Procedurals in Maya

RenderMan procedural primitives are user provided subroutines that can be called upon to generate geometry during the process of rendering. The advantage of procedurals over native Maya geometry is that they can generate complex geometry very efficiently. Procedurals can create incredible geometric complexity from a small number of inputs, and we can delay handling it until the render time. This may make it possible to render much more complex scenes than would be possible with standard Maya representation. Procedurals can also be thought of as units of memory management. The renderer can elect to unload all concrete geometry associated with a procedural knowing that the geometry can be regenerated should there be further need for it.

RenderManProcedural (RiProcedural)

RenderMan for Maya ships with the generic procedural node type of RenderManProcedural that can be used to load a RiProcedural subroutine (not RiProcedural2). This is meant to be quick and easy without requiring installation or customization. For flexibility, a Pre Shape Mel attribute can be used to set the string data parameter during scene translation.

Custom Nodetype Extensions (RiProcedural and RiProcedural2)

Alternatively, users can specify custom mappings of arbitrary Maya nodes to DSO procedurals using .args file parameter lists. This is meant to provide seamless RenderMan integration with exiting Maya plug-ins.

Environment Configuration

RfM can be made aware of custom .args file procedurals by placing a config/mayaTranslation.json extension file under `RFM_PLUGINS_PATH` (22.2), `RFM_SITE_PATH` or `RFM_SHOW_PATH`. Example site configuration:

```
setenv RFM_SITE_PATH /tmp/simple_procedural
```

`RFM_PLUGINS_PATH` is specially designed for procedurals and takes a list of paths:

```
setenv RFM_PLUGINS_PATH "/tools/plugins/myHairPlugin:/tools/plugins/myCrowdPlugin:/tools/plugins/mySpherePlugin"
```

Each path should contain the procedural as well as a `config` directory that may contain the usual files: `rfm.json`, `extensions.json`, `menu.json`, `shelf.json`, etc.

The paths in `RFM_PLUGINS_PATH` will be added to render-time search paths, including the following sub paths:

```
Option "searchpath"
"string rixplugin" ["...:/tools/plugins/mySpherePlugin:/tools/plugins/mySpherePlugin/nodes:/tools/plugins/mySpherePlugin/nodes
/args"]
"string shader" ["...:/tools/plugins/mySpherePlugin:/tools/plugins/mySpherePlugin/nodes:/tools/plugins/mySpherePlugin/nodes
/osl"]
"string texture" ["...:/tools/plugins/mySpherePlugin"]
"string procedural" ["...:/tools/plugins/mySpherePlugin"]
```

Nodetype Mapping

The directory should contain config/mayaTranslation.json which declares translator mappings. In this case, we map the custom Maya nodetype "MayaSphere" to the RenderMan procedural "SphereProc". Example config/mayaTranslation.json:

```
{
    "nodeTable": [
        "MayaSphere", "SphereProc"
    ]
}
```

RfM and RenderMan will search RFM_SITE_PATH and RFM_SHOW_PATH for procedurals DSOs and Args of the specified name, i.e. SphereProc.so and SphereProc.args.

Procedural Args File

The args file should set the plug-in typeTag to "procedural" for RiProcedural or "procedural2" for RiProcedural2. A special "string __dsoname" parameter must also be set to the name of the DSO. Other procedural parameter can be declared similar to shading parameters. The "match" feature can be used to translate Maya attributes with differing parameters name, i.e.:

```
<args format="1.0">
    <page name="Parameters" open="True">
        <param name="__dsoname" type="string" default="SphereProc.so" match="-none-"/>
        <param name="radius" type="float" default="1"/>
        <param name="material" match="rman_material" type="string" default=""/>
    </page>
    <typeTag>
        <tag value="procedural2"/>
    </typeTag>
    <rfmdata
        classification="RenderMan/procedural"/>
</args>
```

Example Download

An example Maya plug-in with RenderMan procedural integration: simple_procedural.zip

Late Material Binding

Materials can be late bound using the __materialid mechanism described here: Ri Material Binding. In Maya, the "string __materialid" parameter corresponds to the Maya shading group name. Users may need to enable "Output All Shaders" under the Advanced section of render settings to output all materials up front.

```
setAttr "rmanGlobals.outputAllShaders" 1;
```