

PREFACE

This basic Houdini/RenderMan tutorial is written primarily for students and new users of Side Effects Software's Houdini who want to use RenderMan renderer in conjunction with Houdini's powerful Mantra renderer. Much of the focus here is emphasized upon the preparation of RenderMan for use with Houdini under Windows environment. I am certain that the basic set up is somewhat applicable for Linux and vice versa as well.

Since you have already gotten to this page, I presume that you have at least the basic understanding of Houdini and RenderMan. Therefore I will not go into detail in trying to explain what Houdini or RenderMan does. I am presuming that you have at least some understanding in setting up the environment variable under Windows and/or Linux. I am also presuming that RenderMan software has been installed properly in your system.

In Houdini, RenderMan is tightly integrated and is part of the design consideration when Houdini was first built years ago. Consequently, you will find that Houdini's RIB Output is much cleaner and extremely fast compared to any other 3D applications or RenderMan plug-ins (including MTOR) exist today. Houdini is very direct with RIB generation. After you understand the basic set up for RenderMan to work with Houdini, you will find that Houdini's RenderMan integration is very straight forward. Although writing a RenderMan Shading Language can be highly technical, I hope that you are able to spend some time to understand the RIB structure at the very least.

This tutorial is written as of Houdini Master v5.5 and PRMan Release 10.
With all that being said, let us now move on.

Using RenderMan with Houdini

Setting the environment variable

As with much of the high-end CG applications, the environment variables are fairly important for how certain software executables are called. Generally, the environment variables are often similar to a short-cut to let software know where a particular file or directory is located. However, this is very much depending on how a software is designed.

For Pixar's PRMan (Photorealistic RenderMan) under Windows, they do not seem to append the path location pointing to the directory where prman.exe is located. Therefore, you should check to see if the environment variable for "path" exists by going into Windows' Environment Variables. This particular "path" variable may exist either as a system variable or as a user variable. Here is an example:

```
PATH = %RATTREE%\bin;%RMANTREE%\bin;C:\Exluna\Entropy\3.1\bin
```

%RMANTREE% is pointing to the directory where PRMan is installed.

%RATTREE% is pointing to the RenderMan Artist Tools directory. This will be needed if you intend to use Alfred Output or if you specify "it" (Image Tools) as your Image Device option in RMan Output.

Both the environment variable RATTREE and RMANTREE are appended by Pixar's RenderMan during installation and is only applicable to Pixar's RenderMan. For Entropy or RenderDotC, just point to "/bin" according to the software's installation directory.

Now, why is specifying "path" (like the above) for PRMan important? Well, it is not. However, this will simplify your work a little bit easier in the RMan Output "Render Command" field. If the above path is not specified, Houdini will not know where prman.exe (or entropy.exe if you are using Entropy) is located in a maze of directories. Thus, you will have to type the full installation path like "C:\Pixar\prman\bin\prman.exe" into Render Command in order for Houdini to know the location of prman.exe. By specifying the path to the directory where prman.exe locates, you may simply type "prman" in your Render Command field.

Using NetRender for multiprocessor rendering

For multiprocessor users, you can exploit PRMan's parallelism through Network RenderMan. In short, Network RenderMan will assign each render bucket to the specified host to render. Each frame will render much faster if they are rendered locally. This may be slower if you are rendering across the network depending on your network speed and the remote computer. If you intend to use Network RenderMan for network rendering, please make certain that the network is properly configured. In RMan Output's Render Command field, please type:

```
netrender -h host1 -h host1 -h host2 -r prman  
(Where "host#" is the name or IP address of the machine you want to use)
```

The reason we type "-h host1" twice is to have Network RenderMan assign a render job to two processors in a single computer. It is essentially the same idea as having a distributed network rendering with two computers. In the case above, the two computers exist as one.

For some reason possibly due to the installation of Alfred Server instead of NetRender Server, Network RenderMan will only work if the "Pixar Alfred Server x.x" service is started in your local computer - even if your local computer acquires the RenderMan license from a remote system. Sometimes, the Alfred service may not get started on time by the Windows Services. Therefore, if you find your Network RenderMan not working, you may manually type the following in the command prompt to start the service:

```
net start "Pixar Alfred Server x.x"  
(Where x.x is the version number. This is applicable to Windows ONLY.)
```

If this still will not work, then it's time to head off to your system administrator's office. For more information about NetRender command, please type 'netrender -h' in

the command prompt. Please note that Network RenderMan is not for large render job management. For such task, Alfred may be used. NetRender is only available in Pixar's RenderMan installation. Entropy already has multi-thread process built in to take advantage of multiprocessor computer.

With all these initial tasks taken care of, using RenderMan in Houdini to render should be fairly straight forward.

Shaders and Dialog Script

Depending on your need, you can bring custom RenderMan shaders into Houdini in two different ways. One way is to do it without a visual interface. The other way is to bring a shader in and generate a Dialog Script for Houdini. The latter is a little more involved, but they are all fairly simple.

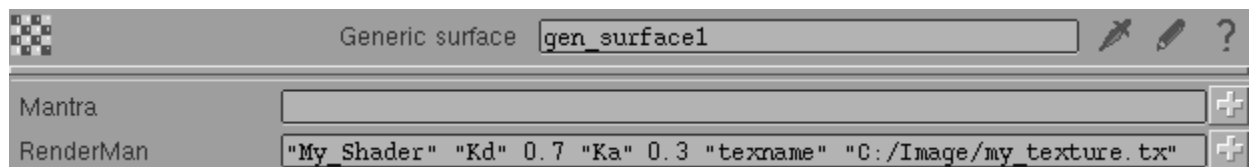
Without the Visual Interface

The easiest way to bring use a custom RenderMan shader is simply to use a Generic SHOP* and specify the name of your custom shader along with its parameter. Although this approach can be faster, you will not have any dialog parameter to slide the value range visually. You also cannot animate the parameter with this approach either unless you manually change the value by hand or write a script to handle such task. But then this would defeat the purpose of the Dialog Script.

If shader animation is not required, any models that use only a static shader usually may not require an accompanying dialog script. Therefore, using a Generic SHOP* can be much faster since Dialog Script is not involved.

** Depending on the type of shader, you will need to use Generic Displacement SHOP, Generic Light SHOP, etc. to its corresponding shader type. Please visit your RenderMan documentation for different shader types if you are not familiar.*

You can choose two different way to assign a custom shader as well. First, let us begin in the SHOP editor and append a "Generic Surface" SHOP under Generator list. In the Generic Surface's "RenderMan" field, type the name of your own Surface shader along with its parameters (if any). You may have something that looks like this:



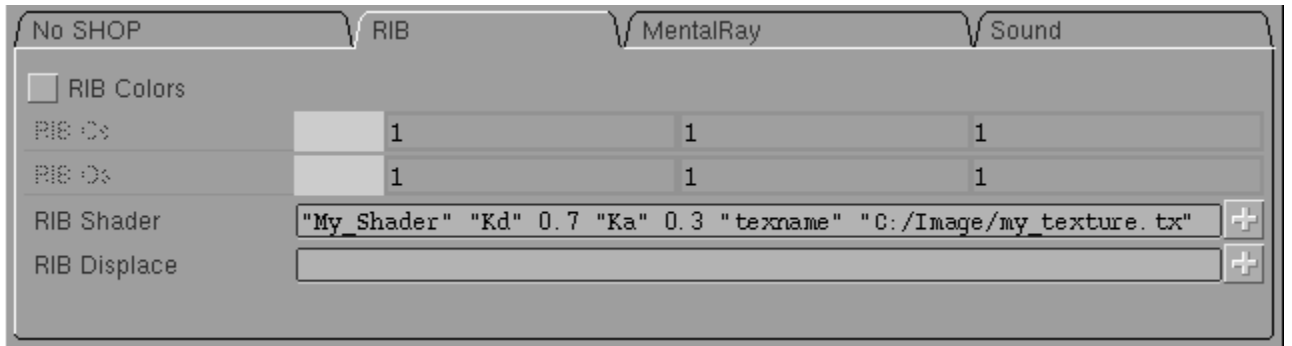
(Figure 1)

As you have noticed, the syntax is exactly the same as those found in RIB (RenderMan Interface Bytestream) file. Only that the word "surface" (shader type) is

omitted when you type your syntax into the RenderMan field of the Generic SHOP. Here is an example of a syntax from RIB file:

```
surface "plastic" "Ks" 0.5 "Kd" 0.7 "Ka" 1
```

The second way of assigning a custom shader without involving SHOP editor is simply to specify your shader name and its parameter to the RIB Shader field ("Shading" tab > "RIB" tab > "RIB Shader") of your Geometry Object. This will apply the specified shader for all your geometry inside Object node.



(Figure 2)

One thing to keep in mind is that the shader name and its parameters in the RIB syntax must match those declared in the Shading Language – they are case-sensitive.

Creating a Visual Interface

So what is the Dialog Script? A Dialog Script is a particular text file with a list of parameters or commands that tell Houdini how to construct a visual interface from the shader's parameter. With visual interface, a user may keyframe and make changes to the value visually in SHOP editor. Dialog Script is also applicable to VEX (Vector EXpression language) and Mental Ray shaders.

It is very important that you follow the direction in this section since the naming convention, directory structure, and the Environment Variable will determine how Houdini searches the custom Dialog Script. There are about two to four important steps involved before getting the custom shader to load in Houdini in conjunction to Houdini's default shaders.

- 1) Compile your *.sl (Shading Language) file if you have any.
- 2) Use 'rmands' command to generate the necessary Dialog Script file.
- 3) Put your *.ds file into its proper directory. (Optional)
- 4) Create or edit your SHOP{*shader_type*}.ri file
- 5) Specify the Environment Variable
- 6) Start Houdini.

Whether you want all users to have a particular Dialog Script or on a per-user basis will dictate how you deal with the directory structure and your SHOP{shader_type} file. First, we will deal with the per-user case. Then we will deal with the easier topic on global-user case.

Creating a Visual Interface on Per-User Scenario

(Special Thanks to Marc Horsfield of Side Effects Software for putting up with me. ^_^)

If you have any *.sl file that you want to bring into Houdini, please compile it first so that you may have SLO (PRMan), SLE (Entropy), or SO (RenderDotC) format depending on which renderer you choose.

After you have the shader you need, you can use Houdini's 'rmands' command to generate the Dialog Script (*.ds) file from the shaders that you have. In command prompt, change your directory to the location where your shader resides and enter something similar:

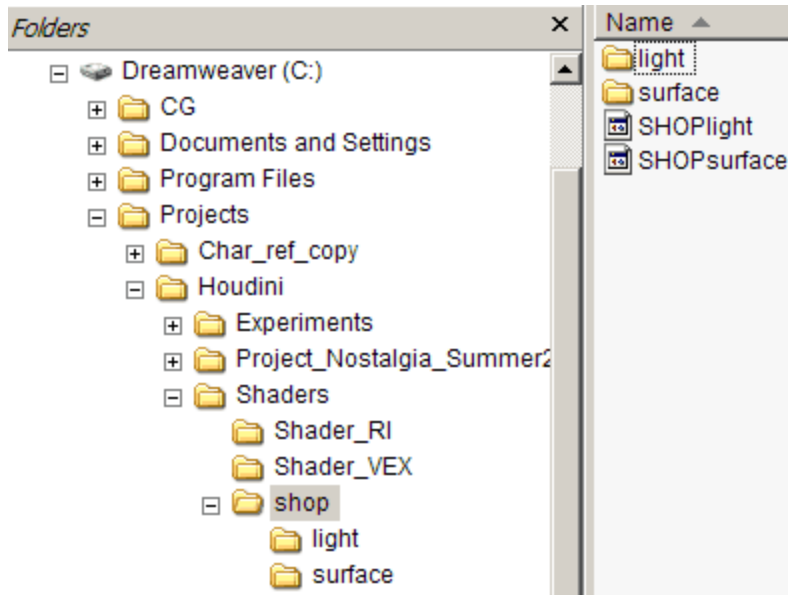
```
rmands Alex_Shader.sle Some_Other_Shader2.slo  
(For speed, you may specify multiple shaders in one rmands command.)
```

After you have successfully generated the Dialog Script file, you will notice that a new folder is created in the directory that you ran your 'rmands' command. The name of the folder will correspond to the shader type you have. So if you generated a Dialog Script for a Surface shader, you will see a folder named "surface". Inside this folder will contain the Dialog Script file. The filename of the Dialog Script file does not have to be the same as the filename for your shader, but it should be something that you will recognize and able to associate them if you choose to rename them.

Now is the very imperative section.

You MUST mimic Houdini's "../shop" directory structure for your own custom directory. There is no other way around this. In your "\$HFS/houdini/shop" folder -- where \$HFS is the location of your Houdini installation directory -- you will find a list of folders include "surface", "displace", "light", etc. as well as a list of files such as "SHOPsurface", "SHOPdisplace", "SHOPlight", etc. depending on the type of shader. Hence, you will need to mimic the \$HFS directory structure on which ever shader type you want to use.

For example, on where ever you want your own project folder to be, inside that project folder, you must have a "shop" folder. Inside this "shop" folder will contain your {shader_type} folder as well as SHOP{shader_type} file. **The most important things that you must have here is the "shop" folder and the SHOP{shader_type} file inside the "shop" folder.** The {shader_type} folders where your actual Dialog Script resides are mainly for organizational purpose and can be located anywhere in your computer as long as you specify it in your SHOP{shader_type} file. Figure 4 shows what mine look like if I only need custom Light and Surface Dialog Script to be appended into Houdini's SHOP editor.



(Figure 4)

It is also very important that your custom SHOP{*shader_type*} file does not have any file extension.

After you have taken care of the custom directory structure, you must edit the SHOP{*shader_type*} file in your custom “shop” directory. If SHOP{*shader_type*} file does not exist, please create a new text file, rename it, and remove the .txt extension. Please DO NOT edit the SHOP{*shader_type*} in the \$HFS/houdini/shop directory – at least not on Per-User case. In this script will contain your SHOP shader name, location of this particular Dialog Script file, and the label it will appear when you hit the “Tab” key in SHOP’s network editor.

```
//SHOP Shader Name-----Location of the *.ds file-----Label name

//Entropy
en_caustic      ../light/en_caustic.ds      -label "RMan/Entropy Caustic"
en_envlight     ../light/en_envlight.ds     -label "RMan/Entropy Environment Light"
```

(Figure 5)

The SHOP shader name and the label name can be any name you like. However, it is more preferable that you use the name that you can easily associate with a particular shader. The *.ds path location can either be a relative path (figure 5) or an explicit path (i.e. “C:/home/shop/light/en_caustic.ds”).

Note to Houdini 5.0 users: If the above example in Figure 5 does not work, you may need to append the following line in the topmost of your custom SHOP{*shader_type*} file:

```
include "$HFS/houdini/shop/SHOP{shader_type}.ri"  
(Where {shader_type} is the... corresponding shader type...)
```

With all the file and directory structure properly set, you will now have to deal with the Environment Variable once more. This will not be necessary if you are setting an interface where all users will share the same Dialog Script. But on a Per-User scenario where only a specific user will use a particular Dialog Script, you will need to specify the environment variable in order for Houdini to know where your custom SHOP{*shader_type*} file is located. Here you will specify a Houdini-specific variable: HOUDINI_PATH. For example:

```
HOUDINI_PATH = $JOB/Shaders;H:/folder;&
```

HOUDINI_PATH variable will tell Houdini to search for dialog files and configuration files in the specified location. The "&" will tell Houdini to use "whatever that was already there." \$JOB is another Houdini-specific variable (unless some other program that use JOB variable as well). In the Environment Variable, I set JOB to my custom project folder. So in my case, the environment variable for JOB would be "JOB=C:/Projects/Houdini".

In HOUDINI_PATH, notice that I did not specify the "shop" folder? When the "shop" folder exists in your specified folder, Houdini will automatically look into the "shop" folder for further instruction – which in our case, it is the SHOP{*shader_type*} file. If you name "shop" folder any name, Houdini will not look further.

After everything is set, you can start Houdini and go to the SHOP editor. Whatever name you choose for your label name will appear when you hit the tab key in the SHOP editor.

And finally we will need to take care of the RenderMan shader search path so that RenderMan will know where to find shaders during render time. Let us move on.

Creating a Visual Interface on Global-User Scenario

This section will be for those who are setting a visual interface for shared users on the same computer.

After you have generated your Dialog Script file, you will need to go into the \$HFS/houdini/shop directory -- where \$HFS is Houdini's installation directory -- and edit the SHOP{*shader_type*}.ri file for RenderMan. But first, you will need to make sure that the SHOP{*shader_type*}.ri file you are about to edit has the "Read-Only" attribute unset. You may do this in its file property or in command prompt. In command prompt, please type 'attrib -R SHOP{*shader_type*}.ri' to unset the "Read-Only" attribute.

Since this Dialog Script will be shared by everyone, you should copy your custom Dialog Script to \$HFS/houdini/shop/{*shader_type*} folder depending on the type of shader. When you are ready, you may append your custom name, description, and Dialog Script location (as shown in Figure 5).

Afterward, just fire up Houdini and ready to go.

And finally we will need to take care of the RenderMan shader search path so that RenderMan will know where to find shaders during render time. Let us move on.

RenderMan Shader Search Path

Dealing with shader search path for RenderMan is fairly straight forward as well. There are also a few different ways you can approach dealing with the shader search path. One way is to specify the search path in RMan Output's "Shader Path" under "Specific" tab. Other approach includes environment variable HOUDINI_RI_SHADERPATH, or RenderMan's "rendermn.ini" configuration file.

In this section, "RenderMan" will be referring to RenderMan-compliant renderer in general. "PRMan" will be specifically be dealing with Pixar's PRMan.

For example, in the RMan Output's "Shader Path" field, I have the following set:

```
":&:$HFS/houdini/ri_shaders:$JOB/Shaders/Shader_RI:H:/home/shader"  
(Include double quote. Use colon instead of semi-colon.)
```

In the above example, the "." will tell RenderMan to look in its local directory first. Then again, "." is not quite necessary for most of the time. "&" will tell RenderMan to use whatever was set before. The "@" symbol -- which tells RenderMan to use the standard search path -- can be used as well. In "\$HFS/houdini/ri_shaders", I tell RenderMan to look for all the default shaders found in Houdini's installation directory. Afterward, I specify the location to my custom shaders directory. The order of which comes first usually isn't quite mattered in the search path.

If you only need default shaders from Houdini and PRMan, there is no need to specify any Shader Path in RMan Output as it has already been set by installation. If you are using your own custom shaders in conjunction to the default shaders, you must have "&:\$HFS/houdini/ri_shaders" in conjunction with your own custom shader directory. Otherwise, PRMan will render without any shaders. For other RenderMan-compliant renderer, you may need to specify where its default shaders are.

What happened after you specified the Shader Path in RMan Output is that Houdini will append the search path into RIB file. According to the shader path in the above example, Houdini will generate the follow RIB search path (all in one line) in the RIB file:

```
Option "searchpath" "shader"  
[ ".:&:C:/CG/SideFX/HOUDIN~1.33/houdini/ri_shaders:C:/Projects/Houdini/  
Shaders/Shader_RI"]
```

Using the HOUDINI_RI_SHADERPATH variable is exactly the same as the Shader Path in RMan Output. This is in place of Shader Path in RMan Output. It can be

used on a per-user case or global-user case. Double quote will not be required here. This variable will have effect only if you do not specify any Shader Path in the RMan Output, Houdini will use the Shader Path according to HOUDINI_RI_SHADERPATH variable instead. Depending on your requirement, HOUDINI_RI_SHADERPATH may not be needed.

Using the “rendermn.ini” configuration file (\$RMANTREE/etc) will specifically tell PRMan where to look for shaders. Whatever path you specify in rendermn.ini file will apply to all PRMan rendering. Whether you use your custom text editor to write a simple RIB file, use MTOR, or user Houdini’s RMan Output, PRMan will find the shaders that you specify in rendermn.ini. (HOUDINI_RI_SHADERPATH and RMan Output is specific to Houdini’s output only) Editing rendermn.ini will affect PRMan only and all users on the same computers.

In the “rendermn.ini”, find a line that read “/shaderpath”. Here you can type the shader search path just as you would with all previous approach. However, if you will be specifying any variable-related path, you must enclose them with \${} bracket. Here is an example from the rendermn.ini file that I edited:

```
.:@:${HFS}/houdini/ri_shaders/folder:${JOB}/shaders
```

And finally, you can always save and load your custom presets from Houdini’s RMan Output as another easy and sure method. After taking care of your shader search path, you should now be set to render with RenderMan.

Happy rendering.

Miscellaneous Information For Houdini-RenderMan

How do I render a polygonal model as a perfectly smoothed Subdivision Surface?

In the Object Editor, select the object that will be rendered as Subdivision surface. Go to its “Render” tab. Under “Geometry” field, you will see a pull-down menu list. Select “Polygons as Subdivision Surfaces” and voila!

After turning up the Sampling, my render still contain some strange artifacts. What gives?

In the Object Editor, go to the object’s Render tab and turn on the “Displace Bound.” Be careful not to set the bound too high, otherwise it may bring your computer to its knee. Keep it only high enough that the artifact will no longer present at any camera angle. In most case, the value of Displace Bound should not be higher than your displacement height.

*What is the advantage of using Pixar’s proprietary *.tx file format over a regular *.tif format?*

Since traditional image file format tend to be relatively large for a high quality scene. When you have many texture maps in one scene, they can hog memory. Using Pixar's proprietary MIP-map texture, image can be pre-filtered to save computation time and data access speed during rendering phase.

How do I render a curve in Houdini as RiCurve?

(Credit goes to John Coldrick of XYZ Animation, Louis Dunlevy of Framestore-CFC, and Nicolas Aithadi of The Moving Picture Company)

Houdini itself will output any polygonal curve as RiCurve (which will look like a spline curve). However, controlling its width is another thing to deal with. By default, the width specified by Houdini is set to "1". In order to change its width, append an Attribute Create SOP. Change the name field to "width" (this can be any name you want, but "width" is easier to understand); Class to "Primitive"; Type to "float"; the first Value field to, 0.1 or something small.

Afterward, append an Attribute SOP after the Attribute Create SOP. Go to RenderMan tab. Enter "width" to RiName field and "constantwidth" to RiType field. Specified that this is "Constant Float" in the drop-down list.

In the RMan Output, you should specify the "Cubic Basis" to "linear" (default) if you have 3 points or less; "cubic" if you have 4 points or more.

You can specify the environment variable for HOUDINI_RMAN_CURVE_BASIS and HOUDINI_RMAN_CURVE_STEP to control your RiCurve.

Why RenderMan doesn't render my closed cap of the NURBS tubes even when the Normals are pointing to a correct direction?

Since the cap is a closed-curve, not a surface, RenderMan does not support this feature. There are at least two ways to get around this. One way may be to manually move the vertices of the topmost of the NURBS tube to cap it. The second way may be to use Cap SOP.

What is the supported or unsupported RenderMan Interface Specification in Houdini's output?

For a list of supported and unsupported feature, please refer to the Houdini's Tutorial Guide (chapter 12 pg.219- 224 – as of version 5.0).

Recommended Resources

Website

Malcolm Kesson

RenderMan Professor at The Savannah College of Art and Design

<http://cmpa.ca.scad.edu/faculty/kesson/Ca301>

Malcolm Kesson's RenderMan Course Note
(available in PDF format – excellent place for beginners!)
<http://cmpa.ca.scad.edu/faculty/kesson/Ca301/bookindex.html>

RenderMan Interface Specification – online documentation
http://www.pixar.com/renderman/developers_corner/rispec/index.html

RenderMan Repository
www.renderman.org

RenderMania
www.rendermania.com

University of Hong Kong - CEC
<http://cade.scope.edu>

Books

“Advanced RenderMan: Creating CGI for Motion Pictures” by Anthony A. Apodaca & Larry Gritz

“The RenderMan Companion: A Programmer's Guide to Realistic Computer Graphics”
by Steve Upstill

Poh-Yee Alex Lim
July 2002, Revision 1