Third Party Shaders

You can make any third-party Arnold shader available inside HtoA, whether the shaders come from other plugins for Arnold such as MtoA, or from other third party shader collections. The HtoA shaders are regular Arnold shaders, there's nothing specific in them except the optional metadata to customize the UI, see below.

Installation

You can make additional Arnold shaders or procedurals available in HtoA in several ways:

- You can just drop the .so/.dylib/.dll files that contain them in the **`arnold/plugins`** and **`arnold/procedurals`** sub-folders of your HtoA installation.
- You can append any folder containing nodes or procedurals you wish to make available in HtoA to the **`ARNOLD_PLUGIN_PATH`** environment variable.
- You can create **`arnold/plugins`** orsub-folders **`arnold/procedurals`** containing the shaders or procedurals in any of the directories of the **`HOUDINI_PATH`**.

Customizing shader user interfaces

HtoA will automatically create a default user interface for shaders. If you need to tweak the UI, you can do so by adding metadata to shader node and parameters. Adding metadata can be done in the shader code with the `AiMetaDataSetX()` API functions or by creating a .mtd text file next to the shader DSO with the same base name, see Arnold Metadata Files for more information. We suggest you study the .mtd files provided with the HtoA installation for inspiration.

Shader metadata

- **houdini.label** (STRING): The UI name of the shader node. Defaults to an automatic capitalized version of the Arnold node name.
- **houdini.icon** (STRING): The icon name.
- **houdini.lod** (STRING): The default LOD of freshly created nodes {**`LOW|MEDIUM|HIGH`**}
- **houdini.order** (STRING): A list of parameter tokens, ordered as you wish them to appear in the UI. Parameters not in the order list will be sorted per Arnold scanning order, and after the order-specified ones.
- **houdini.shader_type** (STRING): Arnold shader category {**`volume|light_filter|atmosphere`**}, default is a surface shader.
- **houdini.parm.<type>.<token>**(STRING): Specify an additional parameter, the *<token>* identifier can be used in the *houdini.order* metadata to insert the parameter. The following types are supported:
    - **houdini.parm.folder.<token>** (STRING): A set of folders (tabs), specify a semicolon separated list of tab label and number of parameters in tab pairs, eg. "`Diffuse;4;Specular;5`".
    - **houdini.parm.heading.<token>** (STRING): An underlined label, the argument of the metadata is the label.
    - **houdini.parm.button.<token>** (STRING): A button, the argument of the metadata is the callback script. The script can optionally be prefixed with "`python:`" or "`hscript:`" to specify the script language. The default language is HScript if no prefix is detected.
    - **houdini.parm.separator.<token>** (STRING): A separator line, the argument of the metadata is unused.
    - **houdini.parm.menu.<token>** (STRING): A user defined menu to be used for drop-down menus, choice-lists, etc. The items are semicolon separated token-label pairs, eg. "`one;Item number 1;two;Item number 2`".
    - **houdini.parm.ramp_float.<token>** (STRING): Group 3 array parameters as a float ramp widget. The argument is a space separated list of the parameter tokens. The parameters must be specified in this order:
        - a FLOAT array parameter for the key positions
        - a FLOAT array parameter for the key values
        - a INT array parameter for the key interpolation types

      These parameters will be automatically hidden and remapped.
    - **houdini.parm.ramp_rgb.<token>** (STRING): Group 3 array parameters as an RGB ramp widget. The argument is a space separated list of the parameter tokens. The parameters must be specified in this order:
        - a FLOAT array parameter for the key positions
        - an RGB array parameter for the key values
        - a INT array parameter for the key interpolation types

      These parameters will be automatically hidden and remapped.
    - **houdini.parm.toggle.<token>** (STRING): A toggle checkbox, the argument is the initial state of the checkbox: "`on`" or "`off`"
- **houdini.parm.<type>.<token>.label** (STRING): Specify a UI label for the extra parameter. The default is an automatically capitalized version of the <token> identifier.
- **houdini.parm.<type>.<token>.desc** (STRING): The tooltip text of the extra parameter.
- **houdini.parm.<type>.<token>.join_next** (BOOL): The next parameter will be on the same line.
- **houdini.parm.<type>.<token>.no_label** (BOOL): Do no display the label.

- **houdini.parm.<type>.<token>.callback** (STRING): A callback script called when the parameter value changes. The script can be prefixed with "`hscript:`" or "`python:`" to specify the language. Without a prefix, the script is considered to be written in HScript.
- **houdini.help_url** (STRING): The help page URL.
- **houdini.category** (STRING): The TAB submenu name under witch to file the shader entry.

Parameter metadata

- **houdini.label** (STRING): The UI name of the parameter. Defaults to an automatically capitalized version of the Arnold parameter name.
- **houdini.help** (STRING): The tooltip text.
- **houdini.type**(STRING): A subtype for some parameters:
    - "*`file:image`*": Add an image file requester button for a string parameter.
    - "*`file:*.toto`*": Add a file requester with "`*.toto`" as file filter.
    - "*`menu:<type>:<language>`*": Add a menu to the parameter, of type {**`replace|toggle|append|wild|exclusive|single`**} , whose items are taken from the *`houdini.menu`* metadata (`menu`), or generated with a Python (`python`) or HScript (`hscript`) script {**`menu|python|hscript`**}."
    - "*`opfilter[:<filter>]`*": The parameter is an operator path, with an optional filter specification. For example, "`opfilter:!!OBJ/CAMERA!!`" will filter out everything but cameras. See `PRM/PRM_SpareData.h` for a complete list of valid filter values.
- **houdini.disable_when**, **houdini.hide_when** (STRING): Parameter expression to specify a condition under which a parameter is hidden or disabled (grayed out). Hiding a parameter will not hide its corresponding input. The expression syntax is described here: http://www.sidefx.com/docs/houdini12.5/ref/windows/optype#conditionals
- **houdini.invisible** (BOOL): Parameter will be created, but its UI will not be visible.
- **houdini.no_menu_caps** (BOOL): Do not auto-capitalize menu entries.
- **linkable** (BOOL): This parameter cannot vary per sample and thus no input should be created for it.
- **houdini.no_input** (BOOL): Do not create a corresponding input for the parameter. Use this metadata only in case you need to override the "**linkable**" metadata.
- **houdini.skip** (BOOL): Skip parameter altogether, it will not appear in control nor inputs.
- **min**, **max**, **softmin**, **softmax** (FLOAT): (hard, soft) x (minimum, maximum) values for the parameter. Note that Houdini only supports a min and a max, optionally qualified as hard or soft, so these values will be mapped to a PRM_Range in the getRange() helper function.
- **houdini.menu** (STRING): A user defined menu to be used for drop-down menus, choice-lists, etc. The items are semi-colon separated token-label pairs, eg. "`one;Item number 1;two;Item number 2`".
- **houdini.menu_script** (STRING): An HScript or Python script to generate token-label pair representing menu items. Cf. http://www.sidefx.com/docs/houdini12.1/hom/assetscripts#parameter_menu_scripts
- **houdini.expression** (STRING): A default expression to set on the parameter. If prefixed with "`python:`", the expression will be treated as Python code, otherwise it will be treated as old expression script.
- **houdini.array_token** (STRING): A token string for array parameters that must contain "`%s`" for the container token and "`%i`" for the element index, in this order.
- **houdini.join_next** (BOOL): The next parameter will be on the same line.
- **houdini.no_label** (BOOL): Do not display the label.
- **houdini.callback** (STRING): A callback script called when the parameter value changes. The script can be prefixed with "`hscript:`" or "`python:`" to specity the language. Without a prefix, the script is considered to be written in HScript.

There are several third-party shader resources available for the Arnold plugins, as listed on the Arnold renderer website.

These third-party shaders, extensions, and integrations are not developed or supported by Autodesk but are provided for your convenience.