Denoise

While some RenderMan plugins integrate the Denoise feature, RenderMan does ship with a tool for processing images after rendering. Keep in mind that the appropriate AOVs must be included for the Denoise filter to work.

> Denoising does not affect the alpha channel of images. Doing so can cause artifacts and halos to form and is therefore avoided.

GPU Acceleration

Denoise can attempt to use GPU acceleration. You can activate this mode by adding `--override gpuIndex 0` to the command line, where the number indicates which GPU to use. In a single-GPU system, this will always be 0. If no compatible hardware is found it will failover to CPUs. Using the GPU requires CUDA 7.5 (compute capability 2.0 or later) and a capable graphics card with support.

Enabling GPU Denoise

Globally you can enable the GPU features by changing the .json file here: $RMANTREE/lib/denoise/default.filter.json

gpuIndex corresponds to the GPU installed to use for denoising. If -1 is used, it will use the CPU instead.

```
    "//": "Global settings which apply across all filteres/kernels:",
    "filterLayersIndependently": false,      "//": "If true, filters each render output using weights affected",
                                             "//": "by its color, reducing artifacts but violating additivity",
                                             "//": "and imposing a speed and memory penalty.",
                                             "//": "If false, uses the same weights for all to-be-filtered
layers, ",
                                             "//": "ensuring additivity.",
    "splitSpecularDiffuse": true,            "//": "If true, filters specular and diffuse separately",
    "splitSpecularDiffuseVariances": true,   "//": "When splitSpecularDiffuse and the file has diffuse_var and",
                                             "//": "specular_var layers, use them.",
    "warpType": "linear",                    "//": "cross-frame warp method: linear or nearest",
    "gpuIndex": 0,                           "//": "use GPU denoising on graphics card #0",
```

Denoise Flags

Running `denoise -h` results in the following 'help' output:

```
-o name            Outputs to shotCam_name.exr instead of shotCam_filtered.exr
-n                 Output basename is based on variance image rather than
                     to-be-filtered image
--outdir dir       Outputs to this directory instead of the input file's
                     directory
--filtervariance   If a mix of variance and non-variance files are specified,
                     output includes filtered version of variance's color
                     channels too
--crossframe       Cross-frame sequence mode: filters across frames
--skipfirst, -F    Doesn't output an image for the first frame
--skiplast, -L     Doesn't output an image for the last frame
--layers           Filter only render output layers matching these names.
                     Supports wildcards ?, *, [...].  E.g., --layers 'diffuse,
                     specular,emission[45]'
--list-gpus        This lists CUDA enabled GPUs index
-v name            Uses motion vectors with crossframe mode.  Motion vectors
                     are located based on the variance filename, changing
                     "/variance/" to "/<name>/", "variance." after "." or "_"
                     to "<name>."  Note that "-v variance" will use the
                     motion vectors from the variance file itself
-f configFiles     Filter definition file and/or overrides.  Separate multiple
                     files with '+' or use multiple -f's.  Use -H to list
                     available files.  The default baseFile.filter.json is
                     $RMANTREE/lib/denoise/default.
                     filter.json
-H                 Lists all available filter config files
--override key val Override a value from filter definition file.  Can use
                     multiple times.  If last flag, follow with -- before input
                     file names.  Examples:
                       --override strength 0.5
                       --override 'kernels[1].params.sigma_albedo' 0.05
                       --override debugPixel '[336, 209]'
-t nthreads        Number of threads; default is number of cores on machine
-h, --help         Help
--version          Version information
```

Denoise Filters

The Denoise tool comes with three filter presets:

`default.filter.json`

Used for most scenes. This filter achieves reasonable results without changes.

`sigmaAlphaOnly.filter.json`

Used to filter based on image alpha.

`volume.filter.json`

Used for volumes rendered alone onto transparent black.

It's important to use the volume filter on images containing isolated volume elements or you may introduce more noise into the filtered image using the default filter.

You can override the default filter and settings using the `-f` or `--override` flags. You may also set a different environment override. The below examples are the same result with different methods.

```
setenv NOISE_FILTER_OVERRIDES '{"kernels[0].params.sigma_albedo":0.07}'
```

```
denoise --override 'kernels[0].params.sigma_albedo' 0.07 -- shot.variance.001.exr
```

Optional Pre-made Filter Controls

There are also several override filter settings that can be combined with the base filters. Notice these are named as filteroverride.

fireflyKiller.filteroverride.json

Removes fireflies (bright pixels) from an image.

linearWarp.filteroverride.json

Sets the noise filter's prev/next frame warp type to 'linear'.

nearestWarp.filteroverride.json

Sets the noise filter's prev/next frame warp type to 'nearest' (non-interpolated).

noAlbedoDivide.filteroverride.json

Turns off dividing by the albedo.

noDepth.filteroverride.json

Turns off use of the 'depth' feature to cue the noise filter.

noFireflyKiller.filteroverride.json

Turns off the firefly killer for the noise filter.

nonsplitVariances.filteroverride.json

Defines the parameters of the noise filter, using separate diffuse vs specular variances.

noUnpremultiplyColor.filteroverride.json

Turns off unpremultiply by color options.

splitVariances.filteroverride.json

Defines the parameters of the noise filter, using separate, diffuse vs specular variances.

unpremultiplyColor.filteroverride.json

Turns on unpremultiply by color.

unpremultiplyFeatures.filteroverride.json

Turns on both unpremultiply by color and features.

Example Usage

```
denoise -f +noDepth.filteroverride.json shot.variance.001.exr
```

```
denoise -f volume.filter.json+fireflyKiller.filteroverride.json+linearWarp.filteroverride.json shot.variance.
001.exr
```

Custom Overrides

You may also create your own filter overrides based on your needs. Note that RenderMan looks for the filters under the RenderMan ProServer install directory in `lib/denoise/` The following example creates a control that alters the overall strength of the denoising effect.

```
{
   "//": [
       "This file changes the strength of the noise filter."
   ],
   "filterbanks.*.strength": 0.2
}
```

Saved as `strength0.2.filteroverride.json` you can then use the following command:

```
denoise -f +strength0.2.filteroverride.json shot.variance.001.exr
```

Note that increasing the filter strength may take longer to process the image result.

Custom Filters

Similar to creating custom overrides, you can copy the `default.filter.json` configuration (or one of the others) to a new file and edit it. For example if you saved it to `local.filter.json` you could then use it with the following command:

```
denoise -f local.filter.json shot.variance.001.exr
```

Note that unlike filter overrides, there is no plus sign before the name of the file.

There are three main sections of note in the configuration file. The `kernel` section defines the properties of individual kernels filter kernels – namely the size and tolerances. The `filterbank` section refers to the filter kernels by name and collects them together into filter banks. By default, there are two main ones: one for filtering diffuse layers and one for specular layers (there is also a generic specularAndDiffuse for both in special cases). Finally, there is a `layergroups` section which defines the names of channels in a layer image recognized as belonging together in a layer and gives the names of filterbanks from the `filterbank` section to be used to filter the diffuse and specular elements of the layer.

See the Layered Filtering section here and the comments in `default.filter.json` for more details.