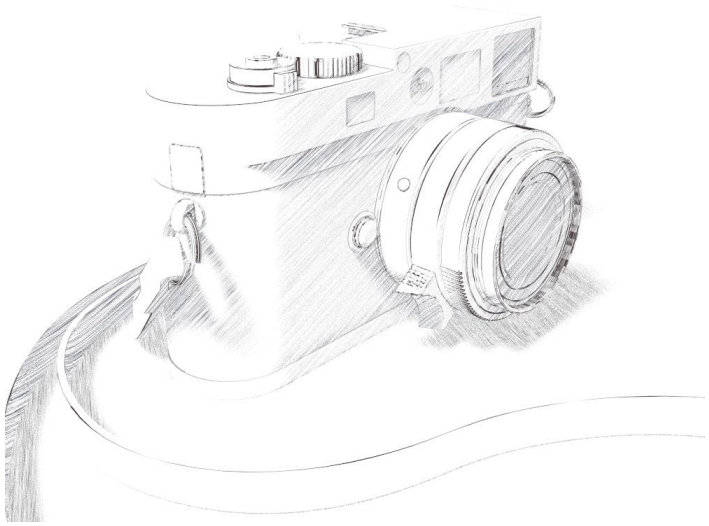
Hatching workflow

Now that we know which OSL nodes to use and why, let's see a practical example of these nodes in action. We will use OSL, **Shadow Matte** and **Ramp RGB** nodes to create a hatching style. It can look very complex, but once you understand the main concept, it is easier to use these nodes to develop other styles. Here is the shading network:



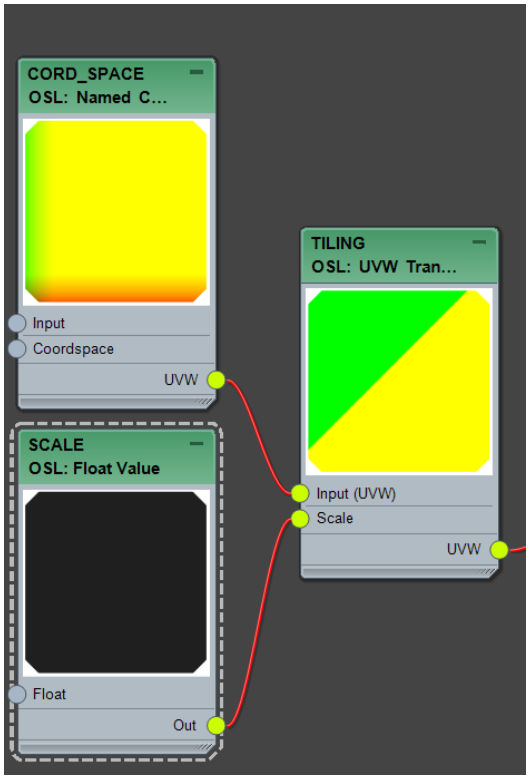
Hatching shading network

The result you get with this **Arnold Toon Shader** is this simple hatching:



Hatching shading technique

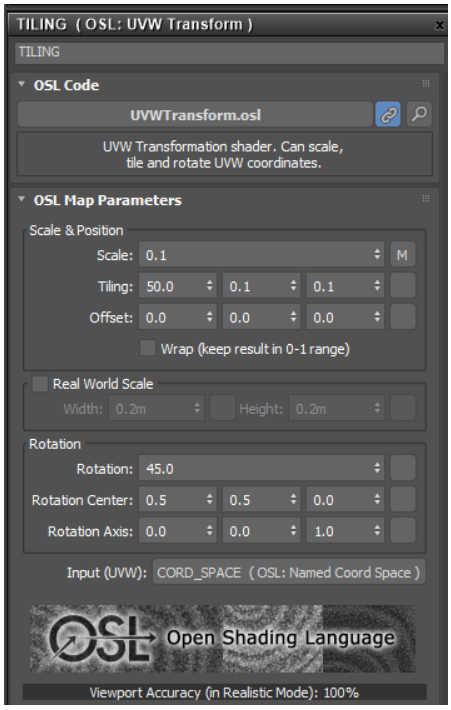
Let's analyze each node and why we are using it. Starting with this section:



OSL Named CoordSpace \ UVW Transform \ Float Value

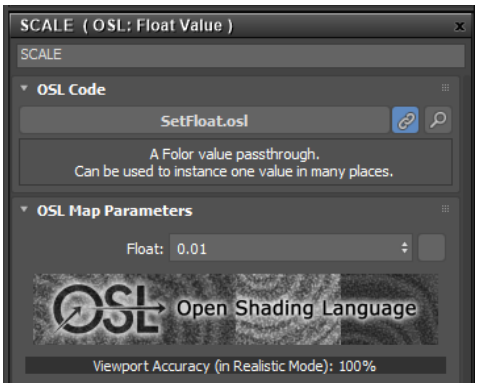
The **OSL Named Coord Space** is a valuable node for selecting which coordinate system you want to use. You have several options available, but I tend to use more are the camera or object coordinate space. When using the **Camera Coordinate Space**, the texture or noise will use the position from the camera origin point. This option will look like you are projecting the bitmap or noise from the camera. When using the **Object Coordinate Space**, the texture or noise will use the object's local coordinate. The origin will be at the object's pivot point. Using this option means when the object is moving or rotating, the value will move with object. In the example above, I am using the **Camera Coordinate Space**.

Then the **OSL Named Coord Space** is connected to the **Input (UVW)** slot on the **OSL UVW Transform.** This node is where you tile, rotate and scale the texture or noise to produce the hatching look. These are the settings I am using:



OSL UVW Transform

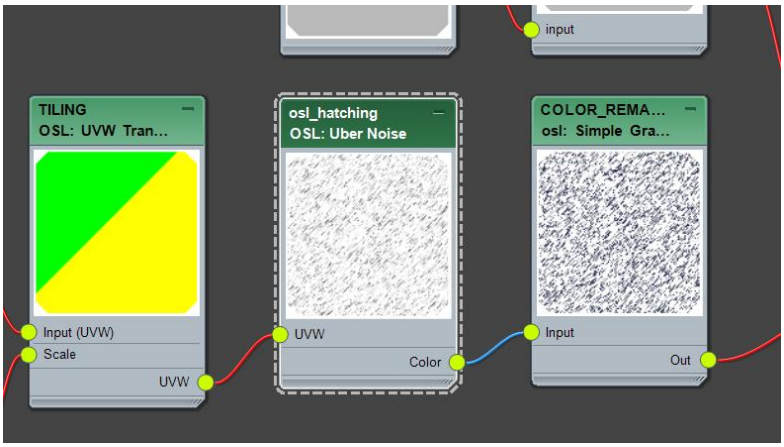
Then the last node is the **OSL Float Value**, and there is nothing special about this node, it is just a simple node to define a float value:



OSL Float Value

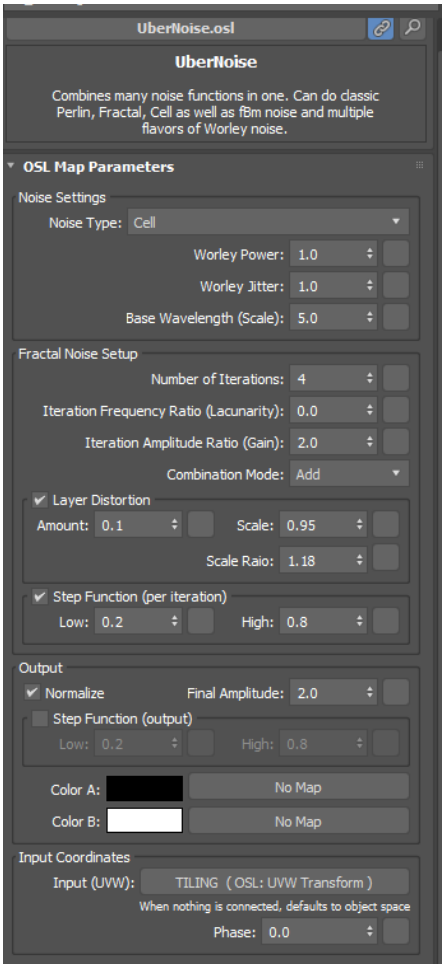
I have this OSL node connected to the **Scale** slot in the **OSL UVW Transform**. Why? Because sometimes, I need to change the scale value on multiple nodes, and instead of selecting each OSL node and manually changing the scale's value, I use one OSL node to do that for me. It avoids mistakes and is a more efficient way of working.

These three nodes are essential because they are the key to define how the texture or noise is applied and transformed. Then you connect the **OSL UVW Transform** output to the UVW slot found on the **OSL Uber noise**:



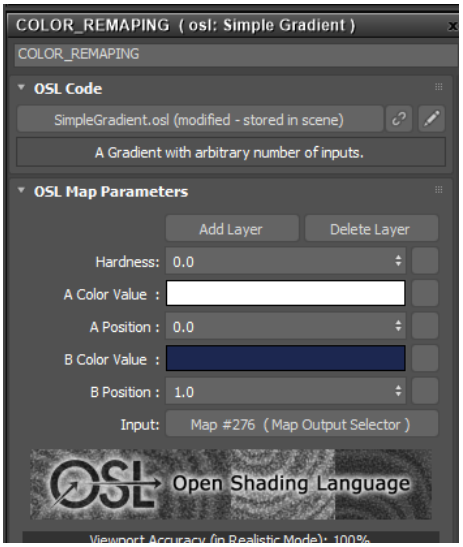
Connect the OSL UVW Transform to the OSL Uber Noise

The **OSL Uber Noise** is where the magic happens, and as I said, a lot of the styles and the incredible things you can achieve more often than not are happy accidents, like Bob Ross would say. My advice is to start the interactive Arnold Preview and play with the **OSL Uber Noise** settings. By doing this, you will begin to understand what each setting does and at the same time, you will come across with interesting results that can be saved on a material library. That is my workflow. Below you can see the settings that I used to create that hatching shading look:



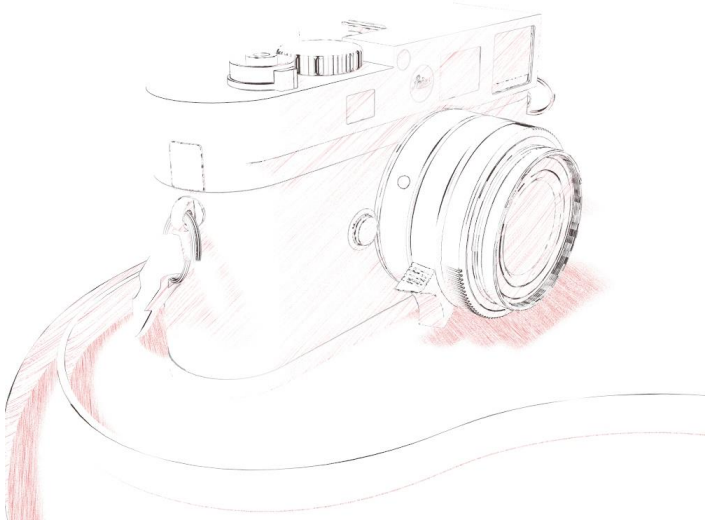
OSL Uber Noise - Hatching

The **OSL Simple Gradient** is being used here to adjust the hatching strength and at the same time color mapping if necessary:



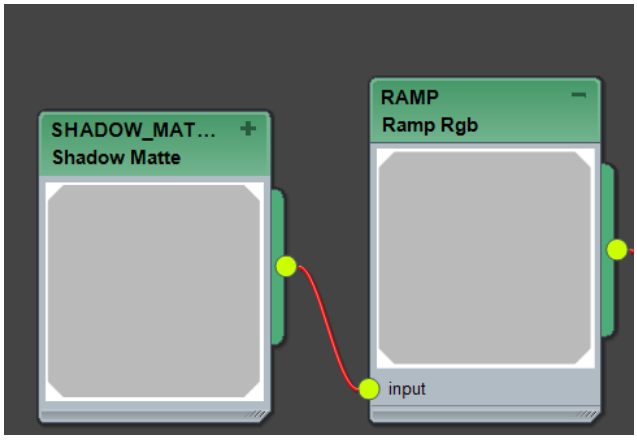
OSL Simple Gradient

The process is quite simple. Delete all the layers until you only have 2. Then **A Position** is set to **0.0** and **B Position** to **1.0**. Do you want to make the hatching less strong? Change the **B Position** to something like **0.5** and try to change the **B Color Value** to a red color:


Hatching adjusted with the OSL Simple Gradient

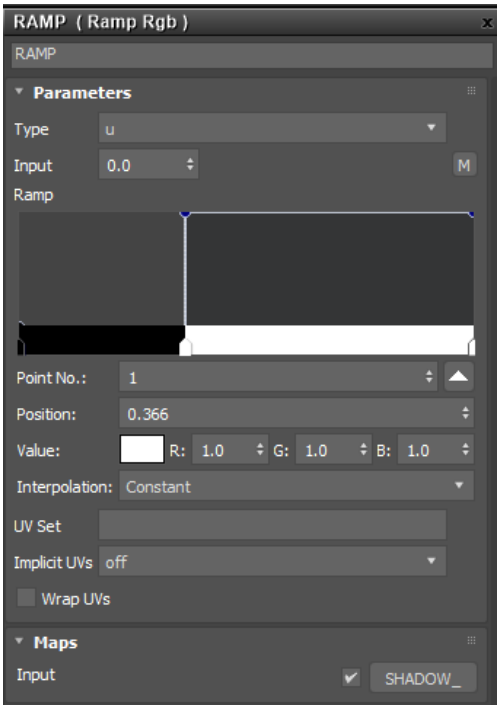
Now, the hatching isn't so strong and is also red. This is an excellent way to fine-tune the hatching produced with the **OSL Uber Noise**. Instead of creating multiple versions of the same **OSL Uber Noise**, with an **OSL Simple Gradient**, you can have numerous variations controlled

with a single **OSL Uber Noise**. The next section is where you control where the hatching is applied:



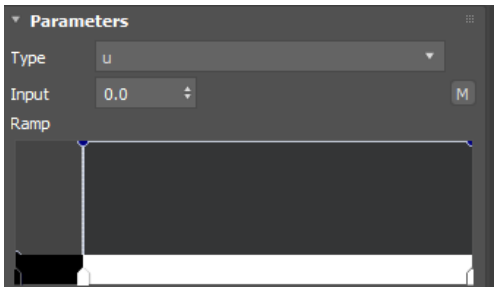
Shadow Matte node

In the "**Shading Techniques – How to**" section, I explain how you can use this node, but the main concept is using a single light in your scene to control where the hatching shading is being applied. The **Shadow Matte** can be used to create a black and white mask for the shadow and lit areas. This is then connected to a **Ramp RGB** to control how big that mask is:



Ramp RGB

Moving the point in the middle allows you to control the extent of the mask used to control where the hatching is applied. Let me show you some examples of how this works. Using this **Ramp RGB**:

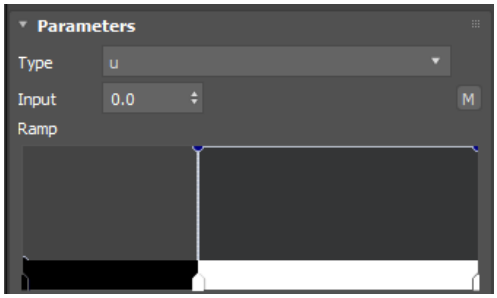


Ramp RGB – Option 1

You create a mask like this:

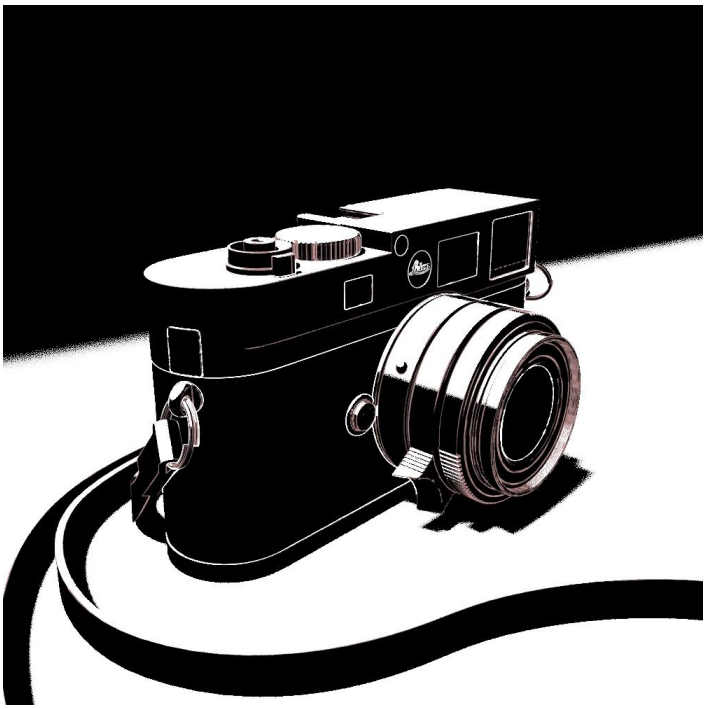
How the mask looks

If you move the middle point higher like this:



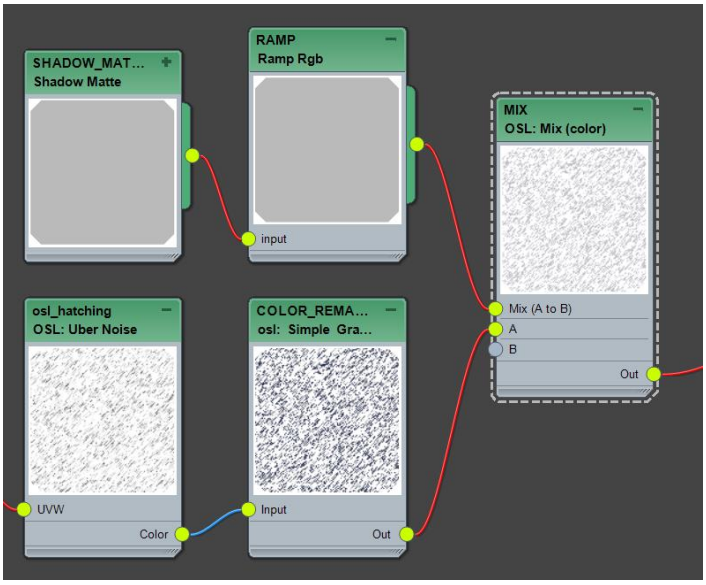
Ramp RGB – Option 2

You create a mask like this:



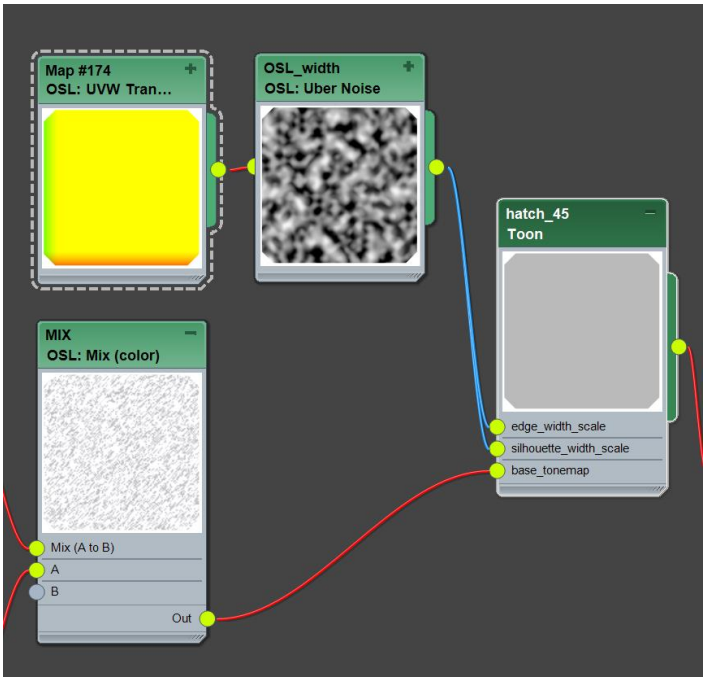
How the mask looks

We use the **Shadow Matte** connected to a **Ramp RGB** node to create and adjust the mask that is connected to an **OSL Mix (color)**:



OSL Mix (color) used to blend two colors

The **OSL Mix (color)** is used to blend two colors or even two different shading techniques. **Mix (A to B)** is where you connect the mask, which in this case is the output from the **Ramp RGB**. The **A** slot is where we connect the hatching, we created with the **OSL Uber Noise** and the **B** slot. In this case, is set to use a white color, but you can also connect a texture or a noise. The rest of the shader is quite simple:

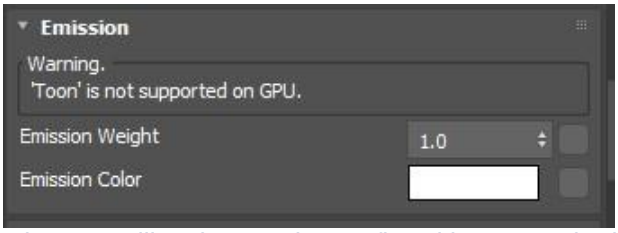


Using another OSL Noise to control the width scale

The final stage is connecting the output from the **OSL Mix (color)** to the base_tonemap input in the **Arnold Toon Shader**. You probably noticed that I also have an **OSL Uber Noise** connected to the **edge_width_scale** and the **silhouette_width_scale**, and I am doing this to control how the edges look. We are going to see how to do that in the next section.

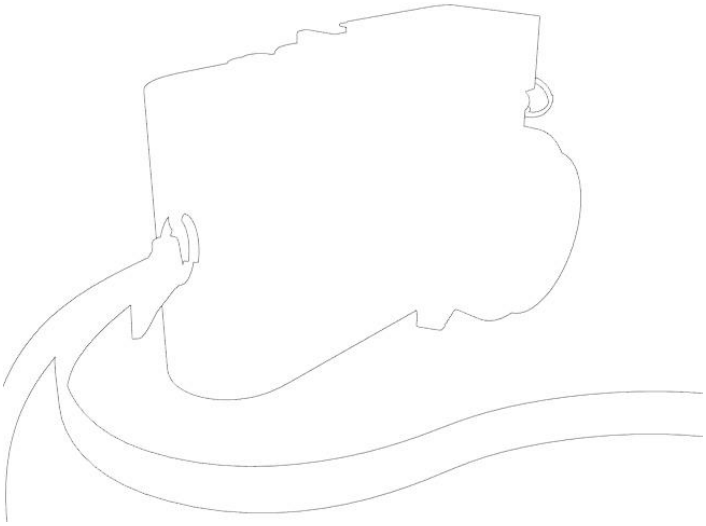
Linework

Lines are the simplest way to express an idea, and they are perfect for diagrams and schematics. Using this technique leads you to focus more on the line quality and surface texture. To help us start exploring how we can create these lines with the **Arnold Toon Shader**, let's begin by playing with the **Emission** option. Go to the **Emission** tab and increase the **Emission Weight** from 0 to 1.



Emission at 1 will make your image flat without any shadows

If you are using the default values for everything else, your render should look something like this.

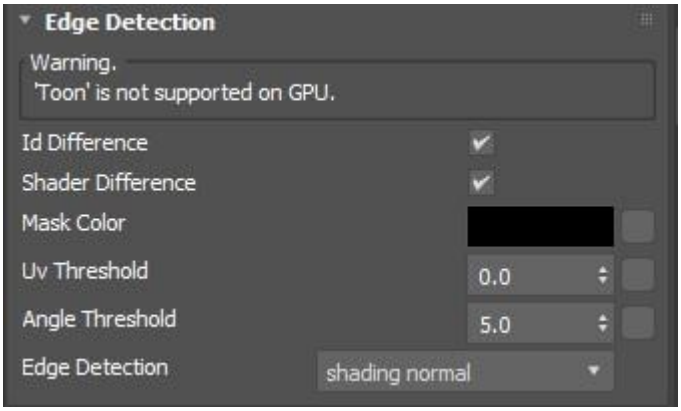


Using the default values, you would probably see just a silhouette

In this case, my Leica camera 3D model is just a single geometry, and I only have one material assign to it. With the default values, we will get a silhouette. If we want to introduce more detail, we can tweak the settings in the following section.

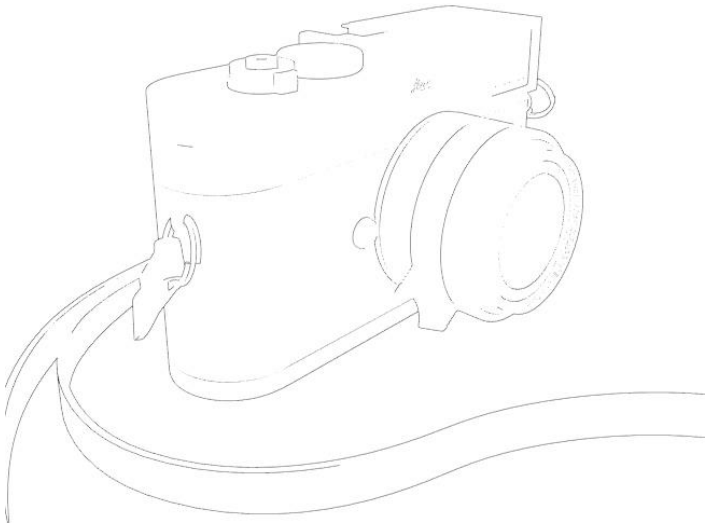
Edge Detection – Angle Threshold

To add more detail, we need to jump to the **Edge Detection** section. Here you will want to play with the **Angle Threshold**.

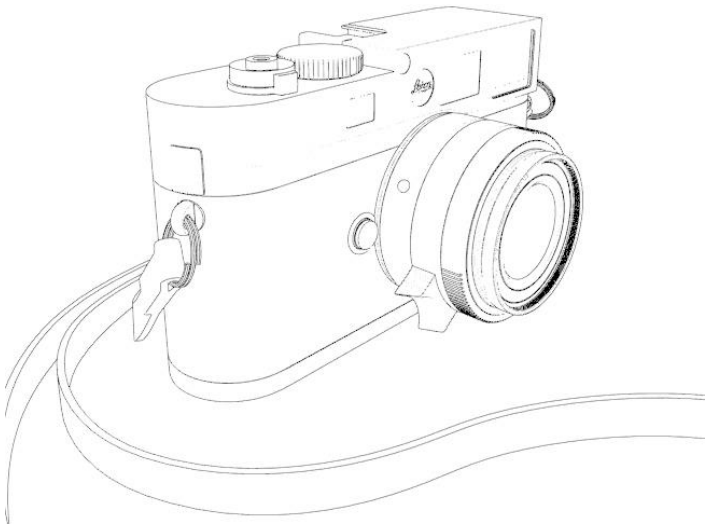


Adjust the Angle Threshold to start seeing more edges on your model

The default value is 180, so you may want to go to values between 90 and 10 or even lower.



Angle Threshold 90



Angle Threshold 30

But the lines you get with the **Arnold Toon Shader** are much more than a basic black line around the geometry. We will explore more options, but I recommend first checking the documentation to see what each option does as we will not cover every single one. The goal of this handout isn't to substitute Arnold's documentation but instead pointing to some techniques and tips to help you develop your style.

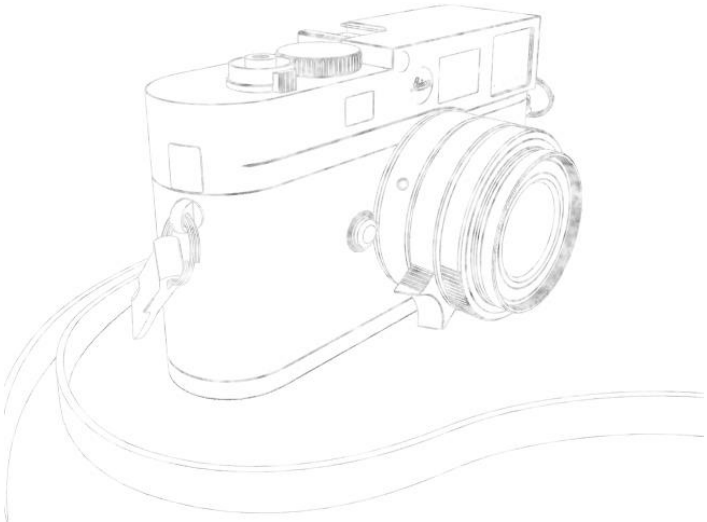
You can find more information about the **Edge** section here:
https://docs.arnoldrenderer.com/display/A5AF3DSUG/Edge

Edge

There are many options to adjust the edges, but they are still limited in achieving particular looks. However, you can still add a lot of detail to the edges by using the **Edge Opacity** and the **Edge Width Scale** options. Let's see how!

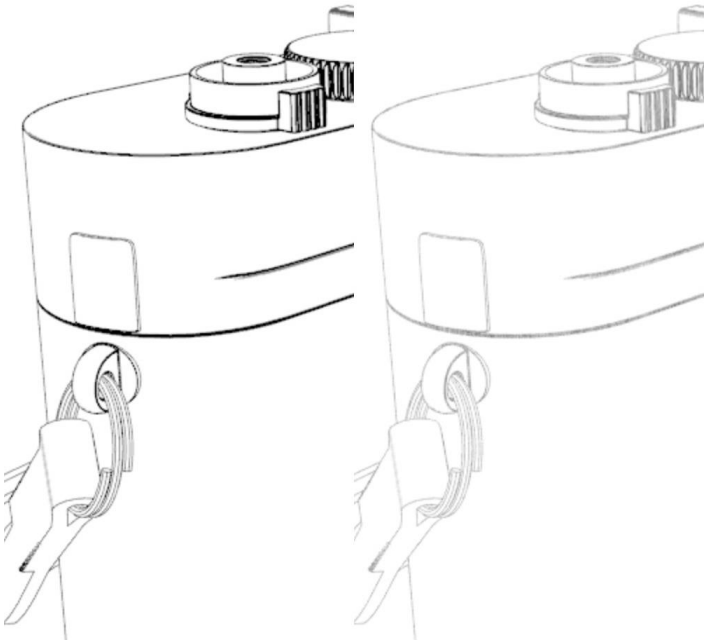
Edge Opacity

One of the things that you want to adjust and tweak is the Edge Opacity. You can do a lot with this option, mainly when connecting a texture, node, or OSL noise. Here are a couple of examples of what you can do.



OSL noise connected to the Edge Opacity

For the example above, I am using a simple **OSL Uber Noise** connected to the opacity slot. Here is another example, with both previews side by side, so that you can compare and how the **OSL Uber Noise** can give a more organic look to the edges.

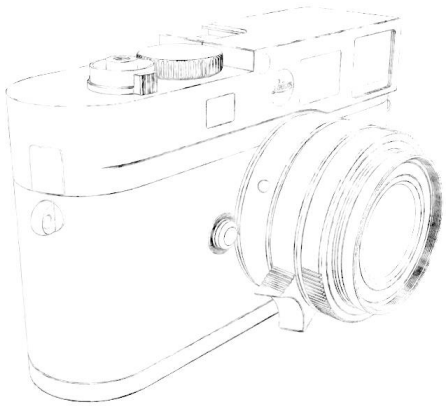


Edge comparison between different opacities.

This a good way to remove the CGI look that you get with those straight lines. And on that instance, another option that you can use to bring a natural look to those edges is by using **Edge Width Scale**.

Edge Width Scale

As the name suggests, adjusting this setting will make the lines thinner. And connecting an OSL noise or a texture will help these lines to look more organic. Most of the time, I have different OSL noises or textures connected to both the **Edge Opacity** and **Edge Width Scale**. Here is an example:



Using the Edge Width Scale, helps achieving a more organic feel

The only downside, in my opinion, is that when you adjust the **Edge Width Scale** to a value below **0.5**, the edge's color starts to become less saturated, which may not be the effect that you want. Using bigger values for the **Contour Filter** will help accentuate this effect, as the contrast will be more significant.

Again, I would highly recommend checking Arnold's documentation regarding the **Edge** and the other settings available.

You can find it here: https://docs.arnoldrenderer.com/display/A5AF3DSUG/Edge

I didn't touch on the **Silhouette** because the same principles we saw so far apply to this option.

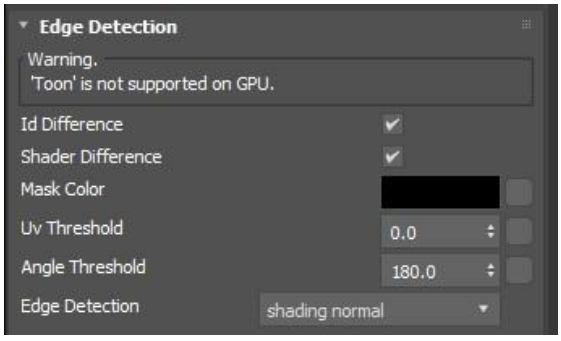
Now that we know how to work with edges and create hatching with OSL nodes let's explore how to recreate shading techniques.

Shading techniques – how to

Let's start with the basics first, and then we will jump to more complex topics. First, let's see how you can apply those beautiful hatches you created with the help of an **OSL Uber Noise** to your model.

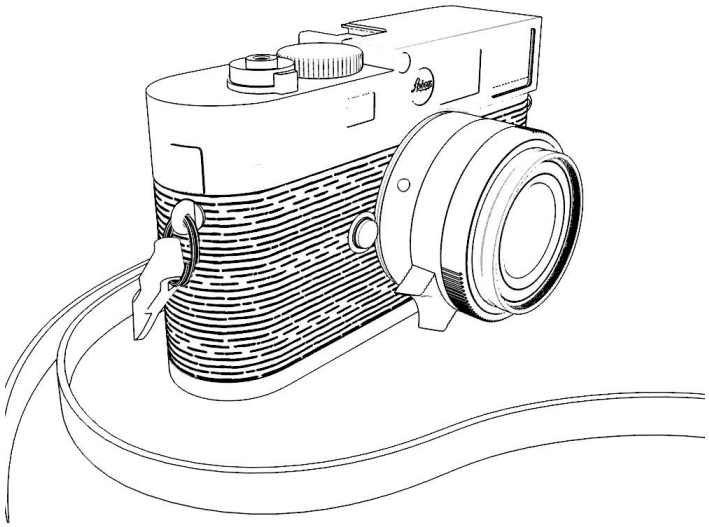
Mask Color

You can find this option in the **Edge Detection** section, and when you connect a map or a node to this option, the **Angle Threshold** is disabled. The edges are now controlled by what is connected to the **Mask Color**.



Plug one of the OSL Noises to the Mask Color

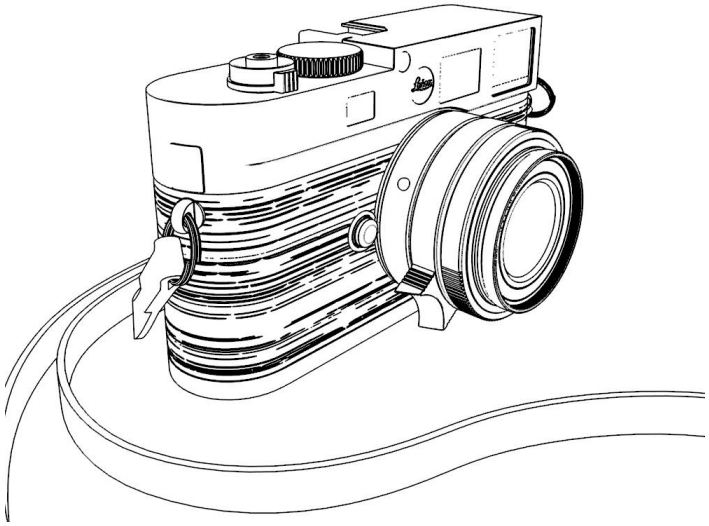
This option is quite powerful as you can recreate different shading techniques using a texture, like the image below:



Using a texture to create these contour lines

The texture used in the previous example

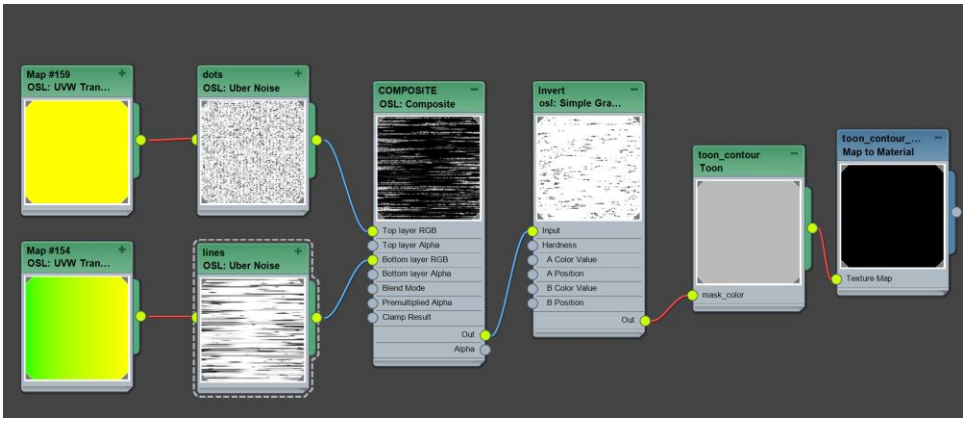
Or, if you want to use a more procedural approach, you can use two **OSL Uber Noises** to create the same look, although more organic:



See bellow how to create these contour lines

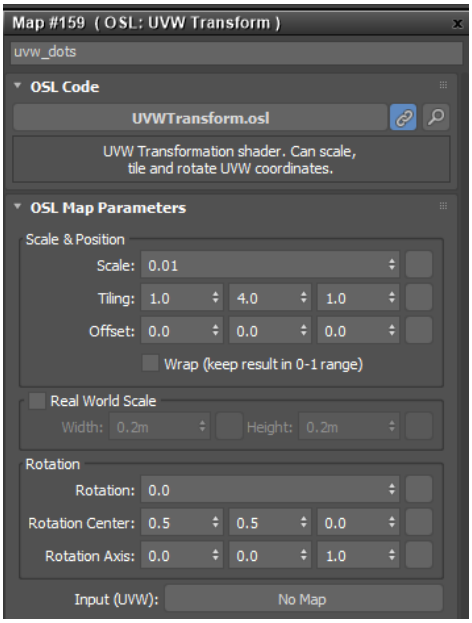
OSL Uber Noise - Contour lines

Very quickly, let's see how I created these contour lines using OSL Uber Noises. A big thanks to Lee Griggs for pushing me to achieve this. The shading network is quite simple for this one:

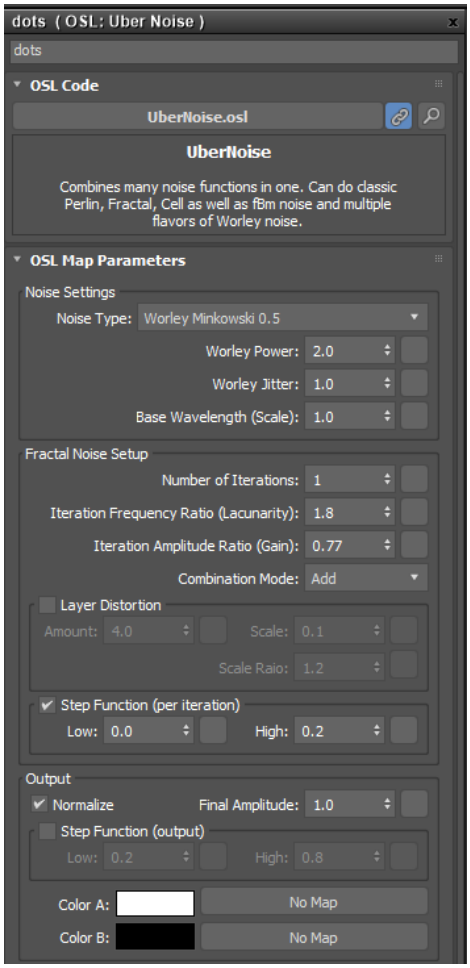


Contour lines done procedurally

Let me start by sharing the settings for the **OSL Uber Noise dots**:

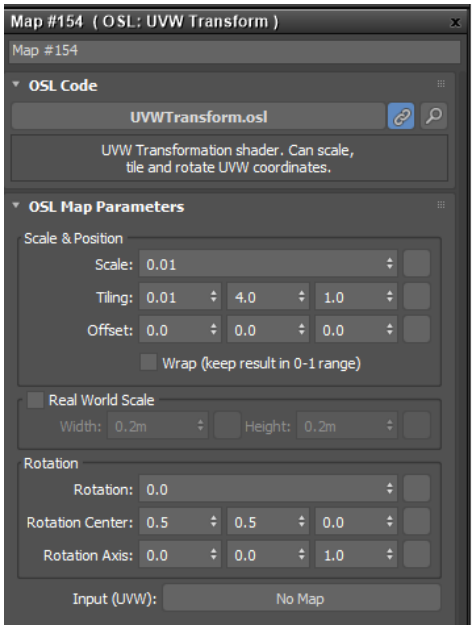


OSL UVW Transform connect to the OSL Uber Noise dots

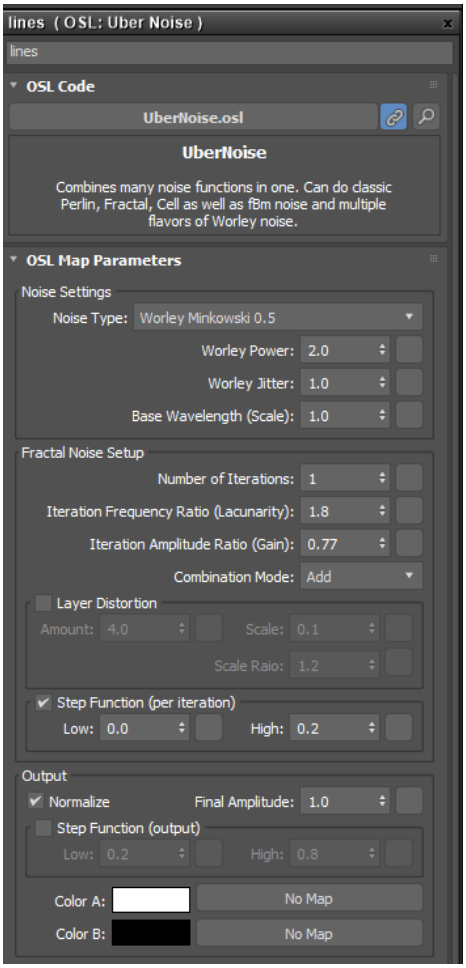


OSL Uber Noise dots

And the settings for the **OSL Uber Noise lines**:

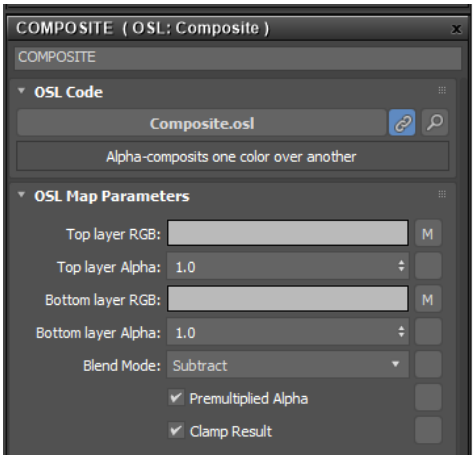


OSL UVW Transform connect to the OSL Uber Noise lines



OSL Uber Noise lines

The idea behind this shading network is to use simple math to create those contour lines. I don't want to make one continuous line; instead, I want the line interrupted in some areas. So, all I had to do was using an **OSL Composite** to subtract the **OSL Uber Noise** dots from the **OSL Uber Noise** lines:

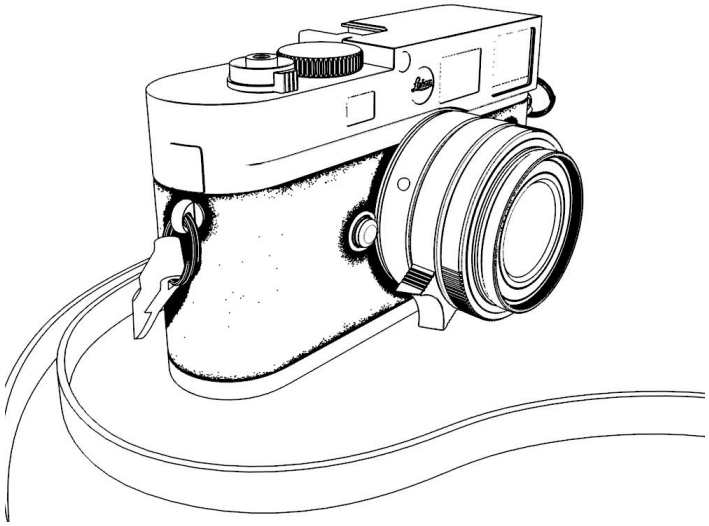


Subtracting the dots from the lines, will create the dash line effect I wanted

And the last node, the **OSL Simple Gradient** is used to invert the result so that the lines are black. Now, let's get back to the main topic.

So, not only you can connect a texture or an OSL noise, but there is another node quite useful to create other effects. Is the **Utility** node found under the **Maps - Arnold – Utility** section.

With this node, you have multiple shader modes to re-create different shading techniques. In the example below, I use the **Ambient Occlusion** shade mode:

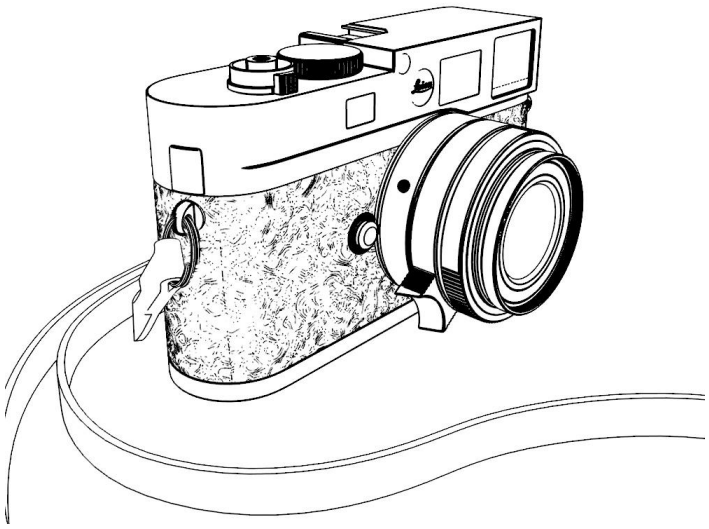


Using the Ambient Occlusion shade mode

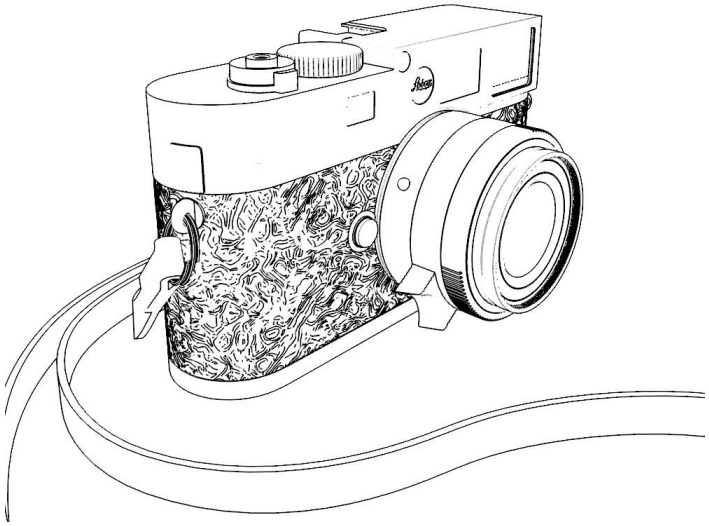
You also have other shade modes like metal and plastic, and you may want to play with the different Color modes. Remember that for these shading modes to work, and you need to have at least one light in your scene. There is another way to create these shading techniques using a normal or bump effect.

Normal\Bump Shading

As you can imagine, this technique uses a texture or an OSL Noise connected to a **Bump2D** or a **Normal map** node, and it can be used to create some interesting shading techniques, like stippling where the small dots are used.

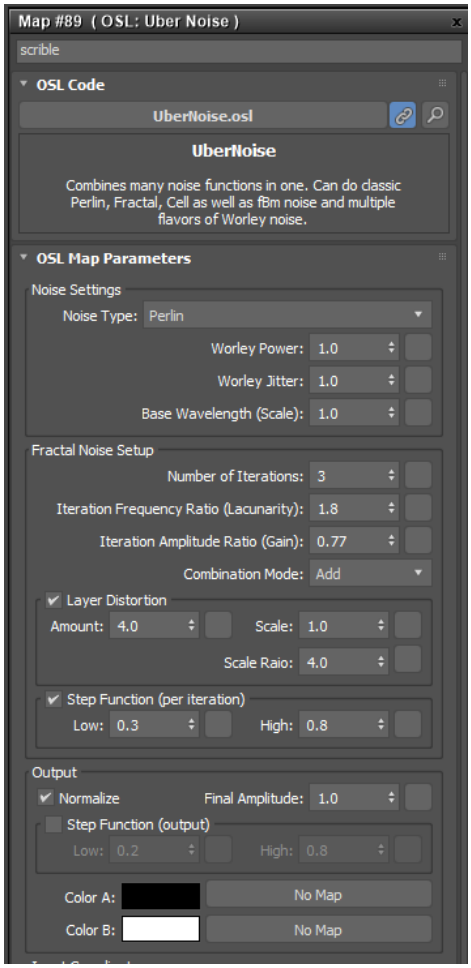


Achieving a stippling effect using a normal map



Using an OSL Uber Noise to create this scribble shading

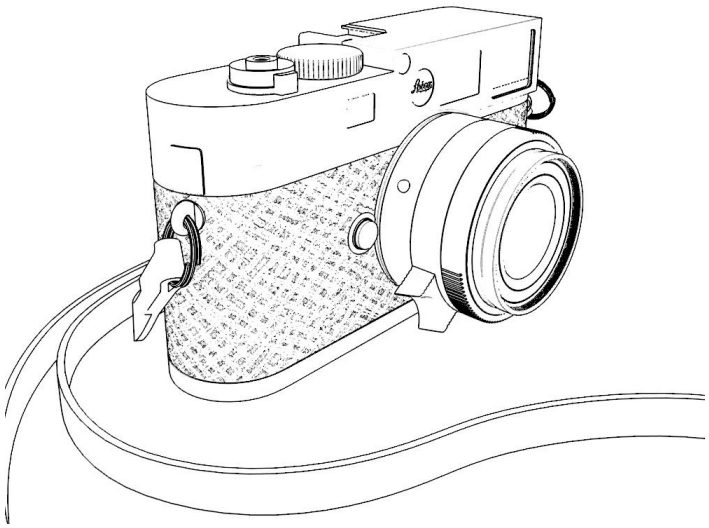
And below are the settings for the **OSL Uber Noise** I used to create this scribble shading:



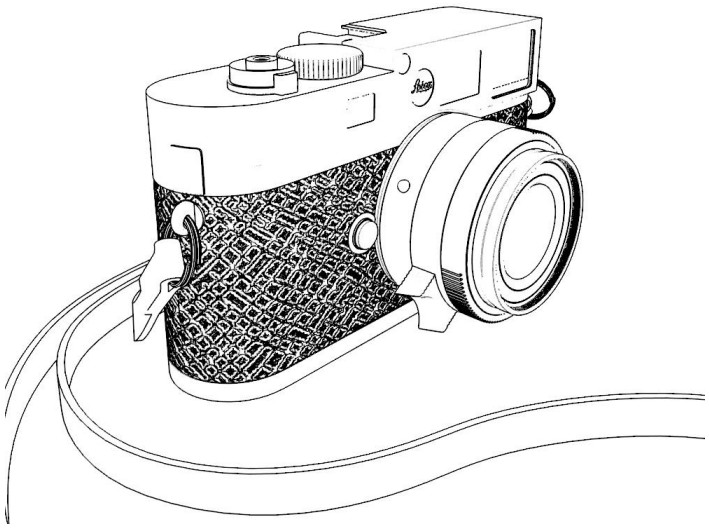
OSL Uber Noise scribble

I find that using a bump or a normal map provides more creative control than using the **Mask Color**. With the **Mask Color** technique to control the lines created, you need to use a node like the **OSL Range\Remapper** to remap the values that will reduce the texture or noise contrast\density.

On the other hand, using a bump or a normal map, you can increase or decrease the strength of this shading technique using the **Angle Threshold**. Here is an example of that:



Angle Threshold – 90



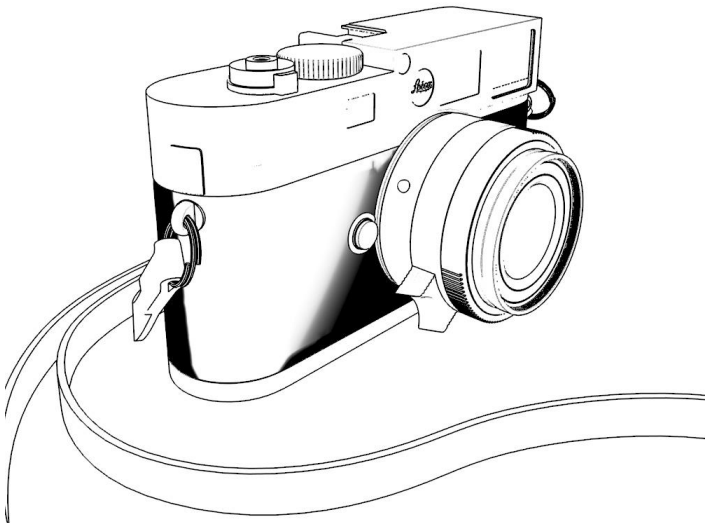
Angle Threshold – 40

Now that you know what you can do with the **Mask Color** and the Normal\Bump shading techniques let's explore some ways you can control how it is applied using some extra nodes.

Facing Ratio

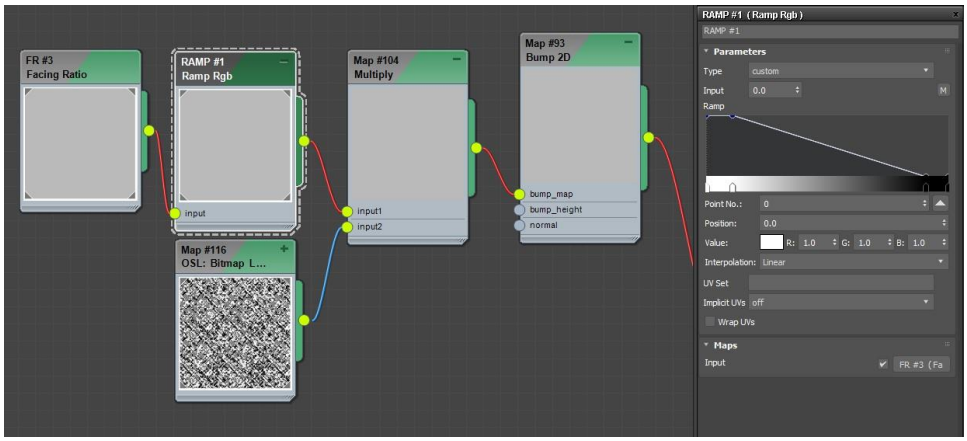
The **Facing Ratio** node can be found under the **Utility** section and is called **Facing Ratio**. You can use this node to create a mask and apply those shading techniques to specific areas of your 3D model. You can find more information about this node here: https://docs.arnoldrenderer.com/display/A5AF3DSUG/Facing+Ratio

Here is an example of that in action:



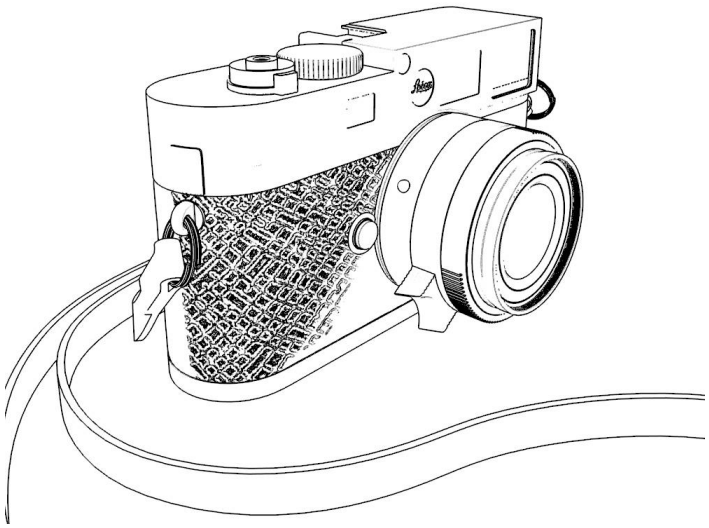
Facing Ratio to create a mask based on the camera

This setup is a little bit more complex, as you can see in the following image:



Shading network for Facing Ratio mask

I connected the **Facing Ratio** to a **Ramp RGB** to enhance the effect and then multiplied the output over the **OSL Noise** that I am using for the shading effect. The output of the **Multiply** node is then connected to the **Bump2D** node. And this setup creates the following look:



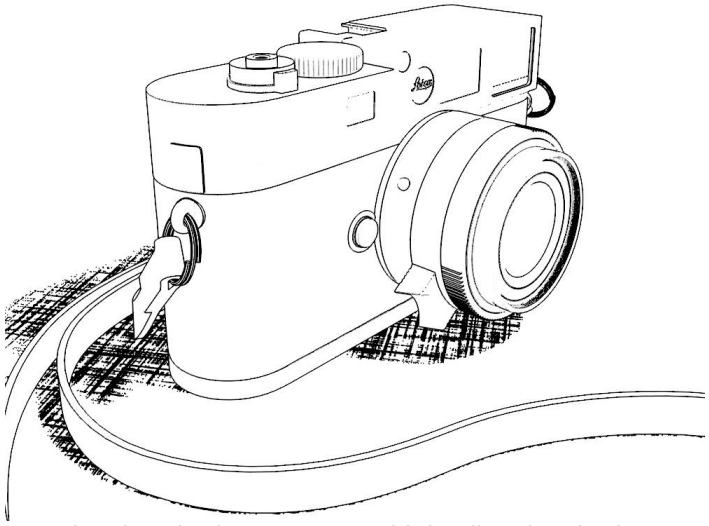
Shading based on the camera angle

It may look like a complex setup, but it provides a lot of flexibility. Using the **Ramp RGB** (Ramp Float also works), you can fine-tune the mask. You can also use this **Facing Ratio** node connected to the **Edge Width Scale**, and this can help clean some of the lines, making the lines seeing less busy.

You can also use another technique to create dynamic masks based on shadows, using the **Shadow Matte**.

Shadow Matte

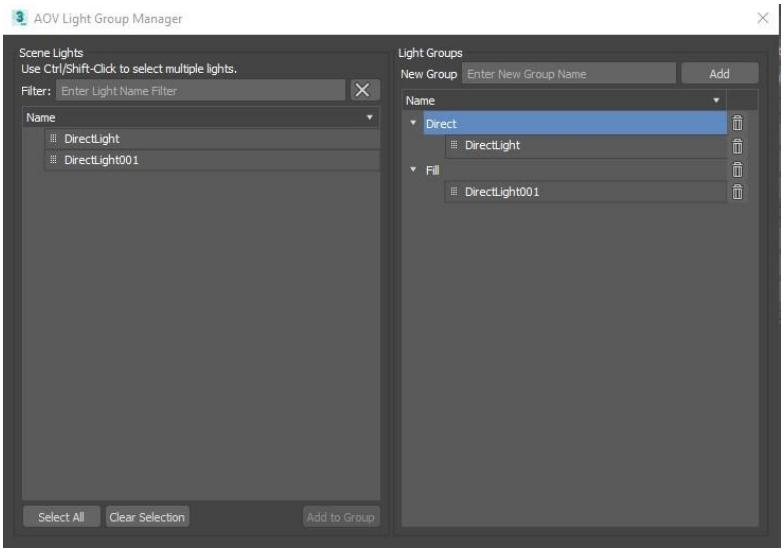
This tool can be found under the **Arnold** – **Surface** – **Shadow Matte**. Although this shader is mostly used to integrate a rendered object onto a photographic background, we can use it to add some detail to the shadows. Here is an example of what you can do:



Using the Shadow Matte to add detail to the shadows

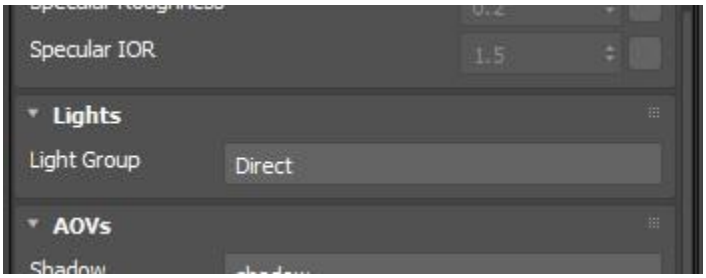
The setup is quite simple. Set the **Background** to **Background Color**, and this will create a black and white mask based on the lights affecting the scene. A couple of things to keep in mind.

If you have more than one light on your scene, you will probably want to select which light will cast the strongest shadow. In my case, I have two directional lights in this scene but will use only one to cast the shadow. To do that, you need to create a light group for the different lights. In my case, it looks like this:



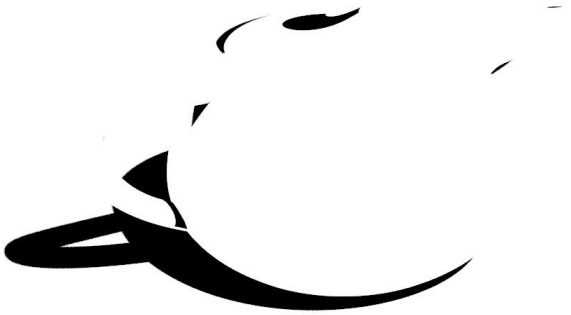
Create at least one light group for your main light

Then, you need to "tell" the **Shadow Matte** which light group to use so that only one set of shadows is cast. In my case, the main light is inside the Direct group, and that is the name that I type in the **Light Group** option:



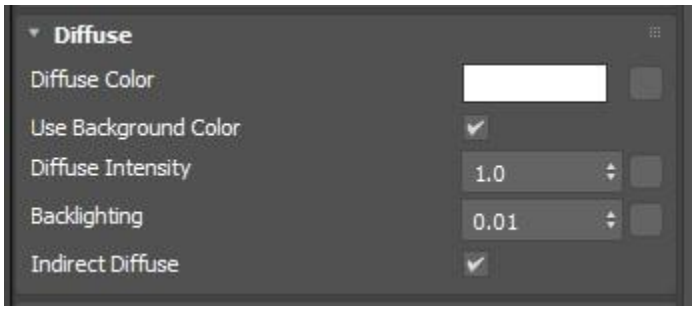
Shadow Matte set to use just the lights inside that group

The final point is that sometimes, the mask is created with the **Shadow Matte**, doesn't look 100% correct, like this example:



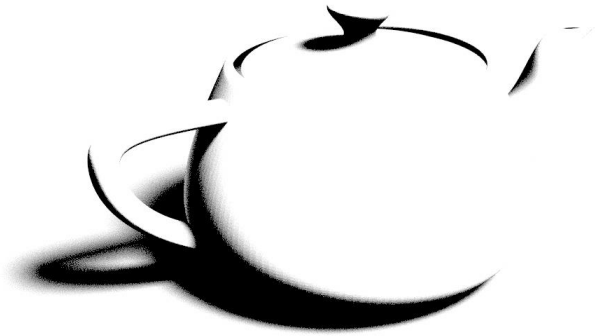
The back of the teapot should be in shadow

To correct this, you need to go to the **Diffuse** section in the **Shadow Matte** and set the **Backlighting** value from 0 to 0.01 or even 0.1:



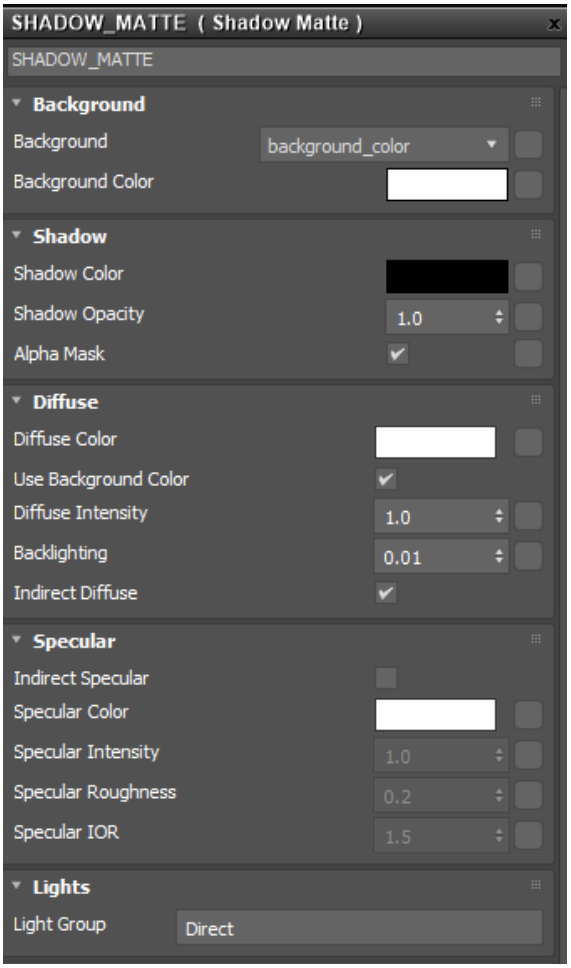
You just need that the Backlighting value is bigger than 0

This is going to help recover that missing shadow. Also, to avoid having such a harsh shadow, you could make the light softer. The setting for that is different depending on each light, but for the distant light, you need to increase the **Angle** value:



Now we have a correct softer shadow

Here is the final setting:

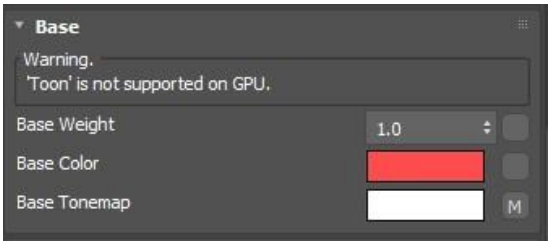


Shadow Matte Settings

Shadow Matte is undoubtedly a handy tool, not only to create masks but also to add color. You can, for example, connect the **Shadow Matte** to a **Ramp RGB** and remap the values for black and white (or add more if you want) and then connect that to the **Emission Color** so you can "fake" shadows and highlights. The following helpful tool is the **Tonemap**.

Tonemap

Tonemap is an extra option available the **Edge**, **Silhouette**, **Base** and **Specular**. This is a potent tool to create stylized looks. In simple terms, **Tonemap** is mapping colors to the shadows, mid-tones and highlights. As you can imagine, to use this technique, you need to have at least one light in your scene. Here is an example of how to work with **Tonemap**:

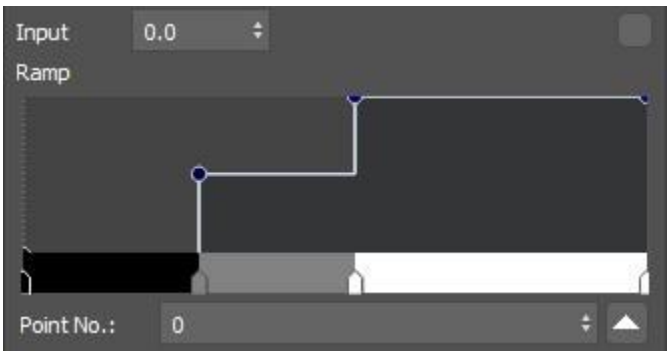


Base Color is set to red

The first thing to remember is that your base color, either for the **Base**, **Edge**, **Silhouette**, etc., changes when the **Tonemap** is applied. Let me show you an example of that:

Without the Tonemap

The image above is just a base color set to red, but if I connect a black and white Ramp RGB to the **Base Tonemap**, like this one:



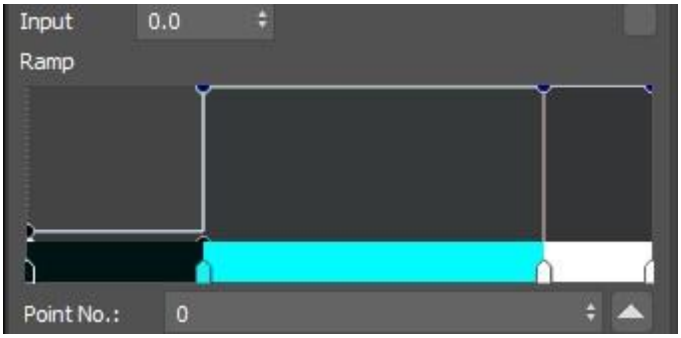
Ramp RGB with Interpolation set to Constant

The **Ramp RGB** is multiplied on top of the **Base** color, making something like this:

Tonemap being applied to the Base Color

As you can see, we don't get a black and white result, but instead a darker red in the shadows and a brighter red on the highlights. Keep in mind that this effect depends on the lights you have in your scene and their intensity. My advice is to have only one directional light and then, if necessary, use the **Emission** to brighten up different areas in the scene.

The same principle applies to the other elements that have a tonemap option. Here is a quick example of the same principle being applied to the **Edge**, but using a colored **Ramp RGB**:



Ramp RGB with colors for the 4 positions

Same principle here as the one we saw for the **Base Color**. This time, my **Edge Color** is set to a yellow and using a colored **Ramp RGB** with some blue tones connected to the **Edge Tonemap**, I will get some greenish edges, as you can see below:

Edge with a tonemap

The 30-minute talk covers this technique in more depth and shows other examples of how you can use the Tonemap to create interesting stylized looks. You can also check this tutorial from Lee Griggs, showing this technique in action:

How to render an attic scene using the toon shader in Arnold

Use Arnold Operators to apply distinctive styles

Now that you know how to use and apply these techniques, I wanted to present one tool that had a massive impact on speed and efficiency in my workflow. I am referring to the use of **Arnold Operators**. These Operators allow you to override any part of an Arnold scene and modify it when you press render. For example, you can have one material in your scene, but different material is applied when you press render. Why is this useful?

In a fast-paced environment, where you may need to produce a couple of renders quickly, any time you can save is welcomed. For example, when I receive models from architects, they rarely will be the final model, but at the same time, they tend to follow a specific naming convention for the objects. Having that naming convention helps me save time because I can automatically assign materials to those models based on that naming.

So, right from the start of your project, you need to have things relatively organized. Avoid the famous object056 or box004 naming convention, as it doesn't help identify the objects. Instead, having a mesh called glazzing_top_level_arch gives me exactly what I need to assign materials and, in this case, Arnold Toon Shaders. Imagine having one 3ds Max scene with all the stylized looks you developed, and then all you have to do is bring those 3d modes as an Xref and press render. Sounds too good to be true? Let's see how to do it

You can find more information about **Arnold Operators** here:

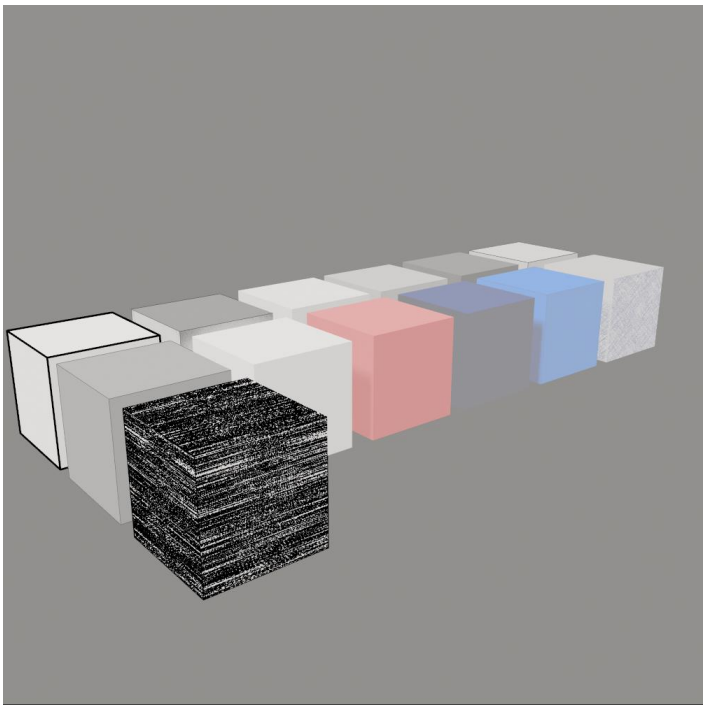
https://docs.arnoldrenderer.com/display/A5AF3DSUG/Operators

https://docs.arnoldrenderer.com/display/A5AF3DSUG/Operator+Tutorials

Arnold Operators - Naming

The first thing to do, as I mentioned, is to name the geometry in your 3ds Max file. Doing this will depend on who provides the 3d model and what they used to organize the objects. Most of the time, the models provided by the client will have some naming convention and usually involve the material they used in Revit or Rhino. That is perfect because you can use wildcards to make selections and apply the materials. Remember that **Arnold Operators** don't work with layers, so you may need to select all the objects in a layer and quickly rename them in the worst-case scenario.

The second thing you need to do is to create a couple of cubes (or spheres, it doesn't matter) so you can apply the **Arnold Toon Shaders** you made to those cubes. Here is an example of what I do:

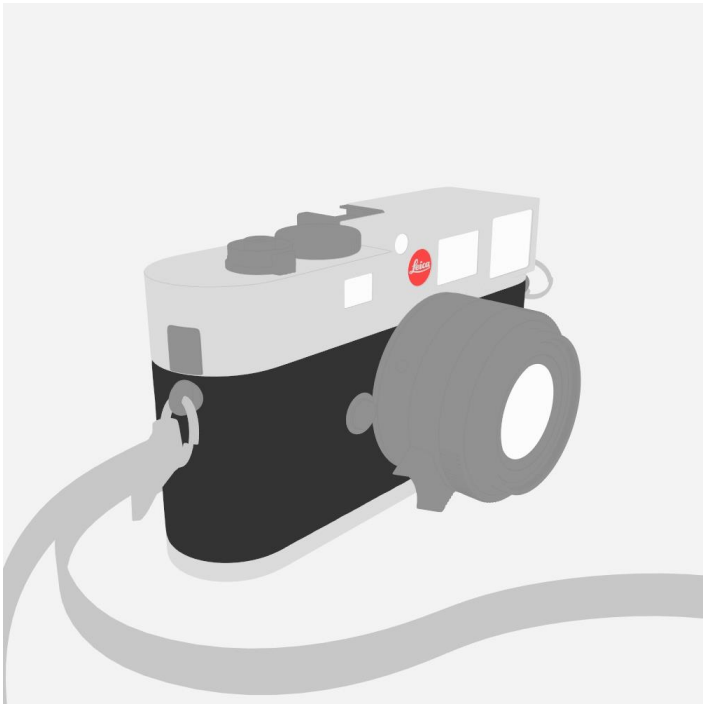


Each cube has a different Arnold Toon Shader assigned

We need those cubes (or any other mesh you may fancy) because in 3ds Max, if a material isn't assigned to a mesh, then that material doesn't exist in the scene. If the material doesn't exist in the scene, we can't assign using the **Arnold Operators**. Also, make sure you give meaningful names to your materials; something like Material#335445 will make your life harder. So now that you know, the basics let's see an example in action.

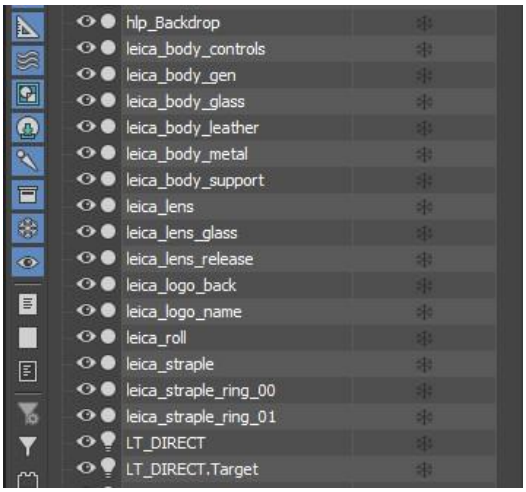
Arnold Operators – Operator Graph

Let’s see how I created this render without having to assign materials manually:



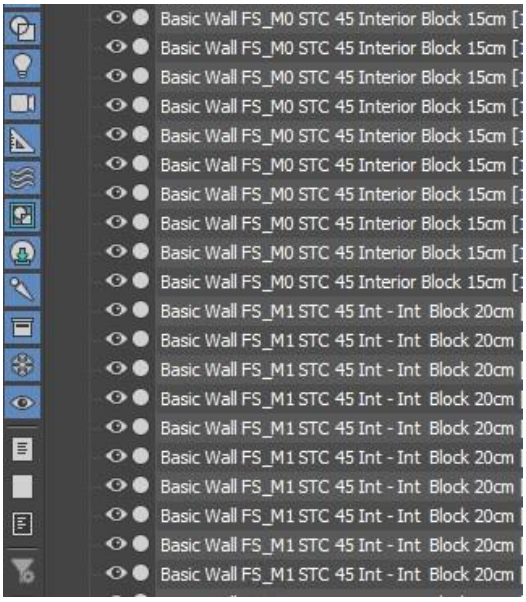
Notan style

The first step is having the model named, which in this case looks something like this:



Naming based on different parts for this model

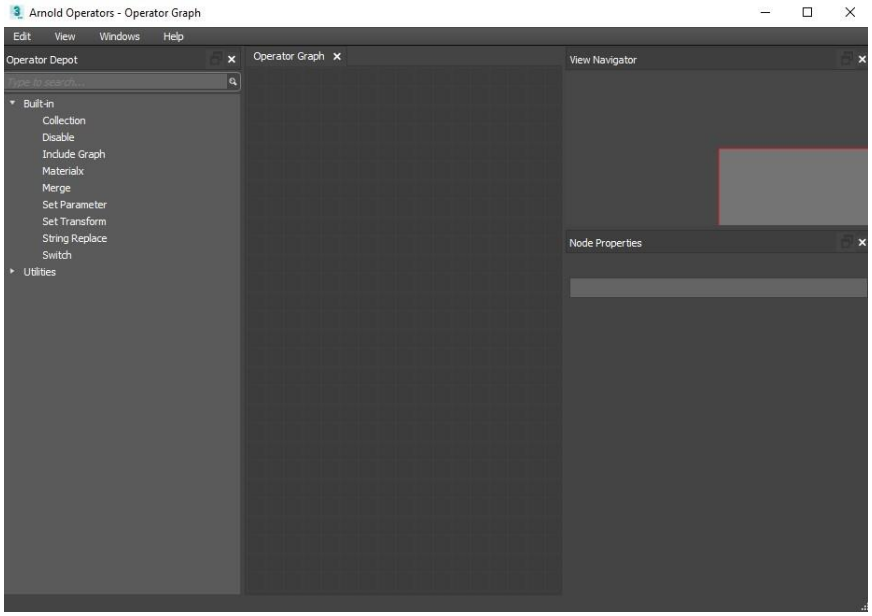
This is a simple example to explain the technique, in a production environment, you may have something like this:



A production model

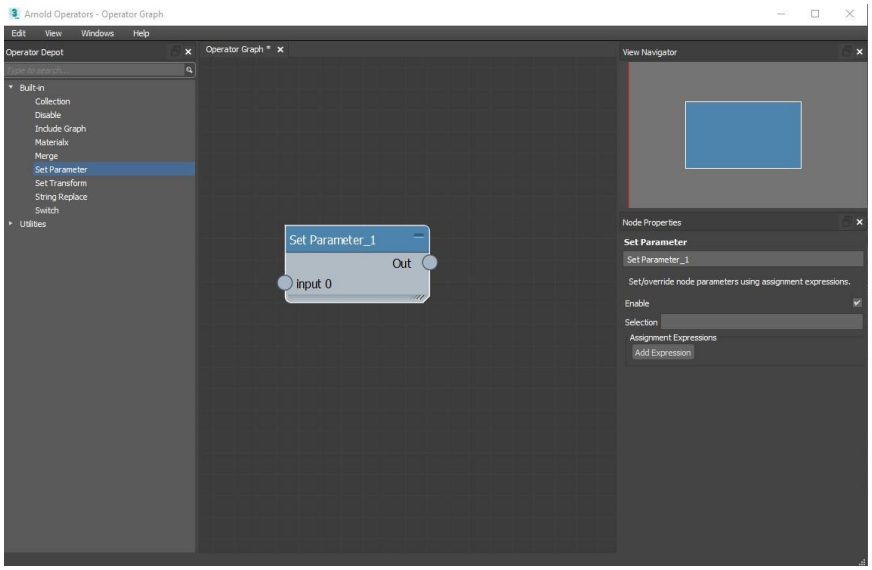
In the example above, I could make my selection using the ***45 Interior*** naming convention and use the ***45 Int*** to make another selection. How?

The next step, is opening the **Operator Graph** window, which you can find on the Arnold menu, and it looks like this:



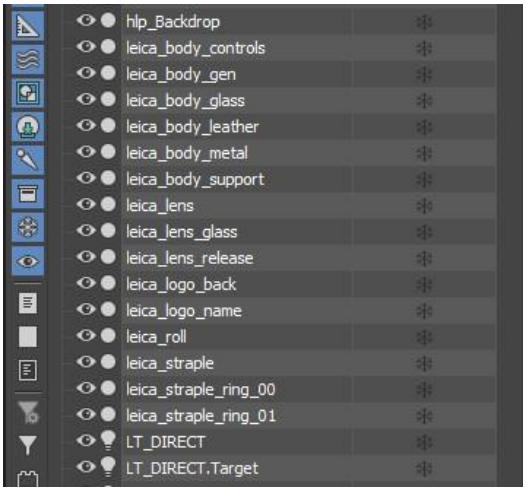
Operator Graph

So now what? The setup is quite simple: you need to create a collection of objects and assign a material. Start by selecting the **Set Parameter** and drag it to the middle section:



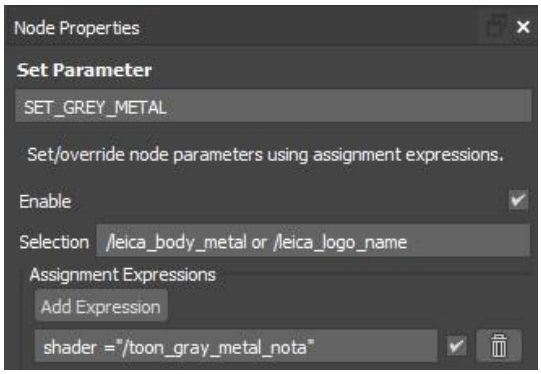
Set Parameter node

See the option called **Selection**? Use this to define which objects will be selected in your scene. Let's get back to the example I am using:



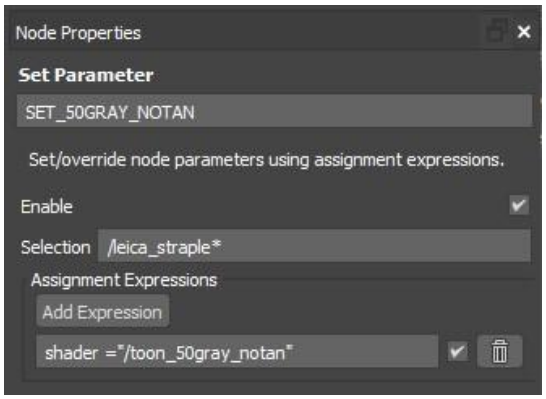
My Leica model divided into sections

Several ways you can make selections. If you want to select two or more objects, use the */something or /something_*something, like this:



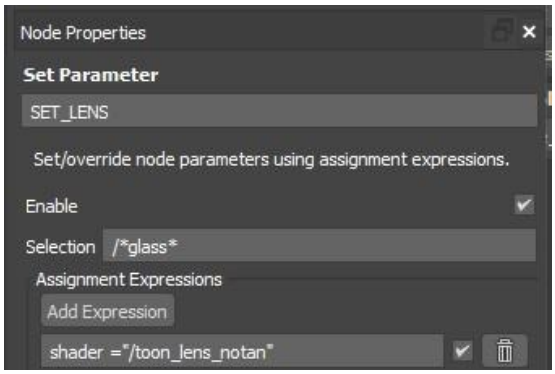
Selecting multiple objects

You can do this for smaller projects, but not practical for production scenes unless you have something particular in your scene. In most of cases, you will want to use a selection like this:



Selecting all the objects with leica_straple

The asterisk (*) tells Arnold to select all the objects started with ***leica_straple*** and apply this material. On the other hand, use the /* to select everything in your scene if you want to apply one material to everything. However, the best option for me is to use this selection:

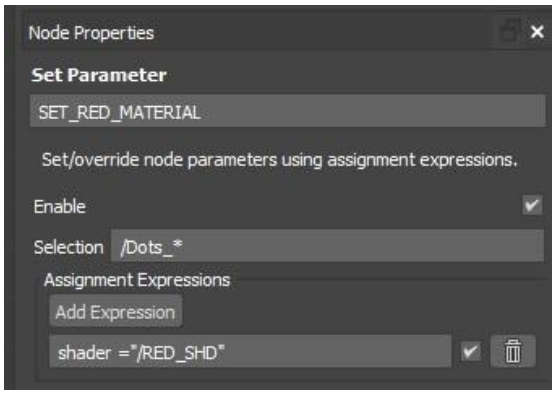


Using an asterisk to isolate words

This is by far the best method for any production scene. By using an asterisk before and after the glass word, I am telling Arnold that no matter what is before or after, as long the word glass is present, select it and apply the material.

Now, how do we assign a specific material to my selection? Click on the **Add Expression** button to add a new field where you can type the following:

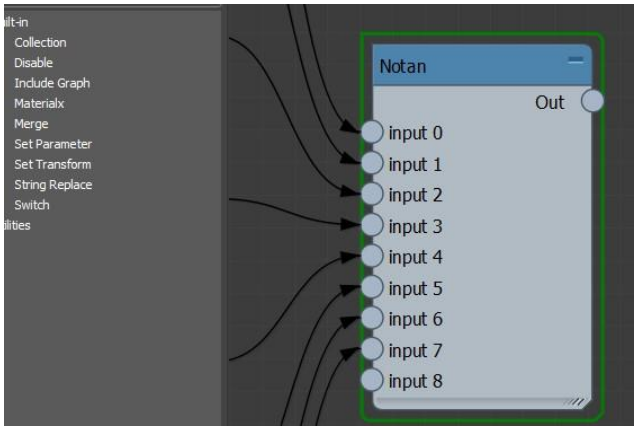
Shader ="/SHADER_NAME"



Using the Set Parameter to assign materials

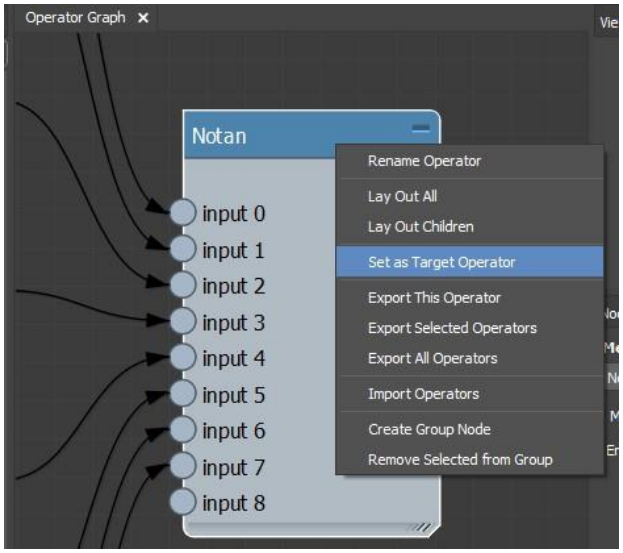
When working with the **Arnold Toon Shader**, you need to assign the **Toon** shader itself. If the geometry you assign the material turns solid purple, that means something is wrong with the material expression. Check the name and make sure you are not missing any quotations marks.

Now that you have all of the selections done, you need to merge them using the **Merge** operator:



Merge node with multiple inputs

The next step is telling Arnold to use this node to render the scene, so select the **Merge** operator, right-click and select the **Set as Target Operator** option:



Selecting this option will make this node active

When this option is selected, a green outline will appear, indicating that the node will be used when we render.

This workflow is very efficient because you can have one 3ds Max scene with all the **Arnold Toon Shaders** that you created and assigned it to thousands of objects with a single click. On top of that, you do not even need to merge the objects; it works perfectly fine with Xrefs, which makes it even easier to manage it.

Final words

I hope this brief overview of some of the most practical techniques and tools will set you on the right track. I did not cover some of the options, like **Specular**, **Sheen**, **Transmission**, but that is something that you can check in the documentation. I share techniques, materials and tips, so keep an eye on my Twitter or my ArtStation.

A big thanks to Lee Griggs, Dan Cherry, Declan Russell and Ivan Genov for their help in reviewing this handout and for all the valuable feedback.