I Render Outputs

I COMMON OUTPUT OP PARAMETERS

I.I PARAMETERS

Though each output driver is tailored for a specific renderer, there are common parameters for each output driver.

RENDER BUTTON

Clicking this button begins the rendering process. Before starting the render, make certain that the parameters below have been set correctly.

CAMERA (NOT APPLICABLE FOR ALL OUTPUTS)

The pop-up menu associated with this parameter displays a list of available perspective objects in your scene (created in the Object Editor). Perspective objects are objects through which you can view and render your scene. Houdini considers lights and cameras to be perspective objects and allows you to render from their vantage points.

VISIBLE OBJECTS (NOT APPLICABLE FOR ALL OUTPUTS)

By default, this edit field is set to render all visible geometry objects (denoted by the * character) in the scene. You can exclude objects from the rendering process by entering the names of only those objects you want visible in the render in the edit field. A series of specific objects should be separated by commas, ***but no spaces***.

It is also possible to specify the visible objects using object groups. The group names are marked by using an at symbol (@) in the pattern. For example:

```
@group_name,arm*
```

Would make all the objects in the group *group_name* as well as all objects which match the pattern *arm** visible.

OUTPUT PICTURE

This option is disabled for some render types. You can specify a filename (which may include regular expressions). Alternately, you can send the output to one of the devices from the list of presets in the ▷ popup menu.

filename

Specify a filename or device to which the renderer should send the generated image.

If you enter a filename, the generated image will be saved to that file. Use can expressions in your filename as described in: *Expression Language > Expressions in File Names* p. 41.

Tip: Don't use spaces in filenames. Although both Unix and NT recognise their usage, a space in a filename will sometimes be interpreted as seperating the two parts of the filename into two seperate filenames, and cause endless trouble with getting your paths recognised correctly. Instead of a space, use a dash (-).

mplay window

Sends the output of the render to the *mPlay* program. These allow you to view a consequtive sequence of renders, and perform several standard image-viewing options while the scene renders. See: Please see: *COPs > 2D Viewport* p. 477 for complete *mPlay* info.

Some Shortcuts:

- You can redirect an in-progress *mantra* render by clicking the part of the image you want the rendering focused on.
- Click inside the *mplay* window with Ctrl 🖱 to get a pop-up menu of options, or type U to toggle display of the *mplay* interface.
- Subsequent renders appear in the same *mplay* window as new frames. Therefore, it is not necessary to close the *mplay* window between renders.

sequence of images

You want each frame to be saved in a separate file with a name that includes the frame number. The built-in variable for the current frame ($F) is very useful.

```
\$F.pic
```

The variable $F will be substituted with the current frame number so images will be saved in the current directory with names like *1.pic, 2.pic, 3.pic* etc.

```
\$JOB/Pic/\$F.pic
```

Images will be saved in a subdirectory of your job. The subdirectory called Pic must already exist. The image files will have names like *1.pic, 2.pic, 3.pic* etc. The variable $JOB can be set by using Houdini's job command.

```
\$JOB/Pic/\$CAM/\$F.pic
```

You many wish to render a sequence from a variety of camera or light positions. To keep everything understandable you can save each sequence in its own subdirectory. You can use Houdini's built-in variable for the name of the current camera (CAM). Images will be saved as, for example:

```
/jobs/Example/Pic/cam1/1.pic
/jobs/Example/Pic/light1/2.pic
```

padded file names

The text string in the edit field *($JOB/$F4.pic)* generates file names for your sequence that have four place-holders. For more on expressions in file names, please refer to the *Expression Language* section *Expressions in File Names* p. 41.

abekas start at 30

The string: *(a60:'$F+29'.yuv)* sends your image to the Abekas in YUV format and saves your animation starting on Abekas frame thirty. You can edit the command string in the edit field and alter the frame number and file format variables there.

IMAGE FORMAT (NOT APPLICABLE TO ALL OUTPUTS)

Specifies the format for images (Houdini .pic, TIFF, JPEG, Cineon, etc.). Generally, you will want it to Infer the format from the filename, but you can specify the format directly by picking one of the available options.

FRAME RANGE

This is the range of frames to render. If this option is turned on, the output driver will render multiple frames starting and finishing at the frames specified. Specify the Start, End, and Increment in the parameter below.

start / end / inc(rement)

These three edit fields control, the first frame number in the sequence to be rendered, the final frame number in the sequence, and the frame increment value. The frame increment can be used to skip frames, or can be a fractional number to render sub-frames.

squash and stretch an animation

Houdini uses floating point keyframing. Therefore there's no need to squash or stretch or use field rendering to cheat in lengthening an animation. Click on the ✙ button on Page 3/3 of the Playbar to bring up the Global Animation Parameters and turn off *Integer Frame Values*. Then you can enter decimal frames in the *Start/End/Inc* field. If you specify a non-integer frame increment, it will give you a stretched or squashed animation without having to adjust the channel data. For example if you specify a frame Inc(rement) of 0.8, you'll get an animation which is 20% longer.

You will want to be careful with the name of the file for the image you're generating. If you use the $F variable in the filename, you will probably have one frame writing over another. In this case it is better to use the $N variable in the filename which specifies the number of the image being rendered. For a complete discussion on filenames, see *Expression Lang > Filenames When Rendering Fields* p. 42.

INITIALIZE SIMULATION SOPS (NOT APPLICABLE FOR ALL OUTPUTS)

The geometry generated by simulation-type SOPs (i.e. the Particle SOP, Spring SOP, POP SOP) is dependent on the geometry from previous frames. For this reason, they can only be cooked in a forward direction. If the output frame being requested occurs at a time previous to the SOP's last cook time, the SOP will not recook. Enabling this button forces all of the simulation SOPs to be reset. They will be recooked from that SOP's start time, ensuring an accurate simulation.

GENERATE SCRIPT FILE (NOT APPLICABLE FOR ALL OUTPUTS)

Many renderers read their input from a script file. For example, RenderMan reads .rib files and *mantra* reads .ifd files. This option allows you to generate such a script file instead of calling a command directly to render. Some output drivers support binary scripts, while other renderers do not.

In the case of General Object Renderers, some renderers generate geometry instead of images (Inventor for example). These renderers will generate the geometry in the script file specified. It is possible to output both binary or ASCII Inventor files.

binary script

Checking the *Binary Script File* option specifies that the script file created by the render will be a binary one. That is, the information is represented in a tightly-coded short-hand rather than as ASCII characters you could read as text. This means the file contains the same information as a script file, but is more compact.

SCRIPT FILE (NOT APPLICABLE FOR ALL OUTPUTS)

This field takes the data generated during the render and creates a script file instead of producing a rendered image.

If the Output OP is a *mantra* driver, it generates an IFD (instantaneous frame description); if it is a RenderMan driver, it generates a RIB file.

RENDER IN BACKGROUND (NOT APPLICABLE FOR ALL OUTPUTS)

Selecting this option causes the rendering process to occur in the background, allowing you to continue working in Houdini while the render progresses. See also the *rps* command in: *Scripting* section > *rps* p. 116.

Note: This option is only valid for rendering single frames.

OVERRIDE DEFUALT RES / RESOLUTION

This parameter allows you to override the default resolution by specifying the *Resolution* manually. The default resolution may be determined in different ways internally. For example, a COP output operator will use the natural image resolution of the COP which it's rendering, while RenderMan would use the resolution specified in the camera object from which it is rendering.

PIXEL ASPECT

The pixel aspect ratio is the ratio of horizontal to vertical pixel size of the rendered image. The default value of 1 gives square pixels – suitable for viewing on your screen. The aspect ratio for an NTSC Abekas with 720 × 486 resolution and a 4 × 3 screen aspect ratio is:

$$\frac{4}{3} \times \frac{486}{720} = 0.9$$

PRE & POST FRAME RENDER SCRIPTS

Many output OPs have a pre-render, pre-frame, post-frame and post-render script available, they function as follows:

- The pre-render script is run one time before any rendering starts.
- The pre-frame script is run before each frame is rendered.
- The post-frame script is run after each frame is rendered.
- The post-render script is run one time after the entire render is completed.

When generating RIB, the post-include file for objects occurs inside the transform block if there is motion blur. This allows for correct motion blurring of ReadArchive data. You can get the behaviour of previous versions of Houdini (4.1 and prior) by setting the environment variable:

RMAN_INCLUDE_FIX

which causes the post-include to be included *after* the transform block.

2 3D TEXTURE GENERATOR OUTPUT OP

2.1 DESCRIPTION

This Output OP is used to drive the *image3d* program which generates 3D texture maps. The operator sets up all necessary options and then invokes the program.

2.2 PARAMETERS

RENDER

Begins the render.

FRAME RANGE

Allows you to specify a range of frames for rendering.

START/END/INC

Specifies the first & last frames and the increment. Note that when rendering a sequence of images with a fraction Inc, $N is the number of the frame rendered, $F is the nearest integer frame number, $FF is the floating point frame number

OUTPUT IMAGE

Name of the 3D texture to generate (typically, with the extension: i3d).

RESOLUTION

Resolution (in X, Y and Z) of the 3D texture

SHOP

The SHOP which controls the parameters for the texture generation.

VERBOSE

Generation statistics will be printed out

RENDER AS

Some SHOPs require particle or metaball geometry to run properly:

No geometry	No geometry is sent to i3dgen
Meta/Particle	Send particles/metaballs
Raw Point	Interpret points as particles

The clipped options allow you to override the bounding box of the geometry with your own.

INITIALIZE SIM

Dynamic simulations will be initialized

object/sop

Choose the object/SOP to specify the geometry

particle scale

An additional multiplier to particle scale.
This is primarily useful when sending down raw points.

DISPLACE BOUND

Enlarge the bounding box by this amount

MIN/MAX BOUNDS

The bounding box of the 3D texture

OVERSAMPLING

Anti-aliasing controls in texture generation.

VARIANCE

Threshold for anti-aliasing levels.

2.3 LOCAL VARIABLES

$N	The current frame of the range specified. This always starts at 1.
$NRENDER	Total number of frames being rendered.

3 ALFRED OUTPUT OP

3.1 INTRODUCTION

The Alfred output OP uses Pixar's program "Alfred" to handle render task management. The output driver can be used with mantra, RenderMan and other render outputs.

Alfred is shipped with Pixar's distrubution of RenderMan. It manages render tasks on a single machine or across a network. This output OP will generate an Alfred script and start Alfred with it. Alfred must be correctly configured.

When rendering using RenderMan, by default, netrman is used, so the local host must be running the nrmserver software.

When rendering using mantra, Alfred must be configured to know about mantra. Please see the readme file in: *$HH/scripts/alfred/* .

3.2 PARAMETERS

RENDER

Start an Alfred render task.

FRAME RANGE

Frame range to render.

OUTPUT DRIVER

The output driver which generates scripts for the Renderer. Typically this is a RenderMan or *mantra* output driver.

REMOTE SHELL

The shell to run on a remote machine. This is only important when using *netmantra*.

REMOTE HFS

Where the remote machine can find $HFS. It requires this knowledge to find the rscript application (in $HFS/bin). This is only used when running netmantra.

INIT. COMMANDS

Initialization commands. These are responsible for setting up the environment (only used when running netmantra).

ALFRED SCRIPT

When rendering from a driver which isn't a *mantra* or RenderMan driver, this script is called to generate the Alfred task (see $HH/scripts/alfred).

ALFRED COMMAND

This is the command used to start Alfred. Additional commandline options may be specified here.

TEMP. DIRECTORY

A location to store temporary files used in rendering.

3.3 CONFIGURING ALFRED FOR USE WITH MANTRA

When using Alfred to drive *mantra*, you must make some small configuration changes to the Alfred scripts:

1. Modify the lib/alfred/alfred.ini file to contain the following lines.

```
set alfLimitLocal(mantra) 1
set alfLimitGlobal(mantra) #licenses
```

2. Modify the *lib/alfred/alfred.shedule* file to allow machines to render using the 'mantra' token. Files in this directory:

alf_mantra.cmd	This script is used when Alfred is rendering using a Mantra output driver.
alf_rman.cmd	Script called to generate an Alfred task when the Alfred output driver is using a RenderMan output driver.
alf_other.cmd	Script called to generate an Alfred task when the Alfred output driver is using something else.

These commands must generate a valid Alfred task list when called. Typically, they will simply generate a simple task for a single frame. However, it is possible to modify these scripts so that they will generate multiple frames with a complicated task hierarchy. For example, it is possible to generate images from multiple output drivers which have dependency lists (i.e. render shadows, render reflection maps, render the final image). There are currently no examples of this.

Note: The Alfred Output OP will set the environment variable $ALF_TMP to be a path to a temporary directory (specified by the output driver).

3.4 HOW ALFRED WORKS

Alfred allows many renders to be processed simultaneously, and can use multiple CPU hosts over the network to render RenderMan, *mantra*, or other Rendering jobs.

The following steps are taken when a job is processed by Alfred:

- Alfred spools the .hip file to a temporary location.
- Each frame to be rendered calls a script to generate the frame using *hscript*, and generates an Alfred task * .
- At the conclusion of the Alfred script, the temporary files are automatically removed.

* If needed, these scripts can be modified to suit your needs. They are located in: *$HFShoudini/scripts/alfred/*.cmd* . The script is responsible for: i) Generating a RenderMan .rib or *mantra* .ifd file; ii) Generating an Alfred task which it does by means of "echo" statements.

IN DETAIL

When the Alfred output driver tries to render a frame, it first looks for an available machine. Once a host becomes available, it starts the remote render. It does this by following these steps:

1. The first thing that is done is that the local host opens a pipe to the remote host using rsh:

```
rsh host $COMMAND
```

The *netmantra* application allows Houdini to have proper process control over the network (i.e. to suspend or kill remote processes).

To have proper process control, the netmantra program starts up a process on the remote machine called *rscript*. This program must exist in *$HFS/bin* on the remote machine (with $HFS specified by the Remote $HFS parameter).

2. The *rsh* process then starts up the shell specified in the parameters on the remote machine. Typically, this should be left as */bin/csh -f* . However, it is possible to change the shell if desired.

3. Before the command from the output driver is started, the environment must be initialized correctly. Since *rsh* doesn’t export the current environment to remote machines, the environment has to be initialized for each separate host.

This is done using the *Initialization Commands*. Typically, it is important to set up the Houdini environment (if rendering using Mantra), or to set the path and environment for other commands (i.e. RenderMan).

To see which environment variables are passed through by *rsh*, try running the command *rsh localhost setenv* .

4. When the initialization commands have been executed, the command (as it appears in the selected output driver) is run on the remote host. The script file generated by the selected output driver is sent to the command on stdin. All parameters will have their variables expanded on the local host, unless they are protected by back-quotes (’).

NOTES

- When *netmantra* sends the IFD over to rscript which pipes it to the render command. The data in the IFD is passed verbatim to the remote host. This means that variables tucked in the IFD are expanded on the remote host.

- The $INIT stuff is basically used to set up any environment you want to have in your shell on the remote host. For example, on UNIX, you need to set the SESI_LMHOST variable and your path.
- When setting up for Alfred rendering, you should try to make all paths “network friendly” so that all referenced files (e.g. texture maps) are visible on the remote machine. It is also a good idea to have the images generated by the renderer go to a common NFS directory.

3.5 SAMPLE INITIALIZATION COMMANDS

- Set the display variable for interactive renders:

```
cd $HFS ; source .useit ;
setenv SESI_LMHOST $SESI_LMHOST ;
setenv DISPLAY $HOST:0
```

- Set the path for RenderMan

```
setenv PATH "/usr/local/prman/bin:$PATH"'
```

- To see display which host is currently receiving the render in the console, add this command somewhere in the initialization command string:

```
echo $NET_HOST
```

- Echo the rendering host to a log file (this assumes that the $LOG_FILE variable is defined on the localhost:

```
cd $HFS ; source .useit ;
setenv SESI_LMHOST $SESI_LMHOST ;
echo $NET_HOST >> $LOG_FILE
```

4 AMAZON OUTPUT OP

4.1 DESCRIPTION

Amazon is a 3D paint package created by Interactive Effects. This Output OP will generate a Tcl script and execute Amazon with the script file. Clicking *Execute* with the default parameters launches a new Amazon session. It passes in the geometry associated with the specified Object/SOP along with all texture maps, and bump maps that object/SOP references. Geometry sent to Amazon is automatically converted to polygonal data and stored in the Work Directory. Texture maps and bump maps are referenced directly by Amazon.

Amazon must be installed on your system in order for this driver to show up.

Notte: If your material references a texture map or bump map via a COP, it might not be exported to Amazon. In this case, load the map as a file instead.

4.2 PARAMETERS

EXECUTE

Generates the Amazon Tcl script and sends the script to Amazon.

FRAME

The frame to generate the script for. $F is the default.

OBJECT

The name of the object you wish to send to Amazon.

SOP

The name of the sop you wish to send to Amazon.

USE MULTIPLE RESOLUTIONS

Will use the resolution of bump maps and texture maps as they are.

NEW RESOLUTION

Resolution to scale the textures and bump maps when brought into Amazon if *Use Multiple Resolutions* is off.

WORK DIRECTORY

Location to store geometry and Tcl files created by the Amazon output OP.

TCL SCRIPT

This is the name of the Tcl script that is generated by Houdini and is sent to Amazon.

CONNECT TO EXISTING AMAZON

Attempts to connect to an existing executable on the host with the specified port number.

AMAZON COMMAND

The command used to start Amazon. Additional command line options can be specified here.

PORT

The port number to connect to.

HOST

The name of the host on which to run Amazon.

UPDATE HOUDINI

After the layers are saved from Amazon, this will update the textures and bump maps in Houdini.

5 CHANNEL OUTPUT OP

5.1 DESCRIPTION

Generates clip files for batch saving of channels from a CHOP.

5.2 PARAMETERS

Only parameters unique to this OP are discussed.
For the rest, see *Common Output OP Parameters* ***p. 717.***

CHOP NETWORK / NAME

Specify the CHOP containing the channels you want to output here.

6 COMPOSITE OUTPUT OP

6.1 DESCRIPTION

The Composite Output OP renders the image(s) produced in the Composite Editor and outputs them to the specified location.

Note: If you need to use old-style COPs (from Houdini 4), then please use *Composite Output OP (OLD)* p. 734. However – old COPs should be avoided if possible.

6.2 PARAMETERS

COP NETWORK / NAME

Select the COP you wish to render from by selecting from the pop-up menus. They display a list of the available networks of Composite Networks and COPs available in Composite Editor.

OVERRIDE DEFAULT RES / RESOLUTION

Allows you to override the default resolution by specifying the resolution manually. See *Override Defualt Res / Resolution* p. 720.

OUTPUT PICTURE

Specify a filename or device to which the renderer should send the generated image.

If you enter a filename, the generated image will be saved to that file. Use can expressions in your filename as described in: *Expression Language > Expressions in File Names* p. 41.

The pop-up menu to the right of the edit field displays a list of available destinations for the data created by the render. You can choose to send the rendered image(s) to a sequence of images, view the image on screen in an *mplay* window, or send the data to an Abekas.

COLOR / ALPHA PLANE

You can specify which image and alpha channel to use from the pop-up menu.

PLANE SCOPE

You can use regular expression to specify which channels to include in the output.

LUT FILE

For Cineon Output, you can specify a LUT file. See: Interface > *Cineon Page* p. 107.

OUTPUT GAMMA

Gamma alters a pixel's intensity in order to compensate for the unique colour characteristics of a given medium. With a Gamma of 1.0 *mantra* thinks a pixel with 50% coverage should be shaded with a 50% intensity. However, many recording devices do not respond to colour in a linear manner, so this is generally not the case in actual practice. How much a device diverges from this linear value is measured by a "gamma curve". The Gamma for video is 2.2, which differs from that of film (1.0). See *Gamma* p. 757 for a full description.

6.3 LOCAL VARIABLES

$N	The frame being rendered
$NRENDER	The total number of frames being rendered.

EXAMPLES

These are different than $F, and $NFRAMES. They are useful when the frame increment is not equal to 1. For example, if the frame increment is 0.5, using $F in the output filename will cause every other frame to be overwritten. By using $N, every filename will be unique. For example, consider the following Output Drivers:

frame range: 1 to 5 by 2

FileName	**$N.pic**	**$F.pic**	**$FF.pic**
First Render (frame 1)	1.pic	1.pic	1.pic
Second Render (frame 3)	2.pic	3.pic	3.pic
Third Render (frame 5)	3.pic	5.pic	5.pic

frame range: 1 to 3 by 0.5

FileName	$N.pic	$F.pic	$FF.pic
First Render (frame 1)	1.pic	1.pic	1.pic
Second Render (frame 1.5)	2.pic	2.pic	1.5.pic
Third Render (frame 2)	3.pic	2.pic	2.pic
Fourth Render (frame 2.5)	4.pic	2.pic	2.5.pic
Fifth Render (frame 3)	5.pic	3.pic	3.pic

7 COMPOSITE OUTPUT OP (OLD)

7.1 DESCRIPTION

The Composite Output OP renders the image(s) produced in the old Composite Editor and outputs them to the specified location. In general, you should not use the old COPs any longer – use COPs2 instead.

Note for Veterans: In Houdini 5.5, the old COPs, TOPs and Materials are superseded by SHOPs, VOPs and COP2. If you want the secret to backward compatibility features, try setting the environment variable: HOUDINI_COPTOPMAT

This will be altogether eliminated in version 6 – you have been warned.

7.2 PARAMETERS

COP NETWORK

The pop-up menu displays the networks of Composite Operations available in Houdini's Composite Editor.

COP NAME

This menu displays the individual COPs associated with the network you selected above.

OUTPUT PICTURE

Specify a filename or device to which the renderer should send the generated image.

If you enter a filename, the generated image will be saved to that file. Use can expressions in your filename as described in: *Expression Language > Expressions in File Names* p. 41.

The pop-up menu to the right of the edit field displays a list of available destinations for the data created by the render. You can choose to send the rendered image(s) to a sequence of images, view the image on screen in an *mplay* window, or send the data to an Abekas.

OVERRIDE DEFAULT RES / RESOLUTION

Allows you to override the default resolution by specifying the resolution manually. See *Override Defualt Res / Resolution* p. 720.

IMAGE FRACTION

This option lets you render your image at a fractional resolution relative to the resolution of the original image.

CINEON LUT FILE

For Cineon Output, you can specify a LUT file. See: Interface > *Cineon Page* p. 107.

cineon white point / film gamma

If you don't specify a LUT file, you can manually specify the White Point and Gamma here.

PRE-RENDER / FRAME SCRIPT

The *Pre-Render* / *Frame* script is included in the render before the actual code of the render or frame. You can use this to customise or modify rendering attributes.

Also see: *Pre & Post Frame Render Scripts* p. 721.

POST-RENDER / FRAME SCRIPT

Same as the Pre-Render / Frame Script, except it is appeneded to the render / frame.

8 GEOMETRY OUTPUT OP

8.1 DESCRIPTION

The Geometry Output operator outputs a sequence of geometry files (.geo or .bgeo) to disk. It takes an object and a SOP as well as an output field where you may specify a pattern. For example:

```
$F4.bgeo, $F.obj, $F4.dxf, etc.
```

If you want to output the scene in other geometry formats, use the *Object Scene Output OP* p. 774.

8.2 PARAMETERS

OUTPUT FILE

Enter a filename in this edit field to which you want to save the geometry of the specified frame-range. Using $F in the filename will output a different file for each frame. The $HIP uses the current project filename with a .geo/.bgeo extension. You can enter in a filename manually, or select from the menu. For example:

Entering this:	**Expands to:**
myFile$F.geo	myFile1.geo, myFile2.geo, myFile3.geo, ...

From the menu, you can select from the following short-cuts for filenames:

Menu Entry	**Short-cut Entered**
Binary Geometry Files	$HIP/$F.bgeo
Padded Binary Geo. Files	$HIP/$F4.bgeo
Geometry Files	$HIP/$F.geo
Padded Geometry Files	$HIP/$F4.geo

OBJECT / SOP NAME

Specify the object and SOP from which to output geometry from.

INITIALIZE SIMULATION SOPS

Forces all simulation SOPs to be reset before performing the render.

PRE-RENDER SCRIPT

Execute this script before any rendering.

PRE-FRAME SCRIPT

Execute this script before each frame.

POST-FRAME SCRIPT

Execute this script after each frame.

POST-RENDER SCRIPT

Execute this script after all rendering.

8.3 LOCAL VARIABLES

N	Frame being rendered
NRENDER	Total number of frames being rendered.

9 HOUDINI MOVIE (HMV) OUTPUT OP

9.1 DESCRIPTION

The Houdini Movie Output OP creates a Houdini Movie Format (.hmv) file.

Alternately, the output can be saved to disk as a script file to be rendered later.

Each HMV file typically contains many frames; this is in contrast to other Houdini image formats (such as .jpg) which contain only one image per file. The HMV file format has been designed to hold many thousands of full resolution images in a single file and for providing realtime playback when using disk-arrays such as the Ciprico.

HMV PLAYBACK SPEEDS

This depends on what you will be viewing the file on. You need the same byte order as the display. An O2 to its screen is RGB or RGBA. If it's going to be played or recorded out its video out port with *hmvrecord*, then the internal format within the .hmv is YUV.

You speed depends on:

- Use of XFS file system (yes is good).
- Through-put: IRIX (faster) vs NT boxen.
- Multi-disk drives and striping disks (two striped 10,000 RPM disks give realtime).
- Don't use .jpeg – hmv is for uncompressed movies.
- The above-noted internal formats.

9.2 PARAMETERS

COP NETWORK / COP NAME

Specify the COP Network and COP Name here from which to generate images.

OUTPUT PICTURE

The Output Picture parameter is the name of the HMV file to be created or updated. If the file is already in existence then the *Update Image File* button will be enabled. If this button is enabled, then the resolution, frame range and image format will be forced to be the same as that found within the file.

OVERRIDE DEFAULT RES / RESOLUTION

Allows you to override the default resolution by specifying the resolution manually. See *Override Defualt Res / Resolution* p. 720.

UPDATE IMAGE FILE

This is only enabled if the file already exists. It causes the movie file to over-write and update the existing movie file.

IMAGE FORMAT

This menu selects the format of how raster images are stored within an HMV file. The chosen format will affect the overall file size as well as the potential realtime playback speed.

RGBA	Stores each pixel as a 32 bit RGBA pixel. The ordering of the bytes is natural to the display byte ordering on MIPS style (i.e. SGI) workstations. For playback, these rasters can be displayed without image or byte order conversions. However, since each pixel contains an alpha (matte) channel, there is a certain amount of I/O that will be wasted during the playback process and therefore is not an optimal format for playback-only uses of HMV files. (Optimal for: Indy, O2, and Octane)
RGB	Stores each pixel as a 24 bit RGB pixel. The ordering of the bytes is not natural to the display byte ordering on MIPS workstations. For playback, these rasters require additional CPU processing to convert the byte ordering for MIPS workstations and therefore a slower playback speed will result.
BGR	Stores each pixel as a 24 bit BGR pixel. The ordering of bytes is natural to the display byte ordering on MIPS workstations. For playback, these rasters can be displayed without image or byte conversions. In addition, since there is no alpha (matte) channel, there is a reduced amount of I/O compared to the RGBA format. This is the fastest format. (Optimal for Indigo)
YUV	Stores the raster image in a YUV format. This is more space-economical than either the RGB or BGR formats. Currently, YUV requires additional processing (YUV to RGB) when displaying and this creates a slowdown in the playback speed. (Video layoff; best format to use with hmvrecord)

9.3 SEE ALSO

- *Compositing > Houdini Movie (HMV) Output OP* p. 738
- *Formats > Format of Houdini Movie (.hmv) Files* p. 279
- *Stand Alone > iplay - View Images* p. 382
- *Stand Alone > minfo – Display Movie Info* p. 411

10 MANTRA OUTPUT OP

10.1 DESCRIPTION

This output driver is used to generate scripts or render using Mantra. *mantra* reads a series of commands from standard input and creates a rendering from them. These commands describe a scene containing objects and light sources. *mantra* will render the scene specified and output an image. This Output OP is a front-end interface for setting up a rendering that uses *mantra*.

Note: The old version *mantra3* used TOPs and materials.
The regular version of *mantra* uses SHOPs. To create your own VEX-based materials (i.e. SHOPs) you should see the *SHOPs* section of the manual.

10.2 PARAMETERS – STANDARD PAGE

For parameters not listed here, see Common Output OP Parameters p. 717.

RENDER COMMAND

When not generating a script file, Houdini runs an external program (i.e. mantra) to actually render the image. The text entered here is the command line that is sent to drive this external program.

For details on the Render command options, refer to *Mantra Render Command Dialog* p. 748, which is invoked by clicking on the ✙ button beside the edit field.

remote rendering

When using *mantra*, the -H *(Remote Hosts)* option may be used to specify a list of hosts to render the image on. For example, the command line:

```
mantra -H host1,host1,host2,host3,host4
```

The above command will start two render processes on host1, and one render process on each of *host2*, *host3* and *host4*. Each of these render processes will contribute to rendering the single image.

SUPER SAMPLE

This parameter affects the quality of the anti-aliasing during the rendering process. Enter values here for the number of samples (rays sent per pixel) per pixel. The sampling is the product of the value in X times the value in Y. The default is 4 × 4, yielding 16 samples per pixel.

DECOUPLE RAY SAMPLE

Allows you to specify the *Ray Sampling* rate independently from the *Sampling Rate* for ray tracing.

FIELDS

A frame of NTSC or PAL consists of two interlaced fields. One field contains the odd scanlines of the frame and the other field contains the even scanlines and therefore they are termed the odd and even fields. NTSC displays at the rate of 30 frames per second which is 60 fields per second. PAL displays at the rate of 25 frames per second or 50 fields per second.

When rendering fields, Houdini will automatically render the two fields and combine them into a single image. Each field will be rendered at one half the frame increment specified in the frame range.

The ability to render and record fields is very important as it can greatly smooth the motion by doubling the apparent frame rate. The actual computation required to create one second of animation on fields is not much more than frames, because only half as many scanlines are needed for each render.

When rendering an image, there are some things which take the same amount of time whether your rendering fields or frames:

- Cooking of SOPs and geometry to be sent down to the renderer;
- Loading of the geometry by the renderer;

When rendering NURBS and certain other primitives, there's also a common overhead that the renderer encounters. As a result, the time it takes to render some scenes' fields may not always double as expected.

It is very important to know whether your record and playback device is “odd” or “even” dominant. That is, whether the odd scanlines are displayed before the even or vice versa. Most devices display the odd scanlines of a frame first, so that will be the correct setting for most devices, however, a small test can save a lot of grief.

If you select the wrong dominance for your device, the final result will look jittery unless you play it backwards. This is so because the motion within each frame will go from the dominant field to the non-dominant field. Choosing the wrong dominance will cause the motion to play backwards for the two fields within each frame yielding jittery-looking motion.

full frame

This is the default setting. Many devices only record frames, or it may be more efficient to record frames. Full Frame renders all scan lines for every Houdini frame. You can also generate a separate picture for each field. This is useful if the device to which you are recording will record fields. To record fields instead of frames, select one of the following instead.

even field dominance

Render even fields for Houdini's odd frame numbers and render odd fields for Houdini's even frame numbers.

odd field dominance

Render odd fields for Houdini's odd frame numbers and render even fields for Houdini's even frame numbers.

MOTION BLUR

This parameter allows you to specify the “default” behaviour for motion blur when rendering. If an object is set to “inherit” motion blur, it will be set to whatever the output driver says (i.e. the object will inherit the behaviour from the output driver). See *Ref > Objects > Geometry Object > Render page > Motion Blur* for details.

Production Tip: Leave most objects as “inherit behaviour”, then specify the motion blur type in the Output driver. This way you can have one render which does motion blur, and another that doesn’t.

DEPTH OF FIELD

Objects can be rendered in focus over a limited range of distances. Objects outside of the limited range of a finite-sized aperture are out of focus. This property is called depth of field. Computer images are generally in perfectly uniform focus; however, by specifying lens parameters in *mantra*, depth of field can be simulated. The focus and f-stop channels of the camera object are used to determine the depth of field.

Depth-of-field is not supported from orthographic cameras.
Depth-of-field cannot have scan line optimization.

JITTER

The Jitter parameter controls the amount of Jitter. Jitter parameter is used by *mantra* as part of its anti-aliasing algorithm. Jitter improves the anti-aliasing quality when things are moving (especially when rendering to video fields). However, for static elements, it is better to turn Jitter off; otherwise you may notice some noise around the edges (especially with type).

A pixel which is to be super-sampled is divided into sub-pixels. The number of sub-pixels is controlled by the */samplex* and */sampley* channels. A single ray is then sent through each sub-pixel within the pixel. When Jitter is 1.0 (the default) the ray can pierce any point within the sub-pixel. This provides good anti-aliasing for moving objects. However, since jitter is random (there is no frame to frame consistency) the edges of static objects may appear to sparkle. You can solve this problem by setting Jitter Scale to 0 so that the ray will always pass through the center of the sub-pixel.

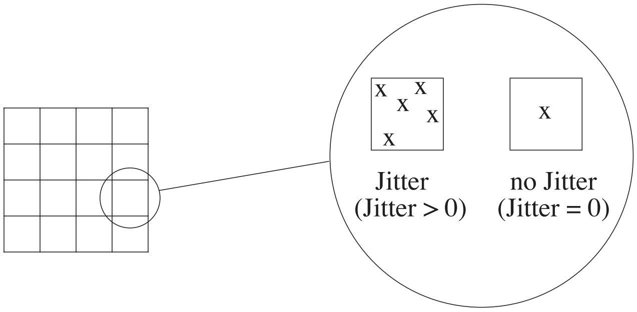
Jittering of ray directions in *mantra* depends on:

a) Number of Samples = 4 × 4 so the highest number of distinct values
you would expect to see is 16.

b) How Near to horizontal/vertical is the Edge you are looking at?

pixel subdivision with jitter

A pixel is subdivided as follows:



sub-divided pixel valid rays through a sub-pixel

In the case of *Jitter Sample*, the ray could go through any of the X positions. With Jitter = 0 (off), the ray will always go through X.

If you picture a diagonal line through the pixels, what you'll probably find is that when you move from one pixel to the next, you won't add a single sub-pixel to the pixel total.

Try drawing a line on graph paper and seeing how many sub-pixels are added across pixels. This usually results in fewer than 16 levels on a single edge. By increasing the *Sampling*, you get more samples per pixel, and thus more different levels. Mantra will allow up to 8 × 8 super-sampling which gives 64 levels per edge.

DITHER

Dither determines the noise added to an image prior to creating the final output pixels. Dithering is the addition of low-level noise to colors before they are converted from high-precision floating point numbers into 8 bit (0-255) integers. You can avoid *Mach Banding* in an image containing slight luminance changes by adding approximately 0.004 (1/255) noise to them.

Normally, dithering is done in HSV (hue, saturation, value) color space in order to provide better dithering for highly saturated objects.

GAMMA

Gamma alters a pixel's intensity in order to compensate for the unique colour characteristics of a given medium. With a Gamma of 1.0 *mantra3* thinks a pixel with 50% coverage should be shaded with a 50% intensity. However, many recording devices do not respond to colour in a linear manner, so this is generally not the case in actual practice. How much a device diverges from this linear value is measured by a "gamma curve". The Gamma for video is 2.2, which differs from that of film (1.0).

In *mantra3*, after a pixel is rendered, its floating point color value is modified by the Gamma value prior to creating the final output 8-bit RGB channels (3×8=24 bits) in the output image.

The default Gamma is 1.0. Larger values brighten the image (especially within dark regions) and values between 0 and 1 darken the image. In all cases full black remains full black and full white remains full white.

typical gamma settings

Film	1.0
SGI Standard Gamma	1.7
Video	2.2

10.3 PARAMETERS – SPECIFIC PAGE

WHITE POINT

An overall scale applied to colour after filtering

FILTER

Pixel filtering controls. You can select from the presets in the ▷ menu.

SHADOW MAPS

Controls which lights will auto-generate Z-depth maps.

REFLECT MAPS

Controls which objects will generate Reflection maps.

INLINE CONVERTED MATERIALS

When converting materials to VEX, include the VEX code in the IFD.

OPACITY LIMIT

Beyond this threshold, transparent objects will be considered to be opaque. This speeds up rendering of scenes containing items of minimal transparency.

Setting the *Opacity Limit* to something smaller than 1 can improve rendering performance for scenes with a lot of transparency (not refraction). When shading transparent surfaces, if the combined opacity exceedes the specified limit, no further shading will be done.

POLY OPTIMIZE

Allows you to set Faster or Higher quality polygon rendering. You can trade off rendering quality of polygons vs. speed of rendering. This only affects polygons which are ‘large’ in screen space.

NULL SURFACE

Optimizes shadow map generation with null shaders.

TRANSFORMS

Combined transforms give better precision, but in some cases, it's useful to separate the camera transforms.

VEX PROFILING

Generates statistics on shading computations, so you can track down and optimize your VEX rendering processes.

There are options to profile VEX code in both Houdini and Mantra. In Houdini, the "vexprofile" command can be used to turn profiling of VEX on or off. There’s an option in the output driver in mantra to turn VEX profiling on or off.

meanings of vex profile code

Profiling of VEX code will show statistics on the execution of VEX.
Profiling generates reports which look something like:

```
Function Excl. Incl. Calls Instructions Instr.
Local Secs Secs Per Call Storage -----

plastic 0.72 0.81 388,527 5,827,905 15.00 9,256
asad_light 0.07 0.07 388,527 3,496,743 9.00 3,096
ambient 0.01 0.01 388,527 777,054 2.00 16
```

The meanings of the headings are:

Excl. Secs	The number of seconds spent executing that particular VEX function.
Incl. Secs	The total time spent executing the VEX function, including all other VEX functions called from within the function. For example, in the table above, the "plastic" shader has an Exclusive time of 0.72 seconds, but an inclusive time of 0.81 seconds. This is because the shader calls the light shaders. The inclusive time includes the time spent in the light shaders.
Calls	This is the number of times the VEX function was evaluated
Instructions	This is the number of instructions executed by the VEX function
Instr. Per Call	This is the average number of instructions per evaluation of the VEX function
Local Storage	This is the amount of RAM required to store local variables.

RecurseLevel	The maximum levels of recursion for the VEX code (only possible in mantra)

The profiler is also able to check for bad results of operations. These bad results often cause white pixels in mantra or geometry to fly off to infinity in SOPs.

Because extra work is done to profile VEX code performance is adversly affected.

10.4 PARAMETERS – DEEP RASTER PAGE

AUXILIARY PROFILES

Controls for rendering deep rasters. If the output image format consists of Houdini .pic files, each layer of the deep raster will be embedded in a single output image as specified here.

Note: By default, floating point data is written to the image file. If the output format does not support floating point images, the *-b* option should be used.

VEX VARIABLE / AUXILIARY FILE

You can output multiple images simultaneously from mantra/prman. You must specify a filename, an export Variable and its Type. You must also choose the output data type.

Tip: Up to 6 auxiliary files to be specified. However, it is possible to add additional output images using the pre-include/post-include statements for cameras.

AUXILLIARY CHANNEL OUTPUT IN MANTRA

Rendering multiple images simultaneously in *mantra* is done through the IFD command:

```
ray_aux [options] filename vex_type vex_variable
```

where the *options* are: *-b channel_type* (similar to the -b option on the mantra command line) which will cause the filename to be created and the value of the VEX variable will be written into the file.

Only variables defined in surface shaders can be output. To be output, the variable must be declared as an export parameter, or it must be one of the global variables:

```
P, s, t, dPds, dPdt, N, Ng, Cf, Of, Af, I, Eye
```

The correct VEX type must also be specified. Valid VEX types are *float*, *vector* or *point*. There is currently no support for integer, string or matrix types.

10.5 PARAMETERS – SCRIPTS PAGE

PRE-RENDER SCRIPT

Execute this script before any rendering.

PRE-FRAME SCRIPT

Execute this script before each frame.

POST-FRAME SCRIPT

Execute this script after each frame.

POST-RENDER SCRIPT

Execute this script after all rendering.

10.6 SPECIAL NOTES

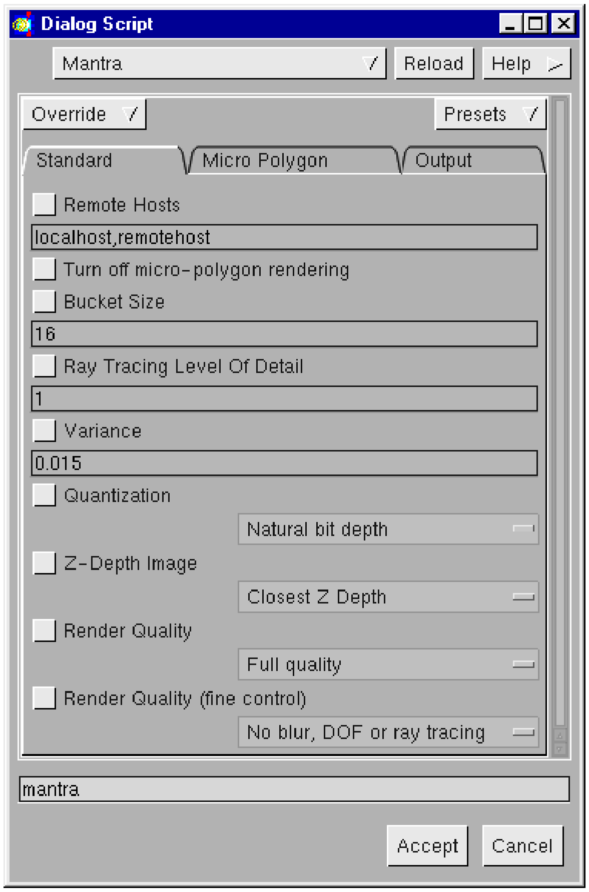
CURVE PRIMITIVES

With *mantra*, any open polygon, Bezier curve or NURBS curve will be rendered as a curve primitive. There are two special attributes:

width	This can be per primitive, point or vertex. It defines the width of the curves for rendering The width is in Houdini units (not pixels).The default width is 0.1 .
orient	A vector attribute (i.e. you can rename "N") to define the perpendicular vector to the curve surface. This allows you to orient the curves however you want. If this attribute doesn't exist, the curves will be oriented so that their normal points roughly in the direction of the camera.

A good example to look at would be the *hair* shader.

11 MANTRA RENDER COMMAND DIALOG



☐ Remote Hosts
localhost,remotehost
☐ Turn off micro-polygon rendering
☐ Bucket Size
16
☐ Ray Tracing Level Of Detail
1
☐ Variance
0.015
☐ Quantization
Natural bit depth
☐ Z-Depth Image
Closest Z Depth
☐ Render Quality
Full quality
☐ Render Quality (fine control)
No blur, DOF or ray tracing

mantra

11.1 PARAMETERS – STANDARD PAGE

Mantra uses VEX for all Shading (the options are similar to old Mantra3).

REMOTE HOSTS (-H)

Mantra can also render using remote hosts. Specify a comma separated list of remote hostnames here (or use the -H option on the commandline). Each remote host must be running *hserver* for this option to work. If multi-processing is specified, network rendering will be used to start multiple renders on the same host.

TURN OFF MICRO-POLYGON RENDERING (-R)

Disables micro-polygons. See *MicroPolygon Rendering* p. 784 for details.

BUCKET SIZE (-B)

mantra breaks up the entire image into small tiles and renders each of these tiles individually. When a small tile size is specified, *mantra* will use less memory, but has to do more work. This option can be used to specify the size of a tile. When rendering interactively, *mantra* will force the bucket size to 16 × 16 pixels. This will slow down the render, but provide better interactivity. This can be overridden by the -B command line option. See *Bucket Size (-b)* p. 761 for more info.

RAY TRACING LEVEL OF DETAIL (-L)

The -L option is a global override for Level of detail. When rendering patch surfaces, in some cases, mantra3 will sub-divide the surface based on a relative level of detail. Surfaces close to the camera will be more finely sub-divided than surfaces further away. The level of detail is usually specified in the Object Editor (see *Level of Detail* p. 288 in the *Objects* section). To speed up renders, the -L option can be used with a fractional value. For example -L 0.5 will render at half the level of detail for all objects in the scene. A level of detail of 0.1 will speed up the rendering considerably.

VARIANCE (-V)

This option (-v option on the command line to mantra) can be used to specify a different anti-aliasing mechanism. Variance Anti-aliasing takes an argument which specifies the acceptable tolerance in color contrast between pixels. If the tolerance is not met, anti-aliasing is performed. This will anti-alias areas of the image which tend to have aliasing (i.e. edges of primitives, texture maps). Increasing the value of the variance (maximum of 1) results in faster renderings. Decreasing the value (minimum of 0) produces higher quality images.

See *Variance Anti-Alias (-v)* p. 761 for more info.

QUANTIZATION (-B)

Quantization (-b) is when pixel component values are transformed from real (i.e. floating point in the range of 0-1) values to integer values. Use this menu to specify if values should be quantized to: *8 bit*, *16 bit*, *Natural bit-depth*, or remain as *Floating-point channel* values. The natural depth is dependent on the target image format.

Z-DEPTH IMAGE (-Z)

The options for this parameter allow you to choose between *Closest* and *Average* depths. *mantra* defaults to *Average*, meaning that it uses the average of the distances of the first two surfaces hit by the ray from the camera. *Closest* means that the first surface hit by the ray coming from the camera determines the depth.

For more information, see *Z-Depth Image* p. 762.

PATCH CRACKS (-C)

Attempts to patch cracks (-c) by stitching adjacent portions together.

RENDER QUALITY (-Q)

Determines the quality (-q) of the render. With it, you can set rendering to be: Full Quality (= 10), disable motion blur (= or less than 8), depth of field (= or less than 7), and ray-tracing (= or less than 4). Each of these increases the speed at the cost of render quality.

When ray-tracing is turned off, shadow shaders are still run on lights, but if the shadow shader makes ray-tracing calls (i.e. fastshadow()), then these calls will still 'fail' in that no rays will be cast.

RENDER QUALITY (FINE CONTROL -Q)

Allows you to set the render quality in relation to *Motion Blur*, *Depth of Field*, and *Ray Tracing* at a finer level.

11.2 PARAMATERS – MICROPOLYGON PAGE

SHADING QUALITY (-S)

The *Shading Quality* (-s) parameter sets the quality with which the lighting model (i.e. material) is applied to the object. This means the material applied to the object can have its sharpness increased or decreased depending on how close your camera is to that object (enter an expression that changes the Shading Quality based on distance from the camera).

Decreasing this value speeds up the rendering process, but also decreases the sharpness on which that material is applied. Increasing the Shading Quality improves the material's texture map, bumps, and dent quality, but also increases the rendering time.

The most important thing to remember is that *Shading Quality* applies to the material in general. If there are textures or bumps, dents or erodes in the material, the Shading Quality value has a tremendous impact on the success of the render.

You therefore want to adjust the *Shading Quality* for a couple reasons:

- Texture maps with text/font images or texture maps used as decals need a higher level of Shading Quality to improve the quality of the material and avoid aliasing in the textures.
- Texture maps with dents, bumps, or erodes with a high frequency need a higher Shading Quality setting in order to avoid flickering.

If the *Shading Quality* is not set high enough you will get noise or aliasing artifacts in your texture map, no matter what your Super Sampling is set to. It can be thought of as Super Sampling for your materials (lighting models).

note – on level of detail vs shading quality

If you're using regular *mantra*, it has two modes of rendering (*mantra3* has equivalent modes). Micropolygon uses *Shading Quality* to adjust the size of micropolygons. Ray-Tracing uses the LOD as a similar control (i.e. they will sub-divide primitives based on the LOD). Thus, you can think of *LOD* as the ray-tracing equivalent of *Shading Quality.* Basically, you can have a lower LOD on a surface which means it will be coarser in ray-tracing, while keeping the primary rays at a high quality.

range of values for shading quality

0.01	Extremely low
5	Extremely high (increases render time about 25 times!)
1 to 2	Nominal.

Note: The *Shading Quality* parameter is available both per-object *(object > Render page > Shading Quality)*, and Globally *(Output Editor > mantra > Render Command* ✙ *> Micro Polygon sub-page > Shading Quality)*. These two values are multiplied to determine the final *Shading Quality* used in the render.

MICRO POLYGON CACHE SIZE (-G)

This option (-G) allows you to specify a cache size for the micro-polygon storage. By default, this is set to 512. Specifying lower numbers (minimum value 8) causes mantra to require less memory. However, rendering times will increase since mantra will be required to do more work.

MICRO POLYGON SPLITS (-S)

If there is a single primitive which spans a large area, but only a portion is visible on the screen (i.e. a single polygon ground plane stretching to infinity), it’s possible that some shading artifacts will appear. These will manifest themselves as large blurry areas on the surface. This occurs when mantra is unable to refine a surface into small enough micro polygons. It is possible to increase the maximum threshold of splits by changing this parameter.

11.3 PARAMETERS – OUTPUT PAGE

VERBOSE

Enabling this option provides you with various feedback about what is going on during the render, including: *Render Time and Memory*, *Loading Information*, and *Render Progress*.

OUTPUT / APPEND TO FILE

Directs the messages output by the *Verbose* option to the file specified here. *Append to File* does the same thing, but appends the file instead of creating a new file.

11.4 MANTRA COMMAND LINE OPTIONS

mantra can be run from a shell script. The above options can be specified form the command line as well as the Render Dialog Script. The options are listed below:

```
Usage: mantra [options] [outputimage]
  -r                            Force mantra into ray tracing mode
                                (no micro-polygon).
  -q                            Set render quality (0 to 10):
                                   Below 9 turns off motion blur;
                                   Below 8 turns off depth of field;
                                   Below 5 turns off ray tracing.
  -s val                        Specify shading rate multiplier
  -L val                        Global level of detail factor
                                (ray-traced shading rate).
  -C val                        Specify ray tracing grid cache size (default 128).
  -G val                        Specify micro-polygon cache size (default 4096).
  -M val                        Specify ray mesh cache size (default 512).
  -S val                        Specify micro-polygon max. splits (default 16).
  -v var                        Specify variance anti-aliasing threshold.
  -A                            Turn off anti-aliasing (default anti-aliasing on).
  -B val                        Specify bucket size for render (default 64).
  -J val                        Set the jitter scale (default 1).
  Image Options:
  -i                            Render interactively.
  -w val, -h val                Specify width & height of output image.
  -b val                        Specify bit-depth of output image
                                (color image only).
                                  0, any - Use the ‘natural’ bit depth;
                                  8, byte, char - Generate 8 bits per color channel
                                 16, short      - Generate 16 bits per color channel
                                 32, float - Floating point data per color channel
                                 The natural depth is dependent on the target
                                  image format.
  -E                            Render image as an even field.
  -O                            Render image as an odd field.
  -z                            Render averaged z-depth image.
  -Z                            Render z-depth image.
  Control Options:
  -H hostlist                   Specify network hosts to render the image on.
                                The list should be a comma separated list of hosts
                                (i.e. -H chili,cayenne).
  -n val                        Set the number of processes (default 1).
                                This option is implemented as:
                                -H localhost,localhost...
  -o file                       Send verbose output to the specified file.
  -p file                       Append verbose output to the specified file.
  -V val                        Set verbose level.
```

`-y <addr>`	Used internally by mantra for network rendering.
`-F`	Use fast interactive device.

12 MANTRA3 OUTPUT OP

12.1 DESCRIPTION

Note:** mantra3* ***is kept only for backwards compatibility.
You should use mantra for any new projects.

This Output OP is a front-end interface for setting up a rendering that uses *mantra*3 which reads a series of IFD commands from standard input. These commands describe a scene containing objects and light sources. *mantra* will render the scene specified and output an image.

12.2 PARAMETERS

RENDER COMMAND

When not generating a script file, Houdini runs an external program (i.e. mantra3) to actually render the image. The text entered here is the command line that is sent to drive this external program.

The default string (mantra3 -a -v 0.05) means that the *Render Command* used will use the *mantra3* rendering program with Variance Anti-aliasing (-v) and Micropolygon rendering (-a) enabled.

For details on the Render command options, refer to *Mantra3 Render Command Dialog* p. 759, which is invoked by clicking on the ✙ button beside the edit field.

production tip – compressing the output

If you are generating very large RIB or IFD files, with UNIX you can *gzip* the output by changing the *Render Command* command from “render” or “mantra3” to:

```
gzip [-9] > file.rib.gz
gzip [-9] > file.ifd.gz
```

It is also possible to render directly from a compressed file:

```
% gzcat file.rib.gz | render
% gzcat file.ifd.gz | mantra3 [mantra_options]
```

In RenderMan 3.7:

```
% gzcat file.rib.gz | render
```

can simply be:

```
% render file.rib.gz
```

reading an ifd you've output to a standalone mantra

If you've opted to created an IFD (instantaneous frame description) instead of rendering directly out of Houdini, you will subsequently need to pipe the output of this IFD file into the standalone *mantra* application. To do this, you should enter something like the following in a UNIX shell:

```
mantra3 -a -v x.xx < file.ifd
```

The re-direct symbol (<) is used to pass the IFD to *mantra3*.

SUPER SAMPLE

This parameter affects the quality of the anti-aliasing during the rendering process. Enter values here for the number of samples (rays sent per pixel) per pixel. The sampling is the product of the value in X times the value in Y. The default is 4 × 4, yielding 16 samples per pixel.

PIXEL ASPECT

The pixel aspect ratio is the ratio of horizontal to vertical pixel size of the rendered image. The default value of 1 gives square pixels – suitable for viewing on your screen. The aspect ratio for an NTSC Abekas with 720 × 486 resolution and a 4 × 3 screen aspect ratio is:

$$\frac{4}{3} \times \frac{486}{720} = 0.9$$

FIELDS

A frame of NTSC or PAL consists of two interlaced fields. One field contains the odd scanlines of the frame and the other field contains the even scanlines and therefore they are termed the odd and even fields. NTSC displays at the rate of 30 frames per second which is 60 fields per second. PAL displays at the rate of 25 frames per second or 50 fields per second.

When rendering fields, Houdini will automatically render the two fields and combine them into a single image. Each field will be rendered at one half the frame increment specified in the frame range.

The ability to render and record fields is very important as it can greatly smooth the motion by doubling the apparent frame rate. The actual computation required to create one second of animation on fields is not much more than frames, because only half as many scanlines are needed for each render.

When rendering an image, there are some things which take the same amount of time whether your rendering fields or frames:

- Cooking of SOPs and geometry to be sent down to the renderer;
- Loading of the geometry by the renderer;

When rendering NURBS and certain other primitives, there's also a common overhead that the renderer encounters. As a result, the time it takes to render some scenes' fields may not always double as expected.

It is very important to know whether your record and playback device is “odd” or “even” dominant. That is, whether the odd scanlines are displayed before the even or vice versa. Most devices display the odd scanlines of a frame first, so that will be the correct setting for most devices, however, a small test can save a lot of grief.

If you select the wrong dominance for your device, the final result will look jittery unless you play it backwards. This is so because the motion within each frame will go from the dominant field to the non-dominant field. Choosing the wrong dominance will cause the motion to play backwards for the two fields within each frame yielding jittery-looking motion.

full frame

This is the default setting. Many devices only record frames, or it may be more efficient to record frames. Full Frame renders all scan lines for every Houdini frame. You can also generate a separate picture for each field. This is useful if the device to which you are recording will record fields. To record fields instead of frames, select one of the following instead.

even field dominance

Render even fields for Houdini’s odd frame numbers and render odd fields for Houdini’s even frame numbers.

odd field dominance

Render odd fields for Houdini’s odd frame numbers and render even fields for Houdini’s even frame numbers.

MOTION BLUR

This parameter allows you to specify the “default” behaviour for motion blur when rendering. If an object is set to “inherit” motion blur, it will be set to whatever the output driver says (i.e. the object will inherit the behaviour from the output driver). See *Ref > Objects > Geometry Object > Render page > Motion Blur* for details.

Production Tip: Leave most objects as “inherit behaviour”, then specify the motion blur type in the Output driver. This way you can have one render which does motion blur, and another that doesn’t.

DEPTH OF FIELD

Objects can be rendered in focus over a limited range of distances. Objects outside of the limited range of a finite-sized aperture are out of focus. This property is called depth of field. Computer images are generally in perfectly uniform focus; however, by specifying lens parameters in *mantra3*, depth of field can be simulated. The focus and f-stop channels of the camera object are used to determine the depth of field.

Depth-of-field is not supported from orthographic cameras.
Depth-of-field cannot have scan line optimization.

JITTER

The Jitter parameter limits the amount of jitter. mantra makes use of jitter as part of its anti-aliasing method. A pixel which is to be super-sampled is divided into sub-pixels. The number of sub-pixels is controlled by the /*samplex* and /*sampley* channels. See *Jitter* p. 742 for a full description.

DITHER

Dither determines the noise added to an image prior to creating the final output pixels. Dithering is the addition of low-level noise to colors before they are converted from high-precision floating point numbers into 8 bit (0-255) integers. You can avoid *Mach Banding* in an image containing slight luminance changes by adding approximately 0.004 (1/255) noise to them.

Normally, dithering is done in HSV (hue, saturation, value) color space in order to provide better dithering for highly saturated objects.

GAMMA

Gamma alters a pixel's intensity in order to compensate for the unique colour characteristics of a given medium. With a Gamma of 1.0 *mantra3* thinks a pixel with 50% coverage should be shaded with a 50% intensity. However, many recording devices do not respond to colour in a linear manner, so this is generally not the case in actual practice. How much a device diverges from this linear value is measured by a "gamma curve". The Gamma for video is 2.2, which differs from that of film (1.0).

In *mantra3*, after a pixel is rendered, its floating point color value is modified by the Gamma value prior to creating the final output 8-bit RGB channels (3×8=24 bits) in the output image.

The default Gamma is 1.0. Larger values brighten the image (especially within dark regions) and values between 0 and 1 darken the image. In all cases full black remains full black and full white remains full white.

typical gamma settings

Film	1.0
SGI Standard Gamma	1.7
Video	2.2

12.3 PARAMETERS – SPECIFIC PAGE

WHITE POINT (RAY_WHITEPOINT)

The White Point allows quick brightening or darkening of the final output. When *mantra3* takes floating point colors (typically 0 - 1), and converts them to integer channels (0-255 or 0-65535), the White Point determines what a floating point value of 1 should be scaled to.

If the white-point value is set to 0.25, the floating point colors will be scaled by 0.25 before being converted to integer values. In this case, the whole image will get darker. In fact, full white (i.e. 1, 1, 1) will be mapped to (0.25, 0.25, 0.25) before conversion to integer. With eight bit channels, this will result in values of 63, 63, 63 instead of 255, 255, 255.

By lowering the white-point, colors brighter than one can be stored in the image. This can be used in compositing, or recording to film to get colors brighter than 1 interpreted correctly.

FILTER (RAY_FILTER)

The Pixel Filter parameter specifies a filter to apply to pixels before quantizing them to integer values. The standard filter types are available for pixel filtering.

Because the filtering occurs before quantization, this will improve anti-aliasing, especially where colors are larger than the white-point setting.

Warning: There is a current limitation that pixel filter width cannot be larger than half the bucket size for rendering. The filter size will be clamped to half the bucket size (meaning that with large filter sizes, renderings may be different when rendered using different bucket sizes).

The default filter is *gaussian 1.5 1.5* which filters samples using a gaussian filter of 1.5 pixel radius.

12.4 PARAMETERS – SCRIPTS PAGE

PRE-RENDER / FRAME SCRIPT

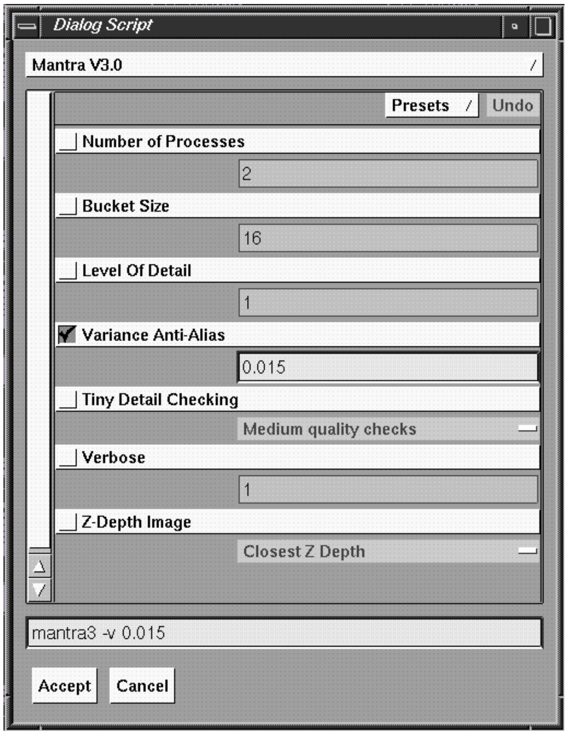
The *Pre-Render* / *Frame* script is included in the render before the actual code of the render or frame. You can use this to customise or modify rendering attributes.

Also see: *Pre & Post Frame Render Scripts* p. 721.

POST-RENDER / FRAME SCRIPT

Same as the Pre-Render / Frame Script, except it is appeneded to the render / frame.

13 MANTRA3 RENDER COMMAND DIALOG



Mantra V3.0

☐ Number of Processes
2

☐ Bucket Size
16

☐ Level Of Detail
1

☑ Variance Anti-Alias
0.015

☐ Tiny Detail Checking
Medium quality checks

☐ Verbose
1

☐ Z-Depth Image
Closest Z Depth

```
mantra3 -v 0.015
```

Note: The regular *mantra* renderer using VEX shaders (SHOPs) is a seperate renderer also included with Houdini. To use it, use a *mantra* output instead of a *mantra3* output. The regular *mantra* is faster and more robust than *mantra3*; while *mantra3* is more compatible with old project files and materials made using TOPs.

THE THREE TYPES OF MANTRA3 RENDERING

mantra3 contains an option to improve both rendering quality and rendering speed called micro-polygon rendering which is the default. This means there are three basic ways that you can get *mantra3* to render scenes; they are:

- Scan Line Rendering (with no option specified)
- Variance anti-aliased Rendering (with -v option > no Motion Blur but fast)
- Micro-polygon Rendering (with -a option > no Metaballs, but fastest)

See See *MicroPolygon Rendering* p. 784 for details.

MICRO-POLYGON RENDERING (-A)

Enabling this option (-a) which is the default, allows you to use micro-polygon rendering which is significantly faster than using either scanline rendering or the variance anti-aliasing method. See *MicroPolygon Rendering* p. 784 for details.

13.1 MANTRA3 – STANDARD OPTIONS PAGE

NUMBER OF PROCESSES (-N)

If your workstation only has only one CPU, you can ignore the following discussion.

If your computer is equipped with more than one processor the renderer can make full use of your system's power. *mantra3* can use multiple CPU's to render a single image. By using the -n option, the number of processes can be specified. *mantra3* permits a single image to be rendered on multiple processors. The *#processors* text string of the Render Options Dialog determines how many processors will be used to render your images. To achieve maximum rendering speed for simple scenes set this text string to the number of processors you have. However, if you wish to continue working in Houdini while the pictures are being rendered, you should set this text string to something less than the maximum. This will avoid bogging down the system too much.

Complex scenes with large amounts of geometry and/or many texture maps take longer to render. It is not always true that more processors mean faster rendering. You may be limited by the amount of memory in your computer. Before rendering a long sequence you should do a time test. Render a single frame with one processor. Render the same frame with two processors, with three, four, etc. Then compare the times. There are also several UNIX tools to handle shared memory management. *ipcs* (Inter Process Communication Status) will display all shared memory blocks, semaphores, etc. which are currently active. These can be deleted using *ipcrm*.

The limit on the shared memory segments may be increased by changing the definition for SHMMNI in the file */usr/sysgen/master.d/shm*, then recompiling UNIX. This should be attempted only by knowledgeable users.

This number will be clamped to the maximum number of configured CPUs of your system. The -N option can be used to force *mantra3* to render with a higher number of processes than the system has CPUs.

text command alternative

The *-n* option permits a single image to be rendered on multiple processors:

```
Houdini-> mantra3 -n 3
```

Specifies three processes.

flushing left-over mantra processes

When *mantra3* renders with multiple processes, all processes share the same data space. No shared memory will be allocated since all processes share the same memory. However, to function correctly, semaphores must be used. In some cases when *mantra3* is killed unexpectedly, these semaphores may linger. The following shell script can be used to destroy these semaphores:

```
#!/bin/csh -f
foreach sem ( ‘ipcs -s‘ )
ipcrm -s $sem >& /dev/null
end
```

BUCKET SIZE (-B)

While you are in the process of rendering with *mantra3*, you can direct the render to focus on a specific area of the image by clicking on unrendered portions with your mouse. *mantra3* will then concentrate on the area in which you clicked. This is called interactive rendering. It takes advantage of the fact that *mantra3* is a tile-based renderer. This means that it breaks up the entire image into small tiles and renders each of these tiles individually. When rendering interactively, *mantra3* will force the bucket size to 16 × 16 pixels. This slows down the render, but provides better interactivity. This can be overridden by the -B command line option. When a small tile size is specified, *mantra3* will use less memory, but has to do more work.

The *Bucket Size* option allows you to specify the size, in pixels, of the area being rendered at any one time. Altering this parameter is useful for test renders where you want to check the results in one area only.

Increasing the bucket size yields a faster rendering time. Decreasing the bucket size produces better feedback from *mantra3's* interactive rendering feature.

Three different rendering methods can be controlled:

- If you move the cursor so that it is outside the image window, *mantra3* will simply render the image moving clockwise from the outside in.
- If you click with left mouse (🖱) in the render window, *mantra3* will focus its rendering activities to the region that was clicked.

When using interactive rendering with multiple processors, only scanlines from the first processor will be drawn to the screen. The other processors will render complete scanlines from the bottom up. Moving the window displays all processors' completed pixels. Clicking with the middle mouse (🖱) will also display all processors' completed pixels.

LEVEL OF DETAIL (-L)

The -L option is a global override for Level of detail. When rendering patch surfaces, in some cases, *mantra3* will sub-divide the surface based on a relative level of detail. Surfaces close to the camera will be more finely sub-divided than surfaces further away. The level of detail is usually specified in the Object Editor (see *Level of Detail* p. 288 in the *Objects* section). To speed up renders, the -L option can be used with a fractional value. For example -L 0.5 will render at half the level of detail for all objects in the scene. A level of detail of 0.1 will speed up the rendering considerably.

VARIANCE ANTI-ALIAS (-V)

This option (-v option on the command line to *mantra3*) can be used to specify a different anti-aliasing mechanism. Variance Anti-aliasing takes an argument which specifies the acceptable tolerance in color contrast between pixels. If the tolerance is not met, anti-aliasing is performed. This will anti-alias areas of the image which tend to have aliasing (i.e. edges of primitives, texture maps). Increasing the value of the variance (maximum of 1) results in faster renderings. Decreasing the value (minimum of 0) produces higher quality images.

when to use the -v option

The -v option should produce large speed improvements (up to three times) in rendering time if any of the following are true:

- There are lots of reflective or refractive components in your image.
- There is a lot of refraction.
- You are using ray traced shadows (i.e. fastShadow, transShadow, or filterShadow)
- You are using curvature of the lens.
- A large portion of the visible primitives are metaballs.

when not to use the -v option

- When the scene has motion blur.
- When the scene has depth of field.
- Speed improvement will not be dramatic if there is little raytracing in the scene.

Note: This option must be Off for Motion Blur and Depth of Field to render!

TINY DETAIL CHECKING

When rendering reflections, refractions or shadows, there are options for tiny detail checking. By default, *mantra3* will do medium quality tiny detail checking. If there is very fine detail in the scene (e.g. rendering shadows from a Venetian blind), high quality tiny detail checking will improve the quality of the shadows or reflections. This will add to the render time though. Low quality tiny detail checking will speed up the render, at a cost of anti-aliasing quality.

Tiny Detail Checking uses variance anti-aliasing to check for fine detail during the rendering process. This method allows you to trade off quality for speed.

Variance checks the luminance values (ranging between 0.0 and 1.0) of adjacent pixels. By setting the tiny detail checking to high almost every pixel will be supersampled – a very slow process. However, if you increase the variance you can speed up your rendering at the expense of anti-aliasing quality.

VERBOSE

The Verbose option sets the amount of data reported to the shell during the rendering process. The higher the value entered in this field, more data will be reported during the render.

Z-DEPTH IMAGE

mantra3 has the ability to render Z-Depth images instead of RGBA images. The Z-Depth image will store the distance from the camera to the first object hit. These Z-Depth images are used for Z-Depth shadows and also by atmospheric lights. The Z-Depth image allows *mantra3* to quickly determine if an object is in shadow or is visible by a light source. There are two styles of Z-Depth images:

- Closest Z-Depth
- Average Z-Depth

The Closest Z-Depth will put the Z-Depth of the closest surface to the camera into the image. The Average Z-Depth will average the distances of the first two surfaces and put that value into the image. Average Z-Depth images are better for use in Z-Depth shadow lights since there will be less noise on the surface. However, they will produce incorrect results for atmospheric lights. By default, *mantra3* will generate closest Z-Depth shadows which will work correctly in either case.

Z-Depth Image generates a file from which shadow depth information can be calculated. A Z-depth image contains the Z axis information for each pixel in the scene. Z-depth images are used to produce a more realistic sense of depth, especially in image compositing.

The options for this parameter allow you to choose between Closest and Average depths. *mantra3* defaults to average, meaning that it uses the average of the distances of the first two surfaces hit by the ray from the camera. Closest means that the first surface hit by the ray coming from the camera determines the depth.

13.2 MANTRA3 – MICRO POLYGON PAGE

SHADING QUALITY

The *Shading Quality* parameter sets the quality with which the lighting model (i.e. material) is applied to the object. This means the material applied to the object can have its sharpness increased or decreased depending on how close your camera is to that object (enter an expression that changes the Shading Quality based on distance from the camera).

See *Shading Quality (-s)* p. 750 for more info.

MICRO POLYGON CACHE SIZE

Allows you to specify a cache size for the micro-polygon storage. By default, this is set to 512. Specifying lower numbers (minimum value 8) causes *mantra3* to require less memory. However, rendering times will increase since *mantra3* will be required to do more work.

MICRO POLYGON SPLITS (-S)

If there is a single primitive which spans a large area, but only a portion is visible on the screen (i.e. a single polygon ground plane stretching to infinity), it's possible that some shading artifacts will appear. These will manifest themselves as large blurry areas on the surface. This occurs when *mantra3* is unable to refine a surface into small enough micro polygons. It is possible to increase the maximum threshold of splits by changing this parameter.

The -S option on *mantra3* only applies to splits of a primitive which crosses the near clipping plane. This removes artifacts when rendering small portions of large primitives.

13.3 MANTRA3 COMMAND LINE OPTIONS

mantra3 can be run from a shell script. The above options can be specified form the command line as well as the Render Dialog Script. The options are listed below.

-	Show command line options
-a	Use micro-polygon rendering algorithm -s <val> Specify shading rate multiplier. -G <val> Specify cache size (default 4096) -S <val> Specify max splits (default 16).
-A	Turn off anti-aliasing (default off)
-B val	Specify bucket size for render (default 64)
-b val	Specify bit-depth of output image: 0, any – use the 'natural' bit depth. 8, byte, char – Generate 8 bits per color chan. 16, short – Generate 16 bits per color chan. 32, float – Floating point data per color chan.
-C val	Specify grid cache size (default 256)
-E	Render an even field
-f file	Specify a file to load (instead of reading from stdin)
-w val; -h val	Specify a width & height for output image.
-o file	Send verbose output to the specified file.
-O	Render an odd field.
-p file	Append verbose output to the specified file.
-q	Set render quality: below 9 turns off motion blur; below 8 turns off depth of field; below 5 turns off ray tracing when using -a .
-S	Polygon splits – applies to splits of a primitive which crosses the near clipping plane. This removes artifacts when rendering small portions of large primitives.
-i	Render interactively.
-L val	Global level of detail multiplier.
-J val	Set the jitter scale (default 1).
-n val	Set the number of processes (default 1).
-N val	Force the number of processes (default 1).
-t val	Set tiny detail checking (default 1). 0 = No tiny detail checking; 1 = Medium tiny detail checking; 2 = High quality checking.
-r	Force *mantra3* into ray-tracing mode.
-v	Use variance anti-aliasing.
-V val	Set verbose level.
-z	Render averaged z-depth image.
-Z	Render z-depth image.
-F	Use fast interactive device.

14 MENTAL RAY OUTPUT OP

14.1 DESCRIPTION

WHAT IS MENTAL RAY?

Mental Ray is a highly efficient fully featured ray tracing program written by Mental Images GmbH. It supports procedural shading, motion blur, global illumination, caustics, volumetric effects and many other features.

Futher details may be found at: *http://www.mental.com/*

HOUDINI AND .MI SCENE FILES

implementation

The ingredients for a MentalRay scene description file (.mi) come from many different places in the Houdini environment. Geometry comes from the SOPs of the displayed objects; the camera position is taken from the rendering camera; shader information is gathered from SHOPs. Finally, the Mental Ray Output OP ties it all together.

generating .mi files from houdini

The Mental Ray Output OP generates .mi files from Houdini. The output driver allows you to choose which objects get rendered and to set global parameters to control how the image will be made.

14.2 PARAMETERS – STANDARD PAGE

CAMERA

The camera parameter allows you to choose which camera the image will be rendered from.

VISIBLE OBJECTS

This parameter provides a pattern of all objects which are visible for the render.

OUTPUT PICTURE

This is the filename that mental ray will render the image to. Houdini automatically detects the filenames "ip" and "md" and will send these renderings directly to *mplay.*

OUTPUT IMAGE TYPE

Specify whether 8 or 16 bit images are generated and whether alpha is included in the output image.

OUTPUT FILRE FORMAT

What format the output file is written in (i.e. TIFF, JPG, etc.)

OTHER PARAMETERS

The rest of these parameters are the same as for *mantra* and RenderMan.
See: *Common Output OP Parameters* p. 717 and *Parameters – Standard Page* p. 740:

- Frame Range
- Start / End / Inc
- Initialize Simulation SOPs
- Generate Script File
- Script File
- Render in Background
- Render Command
- Override Default Res
- Resolution
- Pixel Aspect
- Motion Blur
- Depth of Field
- Jitter
- Gamma

14.3 PARAMETERS – SPECIFIC PAGE

The Mental Ray Renderer (ray) provides command line options for most of the parameters on this page. These are denoted by "RayOption". Further help is available in the ray command by running:

```
ray -h
```

SAMPLES

This specifies the sampling quality that mental ray will use to anti-alias the image.

RayOption: -samples

CONTRAST

This is the contrast threshold for anti-aliasing in RGBA.

RayOption: -contrast

TIME CONTRAST

This is the contrast threshold for temporal anti-aliasing (i.e. motion blur)

RayOption: -time_contrast

TRACE DEPTH

Global ray bounce settings.

RayOption: -trace_depth

FILTER

Choose the filter for combining samples.

RayOption: -filter

INCLUDE

This option is very important if you are using non-standard shader. Mental ray requires the inclusion of a .mi file describing the shaders used in the scene. This parameter is used to specify the files which need to be included. This should be a colon (:) or semi-colon (;) separated list of files. On Windows systems, semi-colons are typically required because of drive specifications. We recommend that Unix users use semi-colons as well for cross platform compatibility.

LINK WITH

Not only does Mental Ray require the including of .mi files describing custom shaders, it also requires linking with the actual shader DSO. This parameter allows you to specify which shaders to link against by listing the files in a colon (:) or semi-colon (;) separated list.

RayOption: -link

LEVEL OF DETAIL

This is a global multiplier for the level of detail specified in each geometry object's *Render* page. For more information on how LOD affects quality please see the help below.

BSP SIZE

The maximum number of nodes allowed in each voxel of the BSP tree.

RayOption: -bsp_size

BSP TREE DEPTH

The maximum number of branches in the BSP tree.

RayOption: -bsp_depth

SURFACE DERIVATIVES

Whether or not Mental Ray should compute surface derivatives for geometry.

SURFACE APPROXIMATION

Mental Ray has two different models for tesselation of geometry. The tesselation can be view dependent or view independent. For more information, please see below.

SHADOWS

Turns sorting of shadow intersections on or off.

RayOption: -shadow

SHADOW MAPS

Turn generation of shadow maps on or off.

RayOption: -shadowmap

REBUILD SHADOW MAPS

Re-build shadow maps for each render (or use static maps)

RayOption: -shadowmap

SHADOW MAP MOTION

Whether to incorporate motion blur into shadow maps

RayOption: -shadowmap

REFELCTION MAPS

Whether reflection maps should be auto-generated for reflecting object (this is the auto-generate reflection map option in geometry objects Shading page).

SHOP LENS

Which SHOP lens shader should be used (if any)

SHOP OUTPUT

Which SHOP output shader should be used (if any)

OUTPUT SHADER TYPE

The type of data which the output shader requires

SHOP ENVIRONMENT

The global environment shader (if any)

SHOP CONT STORE

The contour store shader (if any)

SHOP CONT CONTRAST

The contour contrast shader (if any)

CAUSTICS

Turn on rendering of caustics

RayOption: -caustic

GLOBAL ILLUMINATION

Turn on rendering of global illumination

RayOption: -globillum

CAUSTIC ACCURACY

Caustic estimation parameters

RayOption: -caustic_accuracy

CAUSTICS FILTER

Filter used in caustic reconstruction

PHOTON TRACE DEPTH

Maximum ray-bounces when generating photon maps

RayOption: -photon_depth

PHOTON MAP FILE

Path where the photon map will be written.

RayOption: -photon_mapfile

PHOTON MAP REBUILD

Whether to re-build photon maps for every render

GLOBAL ILLUM ACCURACY

Global illumination estimation parameters

RayOption: -globillum_accuracy

PHOTON VOLUME

Photon volume accuracy controls

RayOption: -photonvol_accuracy

FINAL GATHERING

Enable or disable final gathering

RayOption: -finalgather

FINAL GATHER

Controls over accuracy of final gathering

RayOption: -finalgather_accuracy

14.4 PARAMETERS – SCRIPTS PAGE

PRE-RENDER SCRIPT

Execute this script before any rendering.

PRE-FRAME SCRIPT

Execute this script before each frame.

POST-FRAME SCRIPT

Execute this script after each frame.

POST-RENDER SCRIPT

Execute this script after all rendering.

14.5 GEOMETRY AND .MI FILES

SUPPORTED PRIMITIVES

This is the list of primitives currently supported by the Mental Ray output driver:

Open Polygons	
Particle Primitives	
Metaball Primitives	These polygons are silently ignored when generating .mi files. Particle instancing is not supported in this release.
Closed Polygons/Meshes	These are output directly to Mental Ray.

Sphere/Tube/Circle	These are converted to NURBS surfaces and output to Mental Ray.
Bezier/NURBS Surfaces	These are output directly to Mental Ray. Trim curves are supported natively. *

** If no texture coordinates exist on the NURBS surface, default texture coordinates will be generated which map the unit square to the texture space of the NURBS surface.*

TESSALATION CONTROLS FOR BEZIER/NURBS SURFACES

In the Mental Ray Output OP, there are controls over tesselation for surfaces. It is possible to choose: Don't use view-independent approximation; Always use view-independent approximation; Only for objects with multiple instances.

Mental Ray has the option to tesselate geometry based on the space of the object defined (SOP or view-independent) or based on raster space (view-dependent). In general, view based approximations are easier to deal with since as an object gets smaller in screen space, fewer triangles will be required.

When not using view independent approximations, the "view" keyword is output to MentalRay when defining surface approximations.

The level of detail for the object (determined by the value in the object's *Render* page), is used to determine the distance and angle tolerances for approximation.

The distance tolerance is determined differently based on whether view-dependent or view-independent approximations are used. The following table indicates how each type of approximation is computed:

	Dist (View Dependnt)	**Dist (View Independent)**	**Angle**
Length (1)	2.0/LOD	0.2/LOD	10.0/LOD
Surface (2)	2.0/LOD	0.01/LOD	5.0/LOD
Displace (3)	3.0/LOD	0.02/LOD	10.0/LOD
Trim (4)	2.0/LOD	0.01/LOD	5.0/LOD

1) The length approximation is output for polygons or meshes only.

2-4) When NURBS surfaces are output, separate tesselation controls are generated for Surface/Displacement and Trim curve tesselations.

ATTRIBUTE SUPPORT

Currently, only texture coordinates, surface normals and material attributes are supported in .mi file generation.

14.6 APPLYING SHADERS

Mental Ray supports a large suite of shaders. There is support for all shaders (excepting lightmap shaders) in houdini. The shaders are applied in different parts of the Houdini Environment. Shaders are all defined in Houdini using SHOPs. Each mental ray shader has a SHOP associated with it. The SHOP controls the parameters to the shader, not how the shader is constructed.

The following table describes what and where to apply the shaders.

Shader	Meaning	Where
Surface	The surface shader	OBJ/SOP
Surface Shadow	Surface shader for shadow rays	OBJ/SOP
Displacement	The Surface displacement shader	OBJ/SOP
Geometry	The object geometry shader	OBJ/SOP
Atmosphere	Volume shader definition	OBJ/SOP
Background	Environment shader	OBJ/SOP
Contour	Contour shader	OBJ/SOP
Photon	Photon shader	OBJ/SOP
Photon Volume	Photon shader for volume	OBJ/SOP
Light Shader	Illumination from the light	Light
Emitter	The photon emitter shader	Light
Lens Shader	The camera lens shader	Output
Output	The output shader	Output
Background	The global environment shader	Output
Contour Store	The shader to store information for contour shading	Output
Contour Contrast	The contrast shader for contour shading	Output

Note: OBJ/SOP refers to Geometry Object or Shader SOP.

14.7 MAKING SHOPS FOR MENTAL RAY

Houdini 5.x ships with a program called “mids”. It will parse a .mi file and scan it for shader definitions. When it finds a shader, it generates a dialog script from the shader parameters.

Unfortunately, because the .mi file definition doesn’t contain default values for parameters, the default values in the dialog script will not have intelligent initial settings. To set better defaults, you can either edit the .ds file by hand, or in Houdini, set up the SHOP as you’d like and then choose "Make Permanent Defaults" from the preset menu of the SHOP.

14.8 TIPS AND TRICKS

AREA LIGHTS

Currently, there are no controls for area light sources in Houdini. However, by placing the appropriate string in the post-include file of the light source, area lights can be directly output in the .mi stream. For example:

```
opparm light1 post_include ( "rectangle -.3 -.3 -.3 .3 .3 .3" )
```

will turn light1 into an area light source. Please see the mental ray documentation for details on area light specifications.

SETTING DEFAULTS

Rather than hand editing the dialog script generated by mids, choose "Make Permanent Defaults" in the presets (properties 5.1)

15 OBJECT SCENE OUTPUT OP

15.1 DESCRIPTION

Use the Object Scene Output OP to generate Gouraud Shaded renderings like the ones in the Viewport, or to generate Geometry.

Because the Object Scene Output OP generates renderings that are similar to or the same as the Gouraud shaded view in the Viewport, it is ideal for setting up very fast Gouraud shaded test renders.

You can also use this Output OP to output geometry to Inventor and VRML files. The *Renderer* menu allows you to choose the type of output to create.

15.2 PARAMETERS

Only those parameters unique to this operator are discussed in this section.

RENDERER

This pop-up menu lets you select which renderer to use to produce the scene.

gl-hidden line

Produces a hidden line wireframe image of the scene. This provides a reasonable compromise in speed between Shaded and Wireframe output.

When rendering Wireframe/Hidden line, the Anti-aliasing levels are used to determine whether OpenGL line anti-aliasing is used. If the Anti-aliasing levels are 1×1 then no anti-aliasing will be performed in Wireframe renderings. If the values are larger than 1, then anti-aliased lines will be rendered.

gl-shaded

This uses the graphics hardware of your machine to generate a Shaded image of the scene. The output will be very similar to the Gouraud shaded display in the Object Editor's Viewport. This allows you to create quick tests of your animation.

This renderer is special in that when rendering to a flip book, all the images are sent to the same flipbook.

When this renderer starts, a new window will appear. This is where the render will occur. It is important not to occlude this window with any other windows, otherwise the output images may be corrupted.

gl-wireframe

This output driver is similar to the GL Shaded renderer, except that this renderer will render in wireframe instead of shaded display.

When rendering Wireframe/Hidden line, the Anti-aliasing levels are used to determine whether OpenGL line anti-aliasing is used. If the Anti-aliasing levels are 1×1 then no anti-aliasing will be performed in Wireframe renderings. If the values are larger than 1, then anti-aliased lines will be rendered.

inventor

This renderer will generate geometry instead of generating an image. Each object will be placed in the Inventor file under a unique separator node. To output Inventor, you must fill in the script name.

Inventor does not handle conversion of the following primitive types:
Primitive Circles, MetaBalls.

When using texture maps, for Inventor conversion, they should be in SGI format (.sgi) for other Inventor applications to be able to read them.

You can test inventor output by running “ivview”.

vrml

Like the Inventor renderer, this renderer will generate geometry rather than an image. VRML (Virtual Reality Modeling Language) is the geometry format used on the World Wide Web.

Because this renderer requires a converter from Inventor to VRML which is supplied by SGI (ivToVRML), IRIX 5.3 or greater is required – NT machines will be unable to use this feature. Version 2.0 VRML can be generated (or 1.0) depending on the version of the converter installed on your machine).

To test VRML output, you need to have a VRML plug-in for your web browser. Then open the .wrl file within the web browser.

This converter used can be overridden by setting the environment variable: SESI_VRML_CONVERT to a different application. For example:

```
setenv SESI_VRML_CONVERT myVRMLConverter
```

USE DISPLAY INSTEAD OF RENDER SOP

Allows you to choose to render the display SOP instead of the Render SOP.

Why is this option available in this Output OP and not the others? Because in general, when working with Gouraud shaded views in the Viewport, you will want to set the display SOP differently than the render SOP – often you will want the render sop to output a more sophisticated view, such as after a Texture and Material SOP, and the display SOP to a simpler/faster view for Gouraud shading in the Viewport.

Because this Output OP outputs a rendering that is to be similar to the Gouraud shaded view in the Viewport (unlike the other Output OPs such as mantra or RenderMan), this is the only Output OP to have this option, because it is the only one for which it commonly makes sense.

16 OPENGL OUTPUT OP

16.1 DESCRIPTION

Use this Output OP to create images based directly on OpenGL output. This is very similar to the Object Scene output driver (when using OpenGL rendering), except that the OpenGL driver has specific parameters to control hidden line rendering.

16.2 PARAMETERS

Only those parameters unique to this operator are discussed in this section.

RENDERER

GL Hidden Line	Produces a hidden line image using OpenGL.
GL Shaded	Outputs a shaded view of the scene.
VEX Shaded	Outputs a VEX Shaded view of the scene.
GL Wireframe	Wireframe image producing using OpenGL.

USE DISPLAY INSTEAD OF RENDER SOP

Allows you to choose to render the display SOP instead of the Render SOP.

Why is this option available in this Output OP and not the others? Because in general, when working with Gouraud shaded views in the Viewport, you will want to set the display SOP differently than the render SOP – often you will want the render sop to output a more sophisticated view, such as after a Texture and Material SOP, and the display SOP to a simpler/faster view for Gouraud shading in the Viewport.

LINE WIDTH

Applies to the GL Hidden Line and GL Wireframe renderers, but not the GL Shaded renderer. It controls the width of the lines drawn by the renderer. Its default is 1. If you choose 2, lines will be drawn 2 pixels in width, etc. You can also enter fractional values (eg. 1.5). What you see for those values may depend on your video card.

VARIABLE/CONSTANT SENSITIVITY

This provides the same function as the *Variable/Constant* parameter in Viewport options *(Interface > Hidden Line Sensitivity* p. 133). SGI users will need to tweak it to find appropriate values, but most NT users can leave it alone. Unlike the Viewport options on NT, this setting doesn't depend on the kind of video card you have–even if you have a bad card, the default values of: 2.0, 1.1 will usually give good results.

Note: Only available for the *GL Hidden Line* Renderer. It is not available for the *GL Wireframe* or *GL Shaded* Renderers.

17 RENDERMAN OUTPUT OP

17.1 DESCRIPTION

This output operator uses Pixar's RenderMan rendering program to produce your image(s). The Output OP sets up all necessary options and then invokes RenderMan. Alternately, the output can be saved to disk as a RIB file to be rendered later. For detailed information on RenderMan, see the *RenderMan* section of the *User Guide*.

17.2 PARAMETERS – STANDARD PAGE

Only those parameters unique to this operator are discussed in this section.

PRMAN VERSION

This parameter determines which version of RenderMan to use. You have a choice between versions 3.6 - 3.9, and RenderDotC. Some notes about the various RenderMan versions:

- Uniform attributes on polygons were reversed in version 3.6; they are not in version 3.7, so you can expect a difference in output in the later version.
- *prman* 3.7 uses the Points primitive type when rendering particles employing the disk rendering type. The particles' motion is blurred correctly. The *pscale* attribute is mapped to the width variable in prman 3.7, meaning that the particle width is ignored for particles using the Points primitive.
- Version 3.7 renders particles using the new Curves *(RiCurve)* primitive providing the particle rendering type is set to *lines*.

IMAGE DEVICE

From the *Image Device* ▷ menu, you can select an output format, including Postscript, TIFF, Targa, and SGI, for the render. For more on the image file formats, please see *Image File Formats* p. 263 in the *Formats* section.

RENDER COMMAND

The string in this field is used to invoke the commandline which is actually used to create a RenderMan RIB file and invoke RenderMan. Clicking ✚ displays a dialog in which you can interactively set the options for this. Renderman output is discussed in the *User Guide > Renderman* chapter.

PIXEL ASPECT

The pixel aspect ratio is the ratio of horizontal to vertical pixel size of the rendered image. The default value of 1 gives square pixels – suitable for viewing on your screen. The aspect ratio for an NTSC Abekas with 720 × 486 resolution and a 4 × 3 screen aspect ratio is:

$$\frac{4}{3} \times \frac{486}{720} = 0.9$$

MOTION BLUR

This parameter allows you to specify the “default” behaviour for motion blur when rendering. If an object is set to “inherit” motion blur, it will be set to whatever the output driver says (i.e. the object will inherit the behaviour from the output driver). See *Ref > Objects > Geometry Object > Render page > Motion Blur* for details.

Production Tip: Leave most objects as “inherit behaviour”, then specify the motion blur type in the Output driver. This way you can have one render which does motion blur, and another that doesn’t.

DEPTH OF FIELD

Objects can be rendered in focus over a limited range of distances. Objects outside of the limited range of a finite-sized aperture are out of focus. This property is called depth of field. Computer images are generally in perfectly uniform focus; however, by specifying lens parameters in *mantra*, depth of field can be simulated. The focus and f-stop channels of the camera object are used to determine the depth of field.

Depth-of-field is not supported from orthographic cameras.
Depth-of-field cannot have scan line optimization.

JITTER

The Jitter parameter limits the amount of jitter. mantra makes use of jitter as part of its anti-aliasing method. A pixel which is to be super-sampled is divided into sub-pixels. The number of sub-pixels is controlled by the */samplex* and */sampley* channels. See *Jitter* p. 742 for a full description.

GAMMA

Gamma alters a pixel’s intensity in order to compensate for the unique colour characteristics of a given medium. With a Gamma of 1.0 *mantra* thinks a pixel with 50% coverage should be shaded with a 50% intensity. However, many recording devices do not respond to colour in a linear manner, so this is generally not the case in actual practice. How much a device diverges from this linear value is measured by a “gamma curve”. The Gamma for video is 2.2, which differs from that of film (1.0). See *Gamma* p. 757 for a full description.

17.3 PARAMETERS – SPECIFIC PAGE

INITIALIZE

Use this button to clear or set the parameters to their default values (i.e. the *prman* defaults).

TRANSFORMS

This tag allows you to combine the camera transform with object transforms when generating RIB. The downside of this is that you can't easily determine the difference between camera space and world space from within a shader. The plus side of this is that Houdini uses double precision arithmetic for transform computations. Also, the matrix computations are more robust than the ones used in RenderMan. In some cases combining the transforms in Houdini will produce more accurate images (e.g. if both the geometry and camera are transformed 10000 units from the origin, the transforms will be much more accurate).

INSTANCE

Typically, Houdini will use the retained model paradigm in RenderMan to minimize the amount of duplication of geometry. Some RIB renderers other than *prman* do not handle instancing very well. For these renderers, it is better to avoid instancing entirely. This parameter allows the user to do this.

NULL SURFACE

When generating shadow depth maps, Houdini will by default replace surface shaders with a "null" shader for faster Rendering by RenderMan. However, if users do displacement mapping in their surface shaders, or have totally transparent surfaces, this can sometimes cause problems. This option allows the surface shader to be passed down intact.

DICE STITCH

Turns on or off the *Dice Stitch* option when using Pixar's RenderMan 3.9.

RENDERING USAGE / PROGRESS / DEBUG

These options allow for more rendering statistics to be displayed.

OTHER PARAMETERS ON THIS PAGE...

These parameters map directly to features in RenderMan. Various options and attributes can be set using these parameters. If a parameter is blank, then no value will be output in the RIB (i.e. the default values from the *rendermn.ini* file will be used). Please refer to the RenderMan documentation for explanations of these parameters.

17.4 PARAMETERS – AUX FILES PAGE

FILENAME / DEVICE / VARIABLE

You can output multiple images simultaneously from mantra/prman. In prman, you must specify a Filename, Device, and an output Variable and its Type.

QUANTIZATION / DITHER

Optionally, you can specify a Quantization statement and Dither control for the image.

Tip: Up to 6 auxiliary files to be specified. However, it is possible to add additional output images using the pre-include/post-include statements for cameras.

17.5 PARAMETERS – SCRIPTS PAGE

PRE-RENDER SCRIPT

Execute this script before any rendering.

PRE-FRAME SCRIPT

Execute this script before each frame.

POST-FRAME SCRIPT

Execute this script after each frame.

POST-RENDER SCRIPT

Execute this script after all rendering.

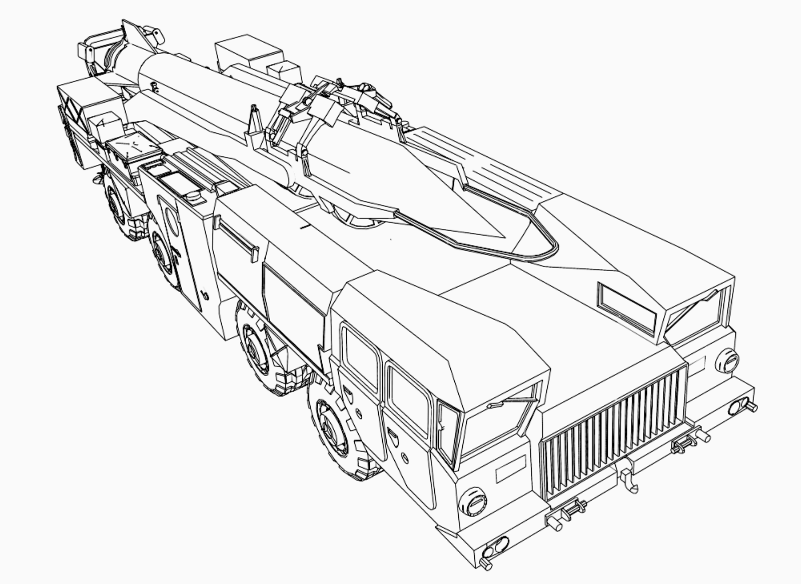
17.6 LOCAL VARIABLES

$N	The current frame of the range specified. This always starts at 1.
$NRENDER	Total number of frames being rendered.

18 WREN OUTPUT OP

18.1 DESCRIPTION

The Wren Output OP produces a hidden line output with a wireframe around the silhouette of objects. For example, when rendering a polygonal sphere, the edges of the sphere (in camera space) are considered to be silhouette edges. The other polygon edges are considered to be internal edges. If an internal edge of a polygon has unique points, it is also rendered in wireframe. This allows you to choose which edges you want revealed on the interior of objects. The *Cusp Polygons* option in the Facet SOP can be used to "harden" edges in smooth models based on a specified angle tolerance.



Sample output from wren – 3D geometry is rendered as a line illustration.

wren will read a scene description file from stdin. The scene description is similar to an IFD, but is much simpler in nature (i.e. no light, fog or material information is required or used by wren).

If polygon colors are attached to the geometry (e.g. with the Cd attribute in the Primitive SOP), these colors will be used to fill the interior of polygons.

When open polygons are rendered, their color is determined by the "Cd" primitive attribute. If the attribute does not exist, then the default wire color is used.

This renderer only supports polygonal models. Therefore, you will have to convert any spline-based (NURBS or Beziers) to polygons before output using a Convert SOP.

Tip: In general, you should use a Facet SOP to generate unique points for Wren.

18.2 PARAMETERS

Only those parameters unique to this operator are discussed in this section.

RENDER COMMAND

The string in this field is used to invoke the commandline which is actually used to create the Wren output. Clicking ✣ displays a dialog in which you can interactively set the options for this.

usage

```
wren [options] [outputfile]
```

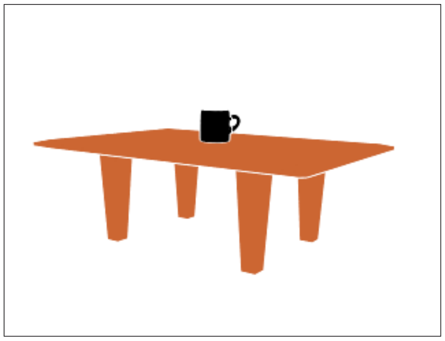
options

`-w width`	Specifies width of output image.
`-h height`	Specifies height of output image.
`-n`	Image will be rendered in “negative” (i.e. a black image with white wireframe).
`-V level`	Specifies verbosity level.
`-l width`	Specifies wire line width.
`-s`	Render with no smoothing.
`-L val`	Global level of detail multiplier.
`-J jitter`	Specify jittering.
`-c`	Disable point consolidation.
`-u attrib`	UV render of the named attribute. This option Replaces the position of the points with a named attribute for rendering. The render window will be automatically converted to an orthographic rendering with the bottom left corner mapping to (0,0) and the top-right corner mapping to (1,1). This option allows users to render the texture-space representation of polygons.
`- m model`	Shading mode – 0-5 (*see note Below): 0 - White Wire / Normal Fill 1 – Black Wire / Normal Fill 2 – White Wire / Ghost Fill 3 – Black Wire / Ghost Fill 4 – White Wire / Matte Fill 5 – Black Wire / Matter Fill
`-p`	Output a Postcript File.
`-P`	Output polygons in Postcript mode.

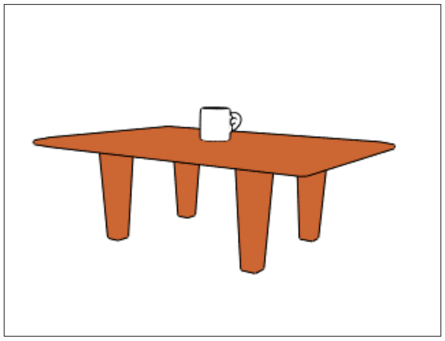
18.3 EXAMPLE

SHADING MODE *

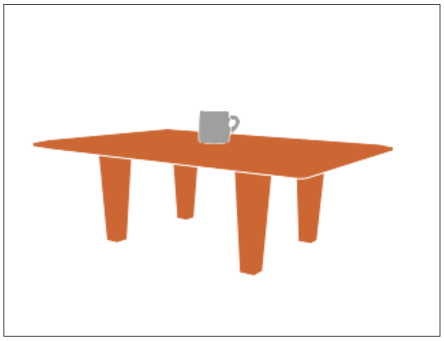
These provide variations of black and white wires, and matte/primitive colour filling as shown:



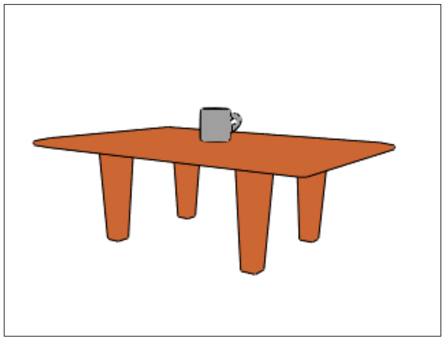
White Wire/Black or Prim Colour Fill



Black Wire/White or Prim Colour Fill

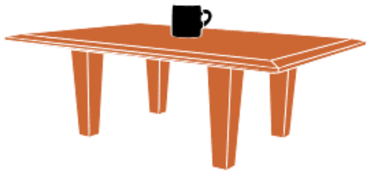


White Wire/Matte Fill



Black Wire/Black Fill

The above examples use the object silhouette as seen from the camera to draw the lines. In order to draw internal lines, each polygon must have unique points. Appending a Facet SOP to the table with the *Unique Points* option enabled (*Cusp Polygons* achieves the same result) and rendering using *Shade Mode*: 0 gives the following result – notice how the edges and legs of the table are more defined:



2 Useful Rendering Info

I MICROPOLYGON RENDERING

I.I INTRODUCTION

There are three basic ways that you can get *mantra* to render scenes; they are:

- Scan Line Rendering
- Variance anti-aliased Rendering
- Micro Polygon Rendering

PROS AND CONS OF RENDERING METHODS

Algorithm	Pros	Cons
Scan Line	• Fast for polygon scenes with no ray-tracing	• Poor anti-aliasing quality • Primitives other than polygons are slow • Ray-tracing is very slow (shadows/reflections/refractions)
Variance	• Fast anti-aliasing of ray-traced scenes • Some primitives (sphere/tube/circle) are very fast to render	• No support for Motion blur or Depth of Field effects • Slows down when there are high-frequency textures
Micropolygon	• High quality anti-aliasing • Fast NURBS rendering • Similar speed to variance for ray-traced scenes • Good user control over speed vs quality • Faster motion blur (lower quality)	• Polygon scenes are slower than Scan Line • Memory usage is higher • Doesn't render metaballs • Doesn't handle depth complexity well

1.2 TIPS & TRICKS FOR MICRO POLYGON RENDERING

DEALING WITH MEMORY

The micro-polygon algorithm is slightly more memory hungry than the other algorithms. You can notice this by using the -V2 option to mantra (which will display the memory usage). The increased memory is used for keeping around sets of micro polygons for faster rendering. There is one control which allows you to tradeoff rendering speed for memory usage. If too much memory is being used, swapping will occur which will slow the render down (or kill the process).

Using the -G option on the mantra command line, it's possible to specify a cache size for the micro-polygon storage. By default, this is set to 512. Specifying lower numbers (minimum value 8) causes mantra to require less memory. However, rendering times will increase since mantra will be required to do more work. For example:

Less Memory/Longer Render	mantra -a -G 64
More Memory	mantra -a -G 8000

1.3 SPEED VS QUALITY

The biggest advantage of the micro-polygon algorithm is that you can specify how quickly objects are to be shaded. This is specified through the "Shading Quality" parameter in the *Render* page of an object. A number less than one (0.5 or 0.2) causes an object to be rendered much faster. A number bigger than one improves the Shading Quality of the object at the cost of rendering time (a value of 1.0 will usually suffice). One important thing to realize is that the Shading Quality can be specified on a per object basis. If an object is taking too long to render, simply lower its Shading Quality a bit (of course, lowering the quality too much will result in poor image quality).

As this is a very important option for micro-polygon rendering, there is a new command line option for *mantra* which specifies a multiplier for the *Shading Quality.* So, for example:

Test Render	mantra -a -s 0.3
High Quality	mantra -a -s 1.4

The *Shading Quality* parameter controls the overall quality of the shading on an object. When dealing with shadows, or reflections, there are additional controls. It's possible to decrease the shadow/reflection quality without very much loss in image quality. This can speed up renders significantly depending on the scene. If you're doing test renders, you can cut the shading rate way down.

Because of the nature of the algorithm, you may find that you need to increase the Shading Quality for reflective/refractive surfaces to get the anti-aliasing you require. If this is the case, you can almost definately turn down the shadow/reflection quality for the object.

1.4 TIPS

It's possible to specify a higher shader quality for a surface which is textured and is causing anti-aliasing problems. The higher shader quality will improve the anti-aliasing on that surface. The rest of the scene will remain unaffected both in quality and time.

When rendering constant shaded or matte objects, it's possible to set the Shading Quality very low – as long as it's not a NURBS or quadric surface, since that would affect the facetting of the edges. However, you will notice artifacts if the surface has texture or atmosphere.

If an object has a large amount of motion blur, decrease the Shading Quality. Typically, a large amount of motion blur hides the shading detail of an object, so it's possible to speed the render up with no visible image quality loss.

SHADING ARTIFACTS

If there is a single primitive which spans a large area, but only a portion is visible on the screen (i.e. a single polygon ground plane stretching to infinity), it's possible that some shading artifacts will appear. These will manifest themselves as large blurry areas on the surface. This occurs when mantra is unable to refine a surface into small enough micro polygons. It is possible to increase the maximum threshold of splits by using the -S option on the command line.

VARIANCE SHADOWS/REFLECTIONS

When rendering using the micro-polygon algorithm, reflections and shadows are anti-aliased using variance rendering. Therefore, it is possible to specify a variance on the command line even when using micro-polygon rendering. If the -a option is specified, mantra will use micro-polygon rendering over any other algorithm.

The -v option does not override the -a option. It works in conjunction. When reflections are done on the -a option, the -v option is used to determine which micro-polygons get anti-aliased.

MOTION BLUR

There is a known bug with motion blur. This may or may not affect your renders. The bug is most visible when primitives are scaling in screen space (i.e. moving away from the camera, rotating, or scaling). The artifact is more visible when the motion blur is large.

On the plus side, deformation motion blur is supported for all primitives. This means that it is possible to have deformation blur on NURBS surfaces.

LEVEL OF DETAIL

Level of detail is not currently used by the micro-polygon algorithm. However, the level of detail will still affect reflections, shadows and refractions.

2 RENDERING SCRIPTS

2.1 INTRODUCTION

It is often much more efficient, in a production environment, to set up your own custom shell script to do your rendering for you then to do all rendering from within Houdini by selecting the *Render* command.

Below, is a full script which you can use as a template to make further modifications. You should setup your render driver as “mantra1” (the default name for a mantra output item). Note that with *mantra3*, you won’t need to use the *shmutil* since shared memory management is already handled by *mantra*.

Tip: A great deal of what is achieved by setting up specialized rendering scripts can be taken care of by using Pixar’s Alfred interface and the Alfred output driver. Alfred allows you to send and manage rendering processes to multiple hosts, and manage a large setup of renders easily. See the *Alfred Output OP* p. 724 for details.

2.2 C-SHELL SCRIPT FOR RENDERING

```
#!/bin/csh -f

if ( $#argv < 3 ) then
    echo Usage: $0 hipfile startframe endframe increment
    exit 1
endif

set hipfile = “$1”
set start = “$2”
set end = “$3”

if ( $#argv > 3) then
    set inc = “$4”
else
    set inc = 1
endif

echo ‘Rendering from:’ $hipfile
echo ‘   Frame range:’  $start ‘to’ $end ‘by’ $inc

#
#  You might want to use something like:
#   hscript $hipfile >& logfile << ENDCAT
#  to keep a log of what gets rendered...
#

hscript $hipfile << ENDCAT
    opcf /out
    set foo = \`execute(“opls mantra1”)\`
    if ( “\$foo” != “single” ) then
  echo “Sorry, can’t find output driver ‘mantra1’”
    else
  # Turn the frame range for single off
  opparm mantra1 trange ( off )
  # If we’re using renderman, you might want to uncomment the following
  # opparm single device ( “tiff” )
  #
  #  Put additional initialization commands in here
  #
  for i = $start to $end step $inc
      opparm mantra1 picture ( /tmp/\$i.pic )
      fcur \$i
      echo \`system("date '+%A, %R')\`: Frame \$i to \`chs("single/picture")\`
      #
      #  Put additional commands in here, i.e. 'unix shmutil'
      #
      render mantra1
  end
    endif
ENDCAT
```

SAMPLE OUTPUT

This is a sample output from the script:

```
Rendering from: script.hip
  Frame range: 1 to 3 by 1
hscript Version 1.1a (Compiled on 09/20/96)
Monday, 12:51 : Frame 1 to /tmp/1.pic
Monday, 12:52 : Frame 2 to /tmp/2.pic
Monday, 12:52 : Frame 3 to /tmp/3.pic
```

3 RENDERING ATTRIBUTES

Following is a list of attributes which Mantra and Wren check for:

3.1 MANTRA

Attribute	Type	Point	Vertex	Primitive
width	float	•	•	•
pscale	float	•		
orient	vector	•		
N	vector	•	•	
surface	string			•
displacement	string			•

3.2 MANTRA3 (OLD MANTRA)

Attribute	Type	Point	Vertex	Primitive
uv	vector	•	•	
Cd	vector	•	•	•
Alpha	float	•	•	•
rest	vector	•	•	
N	vector	•		
rnml	vector	•		
material	string			•

3.3 WREN

Attribute	Type	Point	Vertex	Primitive
linewidth	float	•	•	•
Cd	vector			•

3.4 MEANING OF ATTRIBUTES

`width`	The width of open polygon/Bezier/NURBs curves. These are rendered as "ribbons" which have the width specified. (Point, Vertex, or Primitive SOP followed by an Attribute SOP)
`pscale`	If open polygons/curves are found that do not have a "width" attribute, then pscale is used to determine the width of the ribbons. (Render POP, Point SOP, Particle SOP)
`orient`	Without this attribute, open polygons/curves will be oriented so that their normal points roughly toward the camera. The "orient" attribute specifies an alternative direction for the normal of a ribbon to point at. (Point SOP followed by an Attribute SOP)
`N`	The surface normal. This attribute is ignored for primitives other than polygons/mesh types. (Point SOP, Facet SOP, etc)
`surface`	Specifies a VEX surface shader for the primitive. (Shader SOP)
`displacement`	Specifies a VEX displacement shader for the primitive. (Shader SOP)
`uv`	Texture coordinates (Texture SOP, Point/Vertex SOP)
`Cd`	Diffuse color (Point/Vertex/Primitive SOP)
`Alpha`	The opacity of a point/vertex/primitive (Point/Vertex/Primitive SOP)
`rest/rnml`	The rest position/normal of a surface. (Rest position SOP)
`material`	Specifies a material for the primitive. (Material SOP)
`linewidth`	The width of a line (in pixels). This attribute is only currently supported for open polygons.

3 RenderMan Rendering

1 OVERVIEW

1.1 WHAT IS RENDERMAN?

Houdini can render images with several different renderers; one of these is RenderMan. Whereas the *mantra* renderer comes with Houdini, the RenderMan-compatible renderer *prman* must be purchased separately from Pixar.

RenderMan is actually a specification written by Pixar. This specification describes a language for how a modeling or animation package can communicate with a RenderMan compatible renderer.

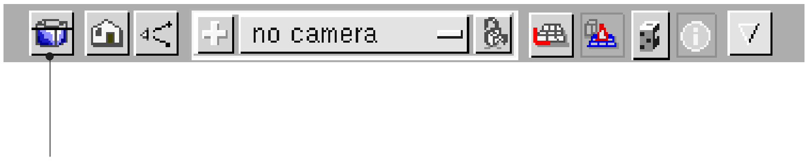
PhotoRealistic Renderman, or *prman*, on the other hand, is the name of a RenderMan compatible renderer sold by Pixar. For convenience, we will generally use the terms “RenderMan”, “RenderMan compatible renderer”, and *prman* interchangeably.

This chapter outlines the subset of RenderMan supported by Houdini.

1.2 QUICK START

Houdini can output complete RenderMan scene descriptions (i.e. ‘RIB Files’). These scene descriptions will include various surface types, shaders, viewing controls, lights, texture maps, and many other rendering controls.

Providing that you have installed RenderMan on your system, using it with Houdini is as simple as clicking an icon.



Click the *Render* Icon to see the *Rendering* pop-up menu

Clicking the *Render* icon (located at the bottom of the Viewport) pops up a menu with at least the following entries: *View* > *Mantra* and *View* > *R-Man*.

It also lists any Render Output OPs you have created. For a simple single-frame render of the current viewport scene, select the *View: R-Man* option – a window appears on the screen, and the scene is rendered with RenderMan.

1.3 FURTHER REFERENCE

PIXAR

Pixar's PhotoRealistic RenderMan renderer, and the RenderMan Specification, is available from:

Pixar Inc.
1001 West Cutting Boulevard
Richmond, CA, 94804 (U.S.A.)
Tel: 415.236.4000

THE RENDERMAN SPECIFICATION

For a complete explanation of the RenderMan Specification, including shading language, the following book is the best reference:

The RenderMan Companion, by Steve Upstill
(A programmer's guide to realistic computer graphics)
Addison-Wesley Publishing Company, 1990.

RENDERMAN REPOSITORY ON THE WEB

http://www.renderman.org/RMR/

This page is just full of RenderMan related information – check it out.

BLUE MOON RENDERING TOOLS

http://www.exluna.com/bmrt/

PRMan isn't the only RenderMan compatible renderer in existence. A shareware RenderMan compatible ray-tracer is available called 'Blue Moon Rendering Tools'. Its a bit slow, but produces excellent results – perfect for learning RenderMan.

RENDERDOTC

http://www.dotcsw.com

RenderDotC is one of the highest performance photo-realistic renderers available, and since it is fully RenderMan compliant, version 3.0.1 (and later) work very well with Houdini.

The most significant thing about this renderer is that the shaders are first compiled to C++ and then compiled to machine language. This results in shaders that evaluate very fast.

2 RENDERMAN SETUP

2.1 RENDERMAN SET-UP

This section assumes that RenderMan is properly installed and running and is of version 3.6 - 3.9. If you type "which render" you should get a path to the RenderMan render command which should be something like:^

`/usr/local/prman/bin/render`

You will need to see your RenderMan installation guide to properly install RenderMan. If you are setting up RenderMan for NT, there will be some minor differences.

2.2 ENVIRONMENT VARIABLES

To properly set up RenderMan with Houdini, you first need to set up some environment variables. These are:

HOUDINI_RI_SHADERPATH	Search path for RenderMan shaders (sets option)
HOUDINI_RI_TEXTUREPATH	Search path for RenderMan textures (sets option)
HOUDINI_PATH	Houdini search path (this is used for the dialog files)

HOUDINI_RI_SHADERPATH

```
setenv JOB /usr/jobs/ROGI  # Make sure directory exists
setenv RMAN_SHADERPATH ".:$JOB/Shaders:$HOME/Shaders:/usr/local/houdini/houdini/
  ri_shaders:&"
```

Note: the $JOB location is optional and can be omitted along with the first line setting the $JOB variable.

By setting this variable, its contents will be inserted into the RIB stream, and will tell the renderer where to go and look.

The $JOB/Shaders location is a user created location and is one example of customizing the environment.

In General, this variable tells RenderMan to first go look for shaders in the local directory, then in a $JOB sub directory called "Shaders" (the job shaders), then it will scan your personal shader directory, then the Houdini default directory, and finally, everything that was already there (the '&'). This '&' will let houdini find all the basic staple shaders like "plastic" that ship with RenderMan.

This variable can be set in Houdini's textport but is generally set or sourced in to your *.login* file.

HOUDINI_RI_TEXTUREPATH

```
setenv RMAN_TEXTUREPATH "???"
```

This controls where the RenderMan render searches for texture maps.

Please remember that RenderMan requires textures to be in a specific format. Convert the image file to a TIFF file, then run the program txmake on the TIFF.

Also, the Houdini FBio table has .tx as an extension for RenderMan texture files so that it will automatically create a texture file for you if you do the following:

```
icp mypicture.pic for_rman.tx
```

At this point, RenderMan is ready to interact properly with Houdini. You should be able to go to an object, bring up the renderman dialog for any shader type, see all the RenderMan default shaders, then press the edit button and edit your shader with a text editor (like vi).

HOUDINI_PATH

```
setenv HOUDINI_PATH ".:$JOB:$HOME/houdini:/usr/local/houdini:$ROGI:$HFS/houd-
  ini:@"
```

Note: the $JOB location is optional and can be omitted.

This defines Houdini's search path for shader dialog files. You only specify the job root directory in the path and not the actual directory containing the Scripts.

Houdini expects the job directory to contain a config/Scripts directory where the dialog files are located and named properly.

The "@" symbol will include all "default" houdini paths since this path is used for other parts of Houdini. For example, the HOUDINI_PATH directory defines the location for all material palettes.

NOTES ABOUT SETTING VARIABLES VARIABLES

@	signifies the “standard shader path”.
&	signifies “whatever was set there before”.

The default *renderman.ini* file states the shaderpath as ".:@", which means "this directory plus the standard shader path" (which is ${RMANTREE}/lib/shaders).

MORE VARIABLES

Following is a table with all the key RenderMan and Houdini variables.

There are more environment variables specific to RenderMan than what are listed here. Please see the PRman documentation for explanations of their use.

For more on environment variables, see the *Scripting* section in the Reference manual.

2.3 RENDERMAN AND HOUDINI VARIABLES QUICK REFERENCE

HOUDINI_PATH	• Houdini variable • Search path for shader dialog script files. • Specify only the job directory in the path. • Must contain a config/Scripts directory where dialog files are located and properly named. • All specific shader dialogs of the proper type must be compiled from the .slo shader using the rmands command.
HOUDINI_STEDIT^	• Specify the program to invoke when editing a shader within Houdini.

HOUDINI_STPATH	• The place to look for source for the shaders.
	• This is a path. For example:
	`setenv HOUDINI_STPATH \`
	`"$HOME/shadetree:$STREE_DIR/`
	`prman-3.7/examples`
	so it would pick up different sets of shaders.

HOUDINI_VIEW_RMAN	• Specifies a command to use for *View:R-Man* in the Viewport Render menu. `setenv HOUDINI_VIEW_RMAN = render`

HOUDINI_VIEW_MANTRA	• Specifies the command to use for *View:Mantra* in the Viewport Render menu.
	`setenv HOUDINI_VIEW_MANTRA =`
	`mantra -a -s 0.7 -v 0.15`

RI_SHADER_PATH	• Sets the path for *ri_shaders* directories.
RMAN_CURVE_BASIS	• Used to control the curve basis for RiCurve level of detail. RiCurve is generally used for recreating hair effects (Houdini 2.5 and later). • Can be set to linear (default) or cubic. Just ensure there are the correct number of points when you use cubic (see the PRman documentation for details). `setenv RMAN_CURVE_BASIS = cubic`
RMAN_FORMAT	• Determines rendered image file format. • Used in the absence of an output driver. • Use "sgif" for .rgb format files, "tiff" is the default. Others are also available.
RMAN_SHADERPATH	• RenderMan variable • Search path for compiled .slo shaders • The directory containing shaders is directly specified in the path
RMAN_TEXTUREPATH	• RenderMan variable • Search path for all required RenderMan .tx texture files.
SESI_SLO_PATH	• Houdini variable. • Specify the directory to build .slo shaders for automatically generated RenderMan shaders. • Used only when rendering to RenderMan and converting materials on the fly.

2.4 RECOMMEND PROJECT DIRECTORY PATH SETUP

You should create a custom directory for your shaders and dialogs. The location is determined by the RMAN_SHADERPATH and HOUDINI_PATH variables.

Here is a recommended generic Job or Project set-up directory structure:

```
Project
  bin
  config
    Scripts
  dso
  geo
  hip
  map
  mat
  pic
  presets
  scripts
  Shaders
```

The two paths you will need for RenderMan will be the *Shaders* directory and the *config* directory indicated above in bold. The *config* directory will also need a *Scripts* sub-directory, so what you end up with are:

`Project/Shaders`	All .sl and .slo files are to be placed here.
`Project/config/Scripts`	All .ds files required for the dialogs in Houdini should be placed here.

CREATING A COMPILED SHADER FROM YOUR .SL SHADER

Your shaders should be placed in the *$JOB/Shaders* directory for your job specific shaders, or in *$HOME/houdini/Shaders* (create the directory if it doesn't exist).

In the *Shaders* directory, you need to compile your .sl shaders with the "shader" command. This will create a .slo file for you. The *shader* command is a RenderMan application so see the RenderMan documentation for more information.

Example: If you have created a shader called *foo.sl*, you will need to enter the following command:

```
shader foo.sl
```

This will create a compiled shader file called *foo.slo*

CREATING RENDERMAN DIALOGS FOR HOUDINI

Once a shader has been created, you now have to create the dialog interface that will be used in Houdini.

All specific shader dialogs of the proper type must be compiled from the .slo shader using the *rmands* command, for example:

```
rmands -c -d foo.slo
```

or to compile all the shaders at once with some preference options and place the dialogs in the proper config/Scripts directory:

```
rmands -c -g 8 -d $JOB/config/Scripts *.slo *.via
```

Please note that running *rmands* on all of the shaders in the directory will concatenate all of them and then save them to the proper shader dialog file automatically for you as follows.

`RMatmosphere.ds`	contains all the atmosphere shader dialogs.
`RMdisplace.ds`	contains all of the displacement shader dialogs.
`RMlight.ds`	contains all of the light shader dialogs.
`RMshader.ds`	contains all of the shader dialogs.

You need to put the resulting .ds file(s) in a directory named */config/Scripts* and to be consistent with above, $JOB/config/Scripts.

A good idea would be to write a script something like:

```
echo creating dialog for shaders...
rmands -c -d . ../../Shaders/*.slo ../../Shaders/*.via
```

or:

```
echo creating dialog for shaders...
rmands -c -d . $JOB/Shaders/*.slo $JOB/Shaders/*.via
```

This script should go in the *config/Scripts* directory.

2.5 SETTING YOUR SHADER EDITOR APPLICATION

Houdini can use any text editor or ShadeTREE to edit a shader interactively from within houdini.

HOUDINI_STEDIT can be used to define your shader editor explicitly or you can rely on the default script that ships with Houdini located in:

```
$HFS/houdini/ri_shaders/editshader.sh
```

This script tests if the shader file to see if it was created with ShadeTREE and if it was, launches ShadeTREE. If not, it uses the text editor defined by your $EDITOR environment variable.

This script is also responsible for automatically re-compiling the shader, and rebuilding the dialog script after committing your changes when you exit the shader editor.

You can make a copy of this script somewhere in your path and edit the file to call up different editors. Or you can use the following environment variable to force Houdini to always use a specific editor.

The example below instructs Houdini to always use ShadeTREE to edit shaders:

```
setenv HOUDINI_STEDIT "$STREE_DIR/bin/st -prman3.7 -f"
```

2.6 GETTING PRMAN TO RENDER TO MPLAY

Through the proto_install utility, it's possible to install display drivers for RenderMan which allow prman to render to mplay.

1) Run proto_install
2) Choose: RManDisplay.inst
3) Choose a location to install the driver

You will be asked:

It is possible to override the default "framebuffer" output device to use mdisplay instead. Would you like to do this? Would you like to do this? [y]

If you select yes, when prman renders images to the "framebuffer" device, they will be sent to the mplay window instead of the default prman framebuffer.

You will then be asked whether you want to test the driver.

This process will have added two new display devices to prman. Choosing "houdini" or "mdisplay" in the device field of the RenderMan output driver will cause images to be rendered to mplay. If you chose to override the "framebuffer" device, then images rendered to the "framebuffer" will also be rendered to mplay.

This process will modify your *$HOME/rendermn.ini* file.

2.7 INSTALLATION FOR ENTROPY

1) Run proto_install
2) Choose: EntropyDisplay.inst
3) Choose a location to install the driver

Entropy does not allow users to override the "framebuffer" device. You will then be asked whether you want to test the driver.

This process adds one new display device to Entropy. Choosing "houdini" in the device field of the RenderMan output driver will cause images to be rendered to mplay.

This process will modify your $HOME/.entropyrc file.

REFERENCE

The display drivers use the "imdisplay" program to transfer data from the renderer to the mplay program. For more information on how this process works, type "imdisplay -", or look at $HH/public/tomdisplay.tar.gz.

2.8 BLUE MOON RENDERING TOOLS SETUP

Blue Moon Rendering Tools
http://www.exluna.com/bmrt/

Blue Moon Rendering Tools (BMRT for short), is a royalty-free rendering program written by Larry Gritz. Some of the things that make BMRT attractive are:

- It is free (just download and go)
- IIt reads standard RenderMan .rib files, and uses .sl shader files.
- It supports ray-tracing and raydiosity
- It generates very good looking images.
- It supports multiple platforms, including WinNT and Linux.

The downside of BMRT is that it is very slow – making it unsuitable for production.

Many users first use BMRT to learn to write shaders, and then purchase PRMan when they become more knowledgeable and they get work that justifies purchasing a copy PRman.

SETTING UP BMRT

- BMRT is very similar to RenderMan in setup.
- Setup BMRT per the instructions in the BMRT documents.

COMPILING BMRT SHADERS

- You will have to copy all of the RenderMan .sl shaders that come with Houdini into your BMRT *shader* directory.
- Next you have to compile your shaders using *slc* – the BMRT shader compiler, not *shader* as with PRMan.
- Any other shaders you want to use have to be saved out of the Houdini material editor and recompiled with BMRT's shader compiler.
- You cannot dynamically render through BMRT as you can with RenderMan.
- Once all the shaders are setup, create a RenderMan Output OP. In the *render* command field, change the *render* command to: *rendrib* (add any command line options you need – although you will not be able to use any RenderMan options).
- To speed things up, set the Super Sample rate to: 1:1 .
- Visit the BMRT site and read the Incompatibilities with PRMan section for more information on how the two applications, PRMan and BMRT, differ.
- Please make sure that you are using the latest version.

KNOWN PROBLEMS

Houdini uses object instancing in its *.rib* files a lot. Older versions of BMRT did not handle object instancing very well, as versions prior to 2.6.-18 corrupted all of the transforms. You will need to use BMRT 2.6.-18 or later to avoid these problems.

2.9 EXAMPLE .LOGIN SETUP

```
unsetenv pathset
limit coredumpsize 0

#---| basic setup procedure |---

alias cd    'set old=$cwd; chdir \!*'
alias dir   'ls -la'
unset noglob

# Set the interrupt characer to Ctrl-C
  if (-t 0) then
    stty intr '^C' echoe
  endif

# Set the default X server.
  if ($?DISPLAY == 0) then
    if ($?REMOTEHOST) then
      setenv DISPLAY ${REMOTEHOST}:0
    else
      setenv DISPLAY:0
    endif
  endif

setenv MOZILLA_HOME /usr/local/netscape
setenv LD_LIBRARYN32_PATH "/usr/lib32:/lib32"

#---| end of basics |---

source /usr/local/bin/baseline
echo "baseline is sourced"

# go home because baseline script leaves me in $HFS
cd $HOME

# These lines are to make local/bin precede the default $HFS/bin
set path = ($HOME/bin $path /usr/local/prman/bin )
rehash

#---| RenderMan Specific Environment Settings |---

# Set the RenderMan shader path
if ( $?HFS ) then
  setenv RMAN_SHADERPATH ".:$ROGI/Shaders:$HFS/houdini/
  ri_shaders:@"
  setenv HOUDINI_STPATH $RMAN_SHADERPATH
  endif
```

```
#---| Project mat Directory |---

# Set houdini path so that there can be a mat directory for
# each project. Note that there is a period as the first directory
# location. This will be the current working directory- or your
# $HIP or $JOB directory
if ( $?HFS ) then
  setenv HOUDINI_PATH ".:$HOME/houdini:/usr/local/houdini:$HFS/
  houdini:@"
endif

#---| Custom |---

setenv HOUDINI_PATH ${HOUDINI_PATH}:/usr/staff/drew/work/Custom-
  Shaders
setenv RMAN_SHADERPATH ${RMAN_SHADERPATH):/usr/staff/drew/work/Cus-
  tomShaders/shaders

#---| TCL/TK Settings |---

set path = (/n/moca/staff/drew/TCL/tcl8.0/unix \
                    /n/moca/staff/drew/TCL/tk8.0/unix $path )
setenv TCL_LIBRARY $HOME/houdini/scripts/tcl8.0
setenv TK_LIBRARY $HOME/houdini/scripts/tk8.0

#---| spy settings |---

echo -n "Do you want to enter spy? "
  set n = $<
  if ( "$n" != "N" && "$n" != "n" && "$n" != "no" ) then
    spy
  endif

if ( ! $?ENVONLY ) then
  spy
endif

# NOTE: Do not set environment variables after setting spy
# EOF
```

3 THE RENDERMAN SCENE DESCRIPTION LANGUAGE

3.1 IMPLEMENTATION

The ingredients for a RenderMan scene description file come from many different places in the Houdini environment. Geometry is taken from the SOPs of the displayed objects; the camera position is taken from the camera OP; all objects have entries for the various shader types; their position gets set from the transformation entries in the geometry OPs. Finally, the output OPs tie it all together.

3.2 RIB IS A KNOWN FORMAT

It is also possible to have Houdini write out RIB “snippets”. These snippets contain only geometry items. This is made possible by having RIB be a known geometry format (write-only, however). Check out the entries in the file */hfs/houdini/GEOio*.

3.3 ACCESSIBLE “MANYWHERE”

There are many different places where you can tell Houdini to write out a RIB stream. These include:

From SOPs, as “Save Object...” and “Save Geometry...” From any view, using the render button From the output OPs, by hitting the “Render” button

3.4 DIFFERENCES FROM PRISMS’ ACTION IMPLEMENTATION

If you have used PRISMS, you will find that the RenderMan capabilities of Houdini are similar to *action*’s. However, many controls in *action* were afterthoughts, and hence had to be “hacked” in. *action*’s Trail SOP motion blur is a prime example of such a “hack”. In Houdini, RenderMan was part of the design consideration. Hence, the entire implementation is much cleaner, straightforward, and powerful.

4 GETTING RIB OUT OF HOUDINI

4.1 RIB STREAMS

A RIB (RenderMan Interface Bytestream) stream is a collection of RenderMan commands. There are two things that you can do with a RIB stream: you can write it to a file, thus creating a “RIB file”, or, you can send the RIB stream directly to the renderer, which, hopefully, will produce a pretty picture.

To send a RIB stream to be rendered, click on a *Render* button below a Viewport. Selecting *View: R-Man* sends the RIB stream directly to the *PRMan* renderer.

To create a RIB file, you first have to create an output OP (described below). From there, you can then create RIB files rather than simple single-frame renderings.

4.2 SAVING RIB FROM SOPS

While you’re in the SOP Editor, you can direct the SOP to write a RIB file. To see the choices, select from the the SOP Tile’s menu. There are two choices:

Save Geometry	This option saves the SOP’s geometry to a file. In the file saver dialog, naming the output file with a “.rib” extension will automatically create a rib snippet containing just the geometry for that SOP.
Save Scene Description	This option creates a renderable RIB file, using the current SOP as the display SOP. The camera setting is the one that matches the current viewport.

4.3 THE RENDERMAN OUTPUT OP

The output OPs in the Output List mode present a unified method of exporting imagery from Houdini. You create sequences of pictures or RIB files through the output OPs. To create a RenderMan output OP, click on the RenderMan icon at the top of the list of output OPs. This adds an entry such as “rman1” to the list of output OPs.

CAMERA

In the parametres area for the rman output OP, the *Camera* pop-up menu lists available perspective objects. Perspective objects are ones through which you can view and render the scene. Houdini considers both cameras and lights to be perspective objects, and thus allows you to render from the vantage point of either of them.

VISIBLE OBJECTS

By default, this edit field is set to render all visible geometry (denoted by the '*' wildcard character). Substituting a list of of object names into this field causes only those named objects to be considered for the render.

OUTPUT PICTURE

This option specifies the name of the image filename that will be created upon rendering (unless 'image device' is set to 'framebuffer' – see the next section *Image Device* p. 804). A typical entry is: *$JOB/Render/foo.\$F4.tif*. It is important to correlate the filename extension, ".tif" in this case, with a proper image device driver.

IMAGE DEVICE

This option determines the image file format of the rendered picture. This is dependent on the image file drivers that are installed in the rendering software. The pop-up to the right of the entry lists most of *PRMan*'s installed display drivers. Enter 'framebuffer' to have the resulting image go to the screen rather than to a file.

FRAME RANGE

If the *Frame Range* button is enabled, you can enter Start/End/Increment values, which determine which frames are output.

GENERATE SCRIPT FILE

By clicking this button, instead of directly rendering Houdini's output, RIB files are created. A typical script filename would be *$JOB/Ribseq/foo.\$F4.rib* .

RENDER COMMAND

If you are having Houdini instantly render your frames, this specifies the name of the executable that will render the frames. Clicking on the ✙ to the right allows you to select parameter flags via a dialog.

SUPER SAMPLE

This is the control for the *PixelSamples* statement. Adding the channel here causes the *PixelSamples* statement to be inserted into the RIB stream.

OVERRIDE DEFAULT RESOLUTION

The resolution of the rendered file is determined by the camera's */resx* and */resy* channels. You can override the settings here. You can enter a horizontal and vertical pixel count for the picture. Additionally, the pixel aspect ratio can also be set here.

MOTION BLUR

The *Motion blur* parameter controls the output of motion blocks into the RIB file. There are three settings: none, transformational, and deformational. Selecting “Transformational Blur” will cause changing Scale/Rotation/Translations to be written out with motion blocks. Selecting “Deformational Blur” will cause topologically consistent yet deforming primitives to be written out with motion blocks. The “motion blur” setting will be inherited by all objects whose “motion blur” parameter is left in it’s default “inherit behaviour” position.

DEPTH OF FIELD

Because it’s an expensive rendering option, depth of field is turned off by default. If your camera has an active focus channel and depth-of-field is checked *on*, you should see a *DepthOfField* statement in the RIB stream.

JITTER

What your pixels do when they see a scary movie.

Jitter can only be 0 or 1 for RenderMan. When 0, it specifies a regular pattern of super-samples; when 1, it specifies that the super-samples be randomised (jittered).

GAMMA

The output gamma of the picture.

4.4 KEYBOARD COMMANDS – ‘RENDER’ AND OPSAVE’

The ‘render’ command is used to invoke an output OP to put out what it’s set up to do. This is exactly like clicking the *Render* button in the interface of the output OP.

```
houdini-> render rman1
```

The *opsave* command causes an OP to write out either geometry (in the case of a SOP) or the scene (in the case of an object OP). In either case, specify the filename with a .rib extension.

```
houdini-> opsave /obj/geo1/font1 foo.rib# just geometry
```

or

```
houdini-> opsave /obj/geo1 foo.rib          # whole scene
```

5 STRUCTURE OF A HOUDINI RIB FILE

Being able to understand a RIB file can be extremely helpful if the rendered image is not what you expected.

Following is a very simple RIB file output by Houdini. When rendered, it creates a grey square in the middle of the picture.

5.1 RIB FILE EXAMPLE #1

```
##RenderMan RIB                                        # line1
# RIB Generated by Houdini
version 3.03
Option “searchpath” “shader” [“.:@”]                   # line 5
ShadingInterpolation “smooth”
# Object geo1
ObjectBegin 2
PointsGeneralPolygons
    [1]                                                # line 10
    [ 4]
    [ 2 3 1 0
  ]
    “P” [ -0.5 -0.5 0 0.5 -0.5 0
   -0.5 0.5 0 0.5 0.5 0]                               # line 15
    “N” [ 0 0 -1 0 0 -1
   0 0 -1 0 0 -1]
ObjectEnd# geo1
FrameBegin 1
 Display “/usr/people/antoine/Render/foo.0001.rgb” “framebuffer” “rgba”
 Format 320 243 1                                      # line 20
 PixelSamples 4 4
 ScreenWindow -1 1 -0.759375 0.759375
 Projection “perspective” “fov” [45]
 Transform [1 0 0 0 0 1 0 0
  0 0 -1 0 0 0 5 1]                                    # line25
 WorldBegin
 TransformBegin
 CoordinateSystem “worldspace”
 TransformEnd
  #                                                    # line 30
  # Light: ambient1
  LightSource “ambientlight” 1 “lightcolor” [0.1 0.1 0.1]
  #
  # Light: light1
  Transform [0.707107 0 -0.707107 0 -0.408248 0.816496 -0.408248 0   # line 35
  -0.57735 -0.57735 -0.57735 0 1.5 1.5 1.5 1]
  LightSource “attenlight” 2  “lightcolor” [ 0.8 0.8 0.8 ]

  Identity# Make sure space is well defined
  #                                                    # line 40
  # Object: geo1
  AttributeBegin
    Attribute “identifier” “name” “geo1”
    Color [0.509746 0.549 0.156465]
    Opacity [1 1 1]                                    # line 45
    Transform [1 0 0 0 0 1 0 0
  0 0 1 0 0 0 0 1]
    Surface “plastic”
    Transform [1 0 0 0 0 1 0 0
  0 0 1 0 0 0 0 1]                                     # line 50
    ObjectInstance 2
  AttributeEnd
 WorldEnd
FrameEnd
```

5.2 ANALYSIS OF RIB FILE

OBJECT DEFINITIONS (LINES 7 - 17)

The object definitions are the first to appear in the RIB file. The object definitions in this case are in an ObjectBegin/ObjectEnd block. The object number is equivalent to the object's position in the objects list ("Houdini-> opls /obj").

FRAME HEADER INFORMATION (LINES 19 - 23)

Following inside the FrameBegin are frame header information, such as the name of the output image file, image size, shading rate, etc.

CAMERA POSITION (LINES 24 - 25)

Right before the WorldBegin is the camera's transformation matrix. This positions the RenderMan camera in accordance with the Houdini camera chosen by the output OP, or is the inspection camera of the viewport.

LIGHTS (LINES 31 - 39)

Next follow all the light definitions. Lights are just another type of shader. The transforms right before the lights position the lights. Lights are on by default. The final Identity statement clears out any existing transformations.

OBJECTS AND ATTRIBUTES (LINES 41 - 52)

Finally come the objects that will be rendered, assuming that they're displayed in Houdini. Each object is in its own "block" (a chunk of RIB), delimited at the beginning and end by and 'AttributeBegin' and 'AttributeEnd' statement, respectively. The object is 'identified', then given a 'Color' and 'Opacity'. The first 'transform' sets up the "shader space" for the subsequent 'Surface' statement, in this case the "plastic" surface shader. The next 'transform' actually positions the object. Finally, an 'ObjectInstance' recalls the object that was defined at the very beginning of the RIB file.

END OF THE WORLD (LINES 53 - 54)

The part of the scene that is rendered is actually inside a WorldBegin/WorldEnd block. The FrameBegin/FrameEnd facilitates multiple render frames inside one RIB file. When rendering Z-depth shadows, you should see one FrameBegin/FrameEnd for each shadow picture, and then one for the main camera view.

6 GEOMETRY AND RIB FILES

This section describes the various types of geometry that Houdini can put into a RIB file. When it comes to creating RIB files, Houdini is very WYSIWYG – “what you see is what you get”. These geometries are translated directly to RIB, with no embellishment, degradation, or translation in the process. You can thus expect to be rendering exactly what’s on the screen.

6.1 GEOMETRY TYPES

PRIMITIVES

Houdini supports just about all the RenderMan primitives. The following RenderMan primitive types and their source inside Houdini.

RiSphere()	Sphere SOP
RiHyperboloid()	Tube SOP
RiDisk()	Circle SOP, end-caps of Tube SOP

POLYGONS

Houdini happily writes out polygons in the RIB stream. Normal interpolation is on by default (ShadingInterpolation “smooth”). Thus, faceting must be accomplished by creating unique points for each polygon (Facet SOP > Unique points).

RiPointsGeneralPolygons()	All SOPs showing polygons

PATCH MESHES

The only current patch mesh supported is a “bilinear” patch.

RiPatchMesh(“bilinear”)	All SOPs showing meshes

NURBS

NURBS are directly supported. NURBS patches are handled by almost every SOP.

RiNuPatch()	All SOPs showing NURBS patches

BEZIER PATCHES

Bezier patches are also supported, but are written as NURBS, of which they are a form.

RiNuPatch()	All SOPs showing Bezier patches.

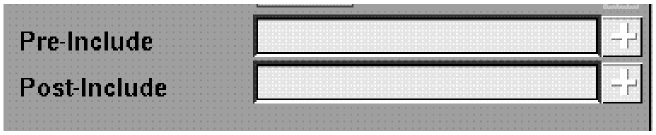
PARTICLES

Particles from the Particle SOP are currently written out as spheres. To get something else you need to use the Copy SOP to place different geometry at each point.

RiSphere()	All particle systems

6.2 GEOMETRY FILE INCLUSION

Houdini has several different places where any extra geometry can be inserted into the RIB stream. These are called “file includes”. Each object has a pair of these, one labelled “pre-include” and the other “post-include”.



6.3 RIB FILE EXAMPLE #2

In addition, the camera and all the lights also sport file pre- and post- includes. Below is a RIB file with each object type including a single line of text.

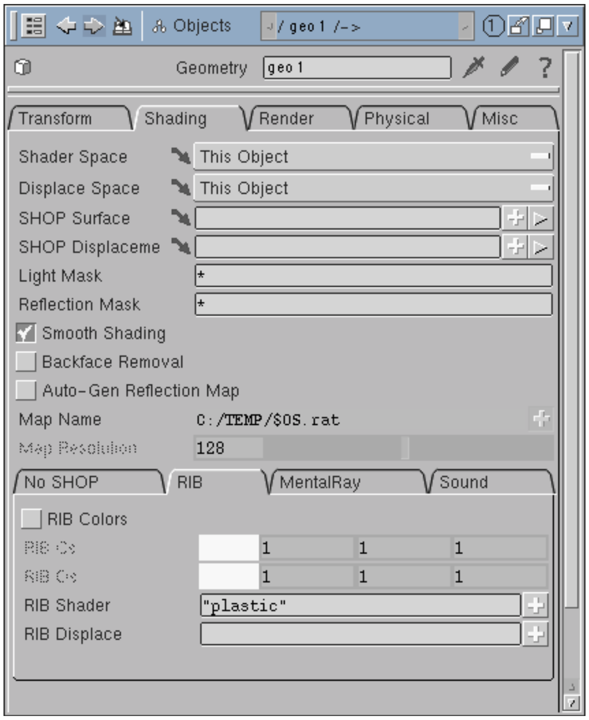
```
##RenderMan RIB
# RIB Generated by Houdini
version 3.03

Option “searchpath” “shader” [“.:@”]
FrameBegin 1
 Display “ip” “framebuffer” “rgba”
 Format 320 243 1
 PixelSamples 4 4
 ScreenWindow -1 1 -0.759375 0.759375
 Projection “perspective” “fov” [45]
# Including /job/rman/Rib/cam1.rib
# CAMERA PRE-INCLUDE FILE
# End of include of /job/rman/Rib/cam1.rib
 Transform [1 0 0 0 0 1 0 0
  0 0 -1 0 0 0 5 1]
 WorldBegin
 TransformBegin
 CoordinateSystem “worldspace”
 TransformEnd
# Including /job/rman/Rib/light1.rib
# LIGHT PRE-INCLUDE FILE
# End of include of /job/rman/Rib/light1.rib
  #
  # Light: ambient1
  LightSource “ambientlight” 1 “lightcolor” [0.1 0.1 0.1]
# Including /job/rman/Rib/light2.rib
# LIGHT POST-INCLUDE FILE
# End of include of /job/rman/Rib/light2.rib

  Identity# Make sure space is well defined
  #
  # Object: geo1
  AttributeBegin
    Attribute “identifier” “name” “geo1”
# Including /job/rman/Rib/object1.rib
# OBJECT PRE-INCLUDE FILE
# End of include of /job/rman/Rib/object1.rib
    Color [0.509746 0.549 0.156465]
    Opacity [1 1 1]
    Transform [1 0 0 0 0 1 0 0
  0 0 1 0 0 0 0 1]
    Surface “plastic”
# Including /job/rman/Rib/object2.rib
# OBJECT POST-INCLUDE FILE
# End of include of /job/rman/Rib/object2.rib
    Transform [1 0 0 0 0 1 0 0
  0 0 1 0 0 0 0 1]
ShadingInterpolation “smooth”
# Object geo1
PointsGeneralPolygons
    [1 1 1 1 1 1]
    [ 4 4 4 4 4 4]
    [ 3 2 1 0
   7 6 5 4
   11 10 9 8
   15 14 13 12
   19 18 17 16
   23 22 21 20
  ]
    “P” [ 0.5 -0.5 -0.5 0.5 0.5 -0.5
   -0.5 0.5 -0.5 -0.5 -0.5 -0.5
   0.5 -0.5 0.5 0.5 0.5 0.5
   0.5 0.5 -0.5 0.5 -0.5 -0.5
   -0.5 -0.5 0.5 -0.5 0.5 0.5
   0.5 0.5 0.5 0.5 -0.5 0.5
   -0.5 -0.5 -0.5 -0.5 0.5 -0.5
   -0.5 0.5 0.5 -0.5 -0.5 0.5
   0.5 -0.5 0.5 0.5 -0.5 -0.5
   -0.5 -0.5 -0.5 -0.5 -0.5 0.5
   0.5 0.5 -0.5 0.5 0.5 0.5
   -0.5 0.5 0.5 -0.5 0.5 -0.5]
    “N” [ 0 0 -1 0 0 -1
   0 0 -1 0 0 -1
   1 0 0 1 0 0
   1 0 0 1 0 0
   0 0 1 0 0 1
   0 0 1 0 0 1
   -1 0 0 -1 0 0
   -1 0 0 -1 0 0
   0 -1 0 0 -1 0
   0 -1 0 0 -1 0
   0 1 0 0 1 0
   0 1 0 0 1 0]
TransformBegin
    ConcatTransform [1 0 0 0 0 -4.37114e-08 1 0
  0 -1 -4.37114e-08 0 0 0 0 1]
    Sphere 1 -1 1 360
TransformEnd
TransformBegin
    ConcatTransform [1 0 0 0 0 1 0 0
  0 0 1 0 0 0 0 1]
    Disk 0 1 360
TransformEnd
TransformBegin
    ConcatTransform [1 0 0 0 0 -4.37114e-08 1 0
  0 -1 -4.37114e-08 0 0 0 0 1]
    Hyperboloid 1 0 0.5  1 0 -0.5 360
TransformEnd
TransformBegin
    ConcatTransform [1 0 0 0 0 -4.37114e-08 1 0
  0 -1 -4.37114e-08 0 0 0 0 1]
    Hyperboloid 1 0 0.5  1 0 -0.5 360
    Disk -.5 1 360
    Disk  .5 1 360
TransformEnd
PatchMesh “bilinear” 2 “nonperiodic” 2 “nonperiodic”
    “P” [ -0.5 -0.5 0 0.5 -0.5 0
   -0.5 0.5 0 0.5 0.5 0]
    “N” [ 0 0 1 0 0 1
   0 0 1 0 0 1]
NuPatch 4 4 [ 0 0 0 0 1 1 1 1] 0 1
    4 4 [ 0 0 0 0 1 1 1 1] 0 1
    “Pw” [  -0.5 -0.5 0 1  -0.166667 -0.5 0 1
   0.166667 -0.5 0 1  0.5 -0.5 0 1
   -0.5 -0.166667 0 1  -0.166667 -0.166667 0 1
   0.166667 -0.166667 0 1  0.5 -0.166667 0 1
   -0.5 0.166667 0 1  -0.166667 0.166667 0 1
   0.166667 0.166667 0 1  0.5 0.166667 0 1
   -0.5 0.5 0 1  -0.166667 0.5 0 1
   0.166667 0.5 0 1  0.5 0.5 0 1]
  AttributeEnd
# Including /job/rman/Rib/cam2.rib
# CAMERA POST-INCLUDE FILE
# End of include of /job/rman/Rib/cam2.rib
 WorldEnd
FrameEnd
```

7 USING SHADERS DIRECTLY

7.1 SELECTING SHADERS

The ability to install custom shaders is one of the main reasons for using Renderman. Shaders allow complete flexibility in the final look of the rendering. However, using RenderMan shaders directly entails more work than just drag-and-dropping material editor components.

All RenderMan shaders are selected in the *Shading* page of the Object OP's dialogs. Houdini supports *surface*, *displacement*, *atmosphere*, and *lightsource* shaders.



/ geo1 /->
Geometry: geo1
Shader Space: This Object
Displace Space: This Object
SHOP Surface:
SHOP Displaceme:
Light Mask: *
Reflection Mask: *
☑ Smooth Shading
☐ Backface Removal
☐ Auto-Gen Reflection Map
Map Name: C:/TEMP/$OS.rat
Map Resolution: 128
☐ RIB Colors
RIB Cs: 1 1 1
RIB Os: 1 1 1
RIB Shader: "plastic"
RIB Displace:

Surface Shaders	Object OP > Shading > RIB Shader
Displacement Shaders	Object OP > Shading > RIB Displace
Lightsource Shaders	Light OP > Shading > RIB Shader
Atmosphere Shaders	Atmosphere OP > Shading > RIB Shader

7.2 SEMI-AUTOMATIC SELECTION

If you write out a RIB file, you'll find many shader calls. How did they get there, even though you didn't select any manually? In addition, how did the parameters to the shaders get set?

There are various RenderMan shader defaulting mechanisms in place. Learning what these are is important, for if you substitute your own shaders, you are forced to manually duplicate the automatic processes.

However, if you create a new shader that has similar parameters to what Houdini puts out by default, you can get the same convenient channel-based control. If you have a three-float parameter such as ‘specularcolor’, simply add the following parameters in the dialog script’s edit fields (and the dialog script will automatically add the back-quotes in the command string at the bottom):

Parameters:	ch($OS/specr)	ch($OS/specg)	ch($OS/specb)
Generated cmd string:	\`ch($OS/specr)\`	\`ch($OS/specg)\`	\`ch($OS/specb)\`

The shader is now under channel control.

7.3 SURFACE SHADER AUTOPARAMETERS

By default, Houdini defaults the surface shader to “plastic”. There are no longer any autoparameters. For comparison, *action*’s “phong”, which mapped to “plastic”, had these autoparameters:

RMan shader params	**Houdini channels**
Ka	average of amb?,
Kd, Ks	hardwired to 1,
specularcolor	spec?
roughness	rough

Closely related (but not actually part of the shader) are the Color and Opacity statement that precede the Surface call. Their values are set by the average of diff? and amb?.

VALID RANGES

In RenderMan shaders (as in most shaders), all parameters are typically 0-1.
0-255 is typically only used when dealing with an 8 bit image.
0-65535 is used when dealing with a 16 bit image.

7.4 LIGHTSOURCE SHADER AUTOPARAMETERS

Depending on the settings in a light object's OP shading dialog, one of the following lightsource shaders is used: attenlight, attenshadow, attenshadowspot, or attenspot. However, to get a shader that actually exists (in *$HFS/houdini/ri_shaders*), you have to leave the shader setting at the default (which is blank).

By default, *attenlight* is used. If you have a string in the 'shadow pic' field, then attenspot shader is used. If you create a lightangle channel, then *attenlight* becomes *attenspot*, and *attenshadow* becomes *attenshadowspot*.

RMan shader params	Houdini channels
atten	lightatten
lightcolor	light?
coneangle	lightangle
conedeltaangle	lightdelta
beamdistribution	rolloff
from	hardwired to [0 0 0]
to	hardwired to [0 0 -1]

If shadows are being created, the light's resx and resy channels determine the size of the z-depth picture.

7.5 OTHER SHADER AUTOPARAMETERS

There are no autoparameters for any of the other shader types.

8 SHADER DIALOG SCRIPTS

8.1 SHADER DIALOG SCRIPTS

Writing shaders is a fun part of using RenderMan. A good starting place is to take a distributed shader and change it a bit, maybe adding a parameter or two. Once you've compiled your shader, you can make a new dialog script for it; this will give you access to all the parameters from within Houdini.

8.2 THE 'RMANDS' COMMAND

The 'rmands' application that comes with Houdini will take any number of RenderMan shaders (.slo) or RenderMan "Looks" (.vma) and create dialog scripts usable inside Houdini.

The first step is to create a home for the soon-to-be created dialog files. From your $JOB directory,

```
Unix% mkdir -p config/Scripts
```

Then, build the dialog scripts:

```
Unix% rmands -d config/Scripts Shaders/*.slo
```

Rmands goes through all the compiled shaders and automatically generates four dialog scripts:

RMshader.ds	for surface shaders
RMdisplacement.ds	for displacements
RMlight.ds	for lightsource shaders
RMatmosphere.ds	for atmosphere volume shaders

The above UNIX command not only creates these four dialog scripts, but it will place them in the *config/Scripts* directory where they will be seen.

Now go back into Houdini and click on the button to the right of the RMan Surface shader. Your custom shaders should be in the list, along with any included in the Houdini distribution.

8.3 SEARCHPATH

Houdini uses a searchpath when looking for dialog scripts. This path is:

```
$HIP/config/Scripts/*.ds
$HOME/houdini/config/Scripts/*.ds
/usr/local/houdini/config/Scripts/*.ds
$HFS/houdini/config/Scripts/*.ds
```

All dialog scripts in these directories will be concatenated, or merged. That is, if there's a RMshader.ds in all four directories, you will see four sets of shaders when you pop up the shader dialog. This permits you to have your own set of dialogs without overriding the system default.

8.4 CAVEATS

It is important not to confuse the existence of a shader in a dialog with the renderer's ability to find it at render-time. The renderer looks for compiled shaders according to its own search path called the shaderpath (see *Shaderpath* p. 816), not the above path. Thus, having the shader interface does not guarantee that the renderer will actually find the shader.

When repeatedly recompiling shaders, you don't need to re-create the dialog scripts every time. The dialogs need to be regenerated whenever a change is made to the parameters of the shader. However, because Houdini caches (keeps in memory) the dialog script when it's accessed, newly created dialog scripts won't show up in Houdini until you save out, exit, and restart Houdini.

9 SHADER OPTIONS

9.1 SHADERPATH

A shaderpath is a text string that consists of various directories through which the renderer can go to search for compiled shaders. Shaderpaths allow you to organize your collections of shaders in any way you choose to best suits your needs. The prman default shaderpath is:

```
.:/usr/local/prman/prman/lib/shaders:&
```

where '.' signifies the current directory, '&' means add whatever was there before, and ':' is the separator for the different entries.

When you write out a RIB file with Houdini, the file will contain a shaderpath statement:

```
Option “searchpath” “shader” \
[“.:@:/usr/local/houdini/houdini/ri_shaders”]
```

where the '@' signifies the standard shaderpath.

If you have any custom shaders, you will need to change the shaderpath. Doing so will indicate to the renderer that you want additional locations searched for compiled shaders.

There is a Houdini variable called RMAN_SHADERPATH. By setting this variable, its contents will be inserted into the RIB stream, and will tell the renderer where to go and look.

However, setting the RMAN_SHADERPATH overrides the Houdini default. Thus, anytime you actually set this variable, you must re-include the Houdini default, so that the renderer finds all the appropriate lightsource shaders.

A good setting would be:

```
Houdini -> set RMAN_SHADERPATH = \
  “.:$JOB/Shaders:$HOME/Shaders:/usr/local/houdini/houdini/
  ri_shaders:&”
```

This causes RenderMan to first go look for shaders in the local directory, then in a $JOB subdirectory called “Shaders” (the shot shaders), then it will scan your personal shader directory, then the Houdini default directory, and finally, everything that was already there (the '&'). This '&' will let it find basic staple shaders like “plastic”.

9.2 SHADER SPACE

Shader space is the coordinate space that is active when the shader call is made. This sounds simple, but its true meaning might be a bit more subtle. The importance of shader space is greatest when shading is done based on the shader global “P”.

For example, look at the original RIB file example #1*RIB File Example #1* p. 806, lines 36 - 37. The lightsource call on line 37 (remember, lights are just another type of shader) is preceeded a transform matrix. The lightsource has a “from” and “to” that point down the negative Z axis. However, when you animate the light, it moves correctly. Why? Because the light does its work internally in “shader” space. And what is shader space? The coordinate system that was resident when the shader was called. Thus, the transform matrix presents a reoriented coordinate space to the lightsource shader, which it uses as its shader space.

For objects, we have surface, displacement, and interior shaders. For each of these shaders, you can explicitly set the shader space. This is done via the object OP > shading > shader_space/displace_space. They default to “this object”, but can be set to follow any other object. For example, you could make a child object, and then slowly move it relative to the parent, and set the parent’s shader space to the child’s. This would give you complete control of the relative shader space.

9.3 BUILT-IN PER-POINT ATTRIBUTES

Each point or control vertex in Houdini can take on attributes in addition to position. Additional attributes such as uv coordinates and normals are generated in SOPs such as Point, Facet, Rest-Position, Texture, Particle, etc.

When RIB is generated from one of these SOPs, a mapping is made between the Houdini name, e.g. “rest”, and it’s correstponding RIB name, “Pr”.

The following is a list of default per-point mappings between Houdini and RIB:

Houdini Attribute	**RenderMan Attribute**
Point “N” (normal)	Per Vertex “N”
Point “Cd” (color)	Per Vertex “Cs”
Point “uv” (texture coord.)	Per Vertex “s” & “t”
Point “rest” (rest postion)	Per Vertex “Pr”

Polygons, meshes, and NURBS patches will get up to one of each of the above attributes for each point. Spheres and quadrics, on the other hand, get four of each, because they technically have four corners, each of which gets one set of attributes.

Vertex attributes in the RIB file override both varying shader parameters and state attributes such as Color. Color, which is accessed in the shader via “Cs”, is declared in RIB with the statement “Color [.5 .5 1]”. In the shader, the color is accessed via the “Cs” variable. However, the existence of a Houdini per-point attribute called “Cd” will trigger the insertion of a per-point attribute “Cs” into the RIB file. Since this per-point “Cs” overrides the state’s color, your per-point coloring will show through in the render.

9.4 CUSTOM PER-POINT ATTRIBUTES

Before Houdini 1.1, adding custom per-point attributes was done via a hack that involved editing the generated RIB file. With the advent of the Attribute SOP in Houdini 1.1, per-point attributes and their RIB representations can now be manipulated within Houdini.

To successfully use per-point attributes, the shader must be able to understand the information Houdini writes out. For example, if you wanted to access the particle “id” in a shader (to give each particle a different but known seed for noise effect), you would need to do the following:

1. Add a “varying float” parameter called “id” to your shader. For example:

```
surface mysurf(varying float id = 0;)
{
  Oi = Ci = noise(transform(“object”, P) * id);
}
```

The parameter “id” defaults to zero, but if per-point attributes called “id” exist in the RIB file, then they will override the parameter.

2. Append an Attribute SOP to your Particle SOP. In the RenderMan section of the Attribute SOP, create a new mapping: enter “id” (without the quotes) in both the “Houdini” and the “RiName” fields; make the RiType “vertex float”; the “offset” can remain zero. Now write out a RIB file. There should now be “id” attributes to your particle spheres, as in this RIB snippet:

```
TransformBegin
  Translate -0.5 -0.5 -0.37344
  Sphere 0.05 -0.05 0.05 360
  “id” [
     0 0 0 0         # sphere is a quadric, and hence has four
   ]                                       # corners which need
values

TransformEnd
```

The noise applied to the spheres should now be different for each sphere. Subsequent birth or death of particles should have no impact on the shading of any spheres.

9.5 SENDING CUSTOM ATTRIBUTES TO A RENDERMAN SHADER

To send arbitrary attributes to your shader as RenderMan variables, you can use the Attribute SOP to create a mapping from the Houdini attribute to a RenderMan variable. Then, in your shader, simply declare the attribute in the shader parameters. For example, if you wanted to have a variable specifying temperature of the surface, generate point alpha values using the Point SOP. Then, in the Attribute SOP, define the mapping:

```
Houdini         RenderMan       Type            Offset
Alpha           Temperature     Vertex Float    0
```

Then, you can declare the Temperature variable in your shader like:

```
surface
heated(...some parameters...; vertex float Temperature = 0;)
```

10 MOTION BLUR

10.1 INTRODUCTION

Motion blur is the term used to describe what is really ‘time anti-aliasing’. Objects that are motion blurred will streak across the screen, as if the shutter of the camera were held open while they moved.

There are two types of motion blur: transformational and deformational. Transformational motion blur is achieved solely with time-varying transformation matrices applied to the objects. Deformational motion blur describes changes in the shapes of the object over time. Houdini supports both types of motion blur.

Assigning motion blur to an object is done in the object OP’s dialog > Render > Motion blur. There are four options: none, inherit behavior, transformational blur, and deformational blur (which includes transformational blur). If the object has no parent, set the option to either transformational or deformational and render. Assuming there’s motion in the scene, the object will blur.

As far as the RIB stream is concerned, you will find MotionBegin/MotionEnd blocks. In between will be either transform matrices or varying geometries.

10.2 TRANSFORMATIONAL MOTION BLUR

Transformational blur is caused by having time-varying transle, rotate, or scale channels. In the RIB stream, these three sets of channels together to make up a Transform statement, which positions the object. Transformational motion blur is what you get when you change this transform over time.

A motion blurred transformation looks as follows in rib:

```
MotionBegin 0 1
  Transformation [ ... 16 floating point numbers ... ]
  Transformation [ ... 16 floating point numbers ... ]
MotionEnd
```

Any and all objects can be motion blurred. The camera object should also get motion blurred, else a fast-moving camera following a fast-moving object will show a blurred object.

10.3 DEFORMATIONAL MOTION BLUR

Renderman can also motion blur geometry that deforms over time. All types of geometry supported by Houdini can be deformation motion blurred. The prerequisite is that the topology, i.e. point and primitive count, remain constant.

If you look at the generated RIB file, there are now additional motion blocks. Rather than containing Transforms, they now contain geometry definitions. For example,

```
MotionBegin 0 1
  PointsGeneralPolygons [ ...many floating point numbers... ]
```

```
    PointsGeneralPolygons [ ...many floating point numbers... ]
  MotionEnd
```

or

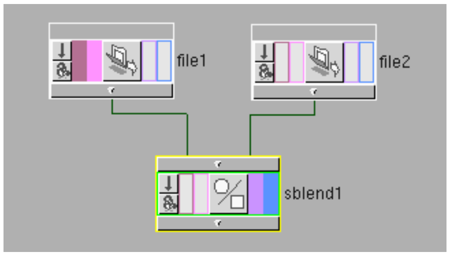
```
  MotionBegin 0 1
    Sphere ....
    Sphere ....
  MotionEnd
```

To get motion blur of objects moving inside the SOPs, the deformational motion blur option must be selected.

10.4 MANUAL DEFORMATIONAL BLUR

Sometimes it is desirable to blur between two “arbitrary” objects, rather than a single object changing over time. A good example of this is when you want to motion blur a file-interp cycle. With a file-interp cycle, you have shape information only at the integral frame times. So, how do you motion blur this?

The answer is to let a Blend SOP do the work, as in the following SOP setup:



The Blend Function is the following:

```
  int($T*$FPS+.01)-floor($T*$FPS+.01)
```

This function creates a sub-frame sawtooth wave. When evaluated at an integral frame number, it returns zero. Halfway to the next frame it returns 0.5, thus nicely blending between the two objects. The “.01” is a “slop” amount to deal with some numerical anomalies in the underlying math computations.

10.5 SHUTTER

The shutter channel on the viewing camera determines the extent of the motion blur. Motion picture cameras can set their shutters anywhere from 0 to 180 degrees. Usually they are set to 180, which means that the shutter is open half the time (180 out of 360 degrees of rotation.) Thus, the Houdini shutter value should usually be set to 0.5. You can set it to less or more, which will cause interesting behaviour.

10.6 IMPLEMENTATION

To implement motion blur, you must take snapshots of the animation environment at two different times, one of these being the “current” time. If your “current” time is frame 50, and now you want to render with motion blur, do you use frames 50 & 5.5, or 49.5 & 50, or 49.75 and 50.25? Houdini takes the approach of “current” and “next”. The reasoning for this is that when you animate with simulations, it is very time consuming to go back to time 49.5, whereas going forwards by 0.5 is very easy.

Thus with Houdini motion blur – because it’s using “current” and “next” frames – you can’t render the last frame of your animation, for there is no “next” frame.

It is interesting to note that Houdini has the ability to cook the frame at any given time. Thus, the shutter and blur amount will always properly correlate.

11 MISCELLANEOUS

11.1 SHADOWS

Shadow generation is automatically done whenever you click on *Auto-generate depth map* in the light object's > Shading page (you must also provide a name for the map in the *Z-Depth Map* field). The RIB output will contain a FrameBegin/ FrameEnd block for every light, plus the one for the main camera view.

Shadows cannot be motion blurred. This is because there is no way to represent motion in a Z-depth picture.

11.2 DEPTH OF FIELD

To render with RenderMan depth-of-field, you must add the focus and fstop channels to the camera. Then, if the 'depth-of-field' flag is turned on in the rman output OP dialog, the RIB stream will contain a DepthOfField statement.

11.3 MATTE OBJECTS

Objects can be placed into the scene but deemed "matte objects". That is, they will occlude other objects and cast shadows, but they will neither appear in the rgb channels, or leave any opacity values in the Alpha channel.

To set an object to be a "matte" object, set the object's 'matte' channel to a non-zero value. The Matte token will be inserted into the RIB stream.

11.4 TEXTURE IMAGES

Houdini can read and write Pixar style .tx texture images. It does this via the *txdspy* command. In the file */usr/local/houdini/houdini/FBio*, there is a line that describes how to go from a .tx file to a .tif file, and back.

```
.tx "txdspy -dspy tiff -dspyfile /tmp/$$.tif %s 2>/dev/null ; icp \
  /tmp/$$.tif stdout ; exec rm -f /tmp/$$.tif"
  "icp stdin /tmp/$$.tif ; txmake -mode clamp /tmp/$$.tif %s ; \
  exec rm -f /tmp/$$.tif"
```

This allows applications like *mplay* to view .tx files directly.

12 FINE TUNING THE RIB FILE

12.1 INTRODUCTION

There are many little-known parameters that can be set in Houdini which control the content of the RIB stream.

12.2 VARIABLES

There are RIB elements that are controlled via Houdini variables. They are the shaderpath, the texturepath, the output file format and the eyesplits option.

HOUDINI VARIABLE USE

RMAN_SHADERPATH	controls where the renderer searches for shaders
RMAN_TEXTUREPATH	where the renderer looks for textures
RMAN_FORMAT	determines rendered image file format. Used in the absence of an output driver. Use "sgif" for .rgb format files; "tiff" is the default. Others are also available.
RMAN_EYESPLITS	sets the "eyesplits" option for PRMan

12.3 CAMERA SPECIFIC CHANNELS

Camera objects can have some channels that play an important role in generating useful RIB files.

shutter	determines exposure, should be 0.5 .
resx, resy	determine the output frame resolution, unless overridden in the output OP.
samplex, sampley	sets the samples per pixel horizontally and vertically.
shaderate	sets the default shading rate. For quality, 1 to .25, 2-100 for speed.

12.4 PER-OBJECT CHANNELS

Each object can also have some RenderMan specific channels.

OBJECT CHANNEL USE

displace bounds	sets the displacement bound attribute, which tells the renderer the maximum amount of displacement an object is likely to use
shaderate	each object can have its own shading rate. Thus, distant objects can maintain a higher rate.

12.5 INCLUDE FILES

Every object in Houdini can also have an arbitrary file. The camera's include-file field is probably the most useful of these. You could, for example, create a file *$JOB/Rib/camera_include.rib*, into which you can put actual RIB snippets, such as a Quantize statement.

13 SUPPORTED FUNCTIONS

13.1 DEFINITION

These are the functions that are supported by Houdini. By supported, we mean that if they have parameters, then those parameters can be controlled in Houdini.

To determine the corresponding RIB statement, remove the 'Ri' and the '() '.

13.2 LIST OF SUPPORTED FUNCTIONS

RiAtmosphere()	Set via the atmosphere objects
RiAtttribute()	object/dispbound channel sets "bound" attribute
RiAttributeBegin()	Begins an attribute block. Brackets action objects.
RiAttributeEnd()	Restores attributes present before previous AttributeBegin.
RiClipping()	Sets clipping planes, from camera/near,far, and light/near,far for z-depth bounds.
RiColor()	Surface color, is average of object/amb? and object/diff?
RiCoordinateSystem()	The "worldspace" coordinate system is defined for you just before the lights are instanced.
RiCropWindow()	For rendering a sub-window, from camera/crop? channels.
RiDeclare()	Per-vertex attributes are now explicitly declared as floats, points, or colors.
RiDepthOfField()	Causes depth-of-field information to be inserted into RIB file. Presence determined by RenderMan output OP's "depth of field" flag, and values controlled with camera's focus, fstop, aperture, and focal channels.
RiDisk()	Disk primitives, and endcaps from tubes.
RiDisplacement()	Displacement shader, in objects dialog for geometric objects
RiDisplay()	Sets the name and type of rendered picture. Contents are controlled by RenderMan output OP's "output picture" and "image device" inputs.
RiExposure()	Controlled with "gamma" channel in RenderMan output OP.

RiFormat()	The resolution of the rendered image, from camera/res? and camera/aspect
RiFrameBegin()	For multi-frame rib files, and separates multiple renderings, e.g. shadows from main view.
RiFrameEnd()	Ends frame block.
RiHyperboloid()	Tube primitives are created with these.
RiIdentity()	Inserted after all lights are defined to set clean state.
RiInterior()	Interior shader, in object OP -> shading
RiLightSource()	A light shader. One for each light, plus one for ambient light.
RiMatte()	Makes matte objects, which have no alpha and aren't visible, but occlude other objects. From non-zero object/matte channel.
RiMotionBegin()	Begins a motion block, in which are either two Transforms or two geometric definitions at two different time values. Existence depends on motion-blur setting in the object OPs.
RiMotionEnd()	Ends a motion block. One for every MotionBegin.
RiNuPatch()	NURBS patches. Note that RMan NURBS patches are all "open"
RiObjectBegin()	Starts the definition of a new object, for potential multiple instancing. Number is same as object number.
RiObjectEnd()	Ends definition of object.
RiObjectInstance()	Recalls object defined with ObjectBegin/ObjectEnd.
RiOpacity()	The initial opacity of an object. Hardwired to [1 1 1]
RiPatchMesh()	Patch mesh geometry. Output from "mesh" objects.
RiPixelSamples()	Supersampling of pixels, used for anti-aliasing. Set by camera/samplex,sampley channels, or in output OP Pixar page.
RiPointsGeneralPolygons()	Polygonal geometry.
RiProjection()	Takes care of perspective/ortho and field of view. Determined by perspective/ortho setting in ostring, plus camera focal and aperture channels.
RiScreenWindow()	Added as part of the frame header.
RiShadingInterpolation()	How polygon normals are interpolated. Controlled by each object's Shading > Smooth_shading flag.

RiShadingRate()	The ratio of the micropolygon to the pixel size. Set by camera/shaderate, or individually on object by object/shaderate.
RiShutter()	The shutter times. Only takes two values are used, 0 and 1. Thus, actual start and end positions are precalculated.
RiSphere()	A sphere quadric. From sphere primitives.
RiSurface()	A surface shader. “plastic” the default.
RiTransform()	Places an object. Determined by the translation, rotation and scale channels of cameras, lights, and objects.
RiTrimCurve()	Appended to NuPatch calls if a trimming profile curve is present.
RiWorldBegin()	Begins the world. All renderable things must follow.
RiWorldEnd()	Ends the world. Much like the first four seconds.

14 UNSUPPORTED FUNCTIONS

14.1 INTRODUCTION

Houdini does not conform to the complete RenderMan specification. Many things have been left out. Some aren't needed, or are supplanted by more powerful statments, e.g. PointsGeneralPolygons vs. Polygon.

14.2 UNSUPPORTED FUNCTIONS LIST

RiAreaLightSource()	Not implemented in PRMan.
RiBasis()	The basis matrix for PatchMeshes, determined by choice of patch mesh type in poly-detail.
RiBegin()	No rib equivalent.
RiBound()	Not really necessary, as RenderMan will compute it on the fly.
RiColorSamples()	Not implemented by PRMan.
RiConcatTransform()	All transforms are absolute.
RiCone()	Can be included in Object OP'S File Includes.
RiCylinder()	Can be included in Object OP'S File Includes.
RiDeformation()	Not implemented in PRMan.
RiDetail()	Not implemented in PRMan.
RiDetailRange()	Not implemented in PRMan.
RiEnd()	No rib equivalent.
RiErrorHandler()	Not supported.
RiExterior()	Not implemented in PRMan.
RiFrameAspectRatio()	Taken care of by Format.
RiGeneralPolygon()	Not needed, as polygons get defined with PointsGeneralPolygons.
RiGeometricApproximation()	Not implemented in PRMan.
RiHider()	Can be included in camera include file.
RiIlluminate()	Turn lights on and off. All Houdini lights are on. objects.

RiImager()	Not implemented in PRMan.
RiMakeBump()	Not supported.
RiMakeCubeFaceEnvironment()	Not supported.
RiMakeLatLongEnvironment()	Not supported.
RiMakeShadow()	Not supported.
RiMakeTexture()	Not supported.
RiOption()	Can be included in camera include file.
RiOrientation()	Can be included in camera include file.
RiParaboloid()	Can be included in Object includes.
RiPatch()	Not needed, as PatchMesh is used.
RiPerspective()	Taken care of by Projection.
RiPixelFilter()	Can be included in camera include file.
RiPixelVariance()	Can be included in camera include file.
RiPointsPolygons()	Not needed. Polygons output with PointsGeneralPolygons.
RiPolygon()	Not needed. Polygons output with PointsGeneralPolygons.
RiProcedural()	Not implemented in PRMan.
RiQuantize()	Can be included in camera include file.
RiRelativeDetail()	Not implemented in PRMan.
RiReverseOrientation()	Can be included in camera include file.
RiRotate()	Not used. Transformations take care of most cases.
RiSides()	Can be included in camera include file.
RiSolidBegin()	Not supported.
RiSolidEnd()	Not supported.
RiTextureCoordinates()	Not supported.
RiTorus()	Can be included in object OP includes.
RiTransformPoints()	Not needed.
RiTranslate()	Transforms are used to position mblurring objects.

15 LIMITATIONS

15.1 THE “SPEC”

The Houdini RenderMan output does not completely conform to the RenderMan specification. The following details these.

PATCH MESHES

Current limitations of patch rendering are that the same basis applies to both U and V directions. It is not possible to use a custom basis; the step may not be changed from the default.

MULTI-FRAME RIB FILES

It is not possible to generate RIB files with multiple renderable frames in them, excepting shadow renderings used for the main camera view.

BINARY RIB

RIB files go out from Houdini in ascii format rather than binary. This is because some calls in the supplied rib library caused core dumps when executed.

LIGHTING MODEL

The lighting coefficients Ka, Kd, – are approximated from the Houdini diffuse, specular, ambient, emission and transparency colors. The latter is more flexible. Channels for Ka and Kd are allocated in Houdini but not yet used.