



HyperMan : Seamless integration of Maya® & RenderMan®

HyperMan Technical Documentation

HyperMan History: The Problem

The original aim was to create a tool, which was able to convert shading networks generated in Maya into RenderMan shader equivalents.

Maya has a wide selection of nodes capable to producing complex and interesting results. However we found that the Maya renderer wasn't powerful enough to produce the high quality result that is expected in production quality pieces.

Many production houses use RenderMan or Mental Ray, both have excellent core architecture. Mental Ray has been bundled with Maya, this allows users to create the look they are after. However we also found that Mental Ray for Maya doesn't give users enough flexibility. Users are able to write custom Mental Ray shaders however the API has limited support and is difficult to work with. Due to this fact we have chosen to exploit the advanced features of RendeMan shading language and make them available to every user of Maya.

HyperMan: The solution

We started by evaluating the possible implementation procedures. The first concept was to implement a RenderMan shading language function for every node via a procedural method. We quickly found that the development time to create these functions was too much.

We proposed the idea of converting Maya nodes into texture files then processing the textures in the same way that Maya does. Using texture maps we can generate the expected results from manipulation within the RenderMan shaders.

The system was developed using MEL (Maya Encrypted Language). The advantage of using MEL was apparent immediately. We could get access to everything quite easily and didn't have to process anything outside of Maya.

MEL does have its disadvantages as it is an interpreted scripting language. However MEL has given us flexibility in the use of string handle functions. The system could have been improved by taking the processing into the Maya API. The API gives us full access to the hypergraph that arranges Maya nodes in a tree structure. MEL scripting is still needed to communicate with MTOR (Maya To RenderMan), to allow us to create an automated system.

Another idea proposed was to create a system, which exported the hypergraph shaders, converted them and then parsed the generated RIB file to attach the RenderMan shaders. This would make a more portable system, but it would not allow for an integrated system. This would limit its use as the user would require knowledge of RenderMan.

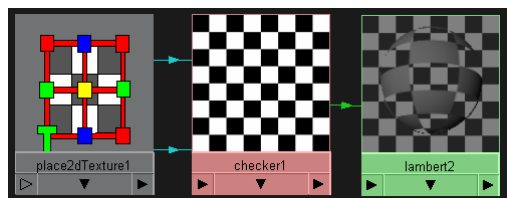
With this in mind we proposed to make the system a user friendly, fully integrated tool that required no further knowledge. Users of Maya would have the facility to exploit the advanced RenderMan features without having to research the RenderMan pipeline.

HyperMan Implementation

In this section we discuss in further detail the action that were taken to produce the HyperMan system.

The first steps taken were in the implementation of the shader translating and texture conversion.

For each basic surface shader in Maya we have a RenderMan equivalent shader. By examining the API source code we determined the shading model used by Maya. Each shader has a shading model for e.g. Blinn is a shading model invented by Jim Blinn. We researched the exact operations taken to create a shader and recreated them in shading language. To deal with the vast number of nodes in Maya we use “convert the texture” to create a texture map. This is then accessed and processed by the RenderMan shader, using a map is a work-around for coding RenderMan procedural equivalents. The outputted file is then converted into a RenderMan format (.tx) that is then plugged into the RenderMan shader. Observe the following example:



Above we have a simple shading network. When the user presses render. The shader is evaluated, each attribute is of the shader is queried for connections. If we find that there is a connection the following node is converted into a texture file using a MEL function, we then run “txmake” a program which is used to convert images into RenderMan usable texture files, we then query if the node is a 2d texture or a 3d texture. Then using MTOR scripting we can import the equivalent shader (in this case hyperLambert.slo) and ask the shader to process the transparency attribute of

hyperLambert.slo using the checker texture map.

Seamless Integration

By closely examining the Maya start-up scripts we were quickly able to integrate RenderMan in the Render globals. Several lengthy MEL scripts each having a different use are loaded up at start-up time.

File formats have to be internally declared to appear in the Render Global settings. Using MEL we can generate an integrated UI to perform various functions via MTOR scripting. Please refer to the source code for more details.

Exploiting RenderMan

To add the advanced features we created a library of functions that are called if requested by the user. MTOR scripting allows us to use arbitrary output variables, which are useful for custom passes. Ray tracing is also supported. Individual passes are generated using surface shaders, an example being occlusion which will send out a given number of rays per point in a cone radius. Irradiance is calculated in a similar way but it returns a colour instead of a float. Both can be cached into files for optimisation. Global illumination is possible by creating a photon map before actual rendering takes place this will calculate the entire light energy in the scene. Then by reading the photon map file we can create global illumination. All features are integrated in such a way that the user can simply tick what they want and press render. Shadows and reflections are produced using image maps, these are automated also. Scripting SLIM templates in TCL that are imported with the shader allow for automatic map generation. It means that the whole system is streamlined and even though the internals are quite complex it is not apparent to the user.

HyperMan Features

- Depth of Field
- Motion Blur
- Ambient Occlusion
- Irradiance
- Global Illumination
- Arbitrary Output variables
- Custom RenderMan shaders
- Image Based Illumination
- Caching of ray traced renders
- Animation caching
- Automatic shadow and reflection maps
- Supported Maya Hypershade Nodes
 - Anisotropic
 - Blinn
 - Lambert
 - Phong
 - Phong E

Any nodes attached in any format to these base shaders can be converted. Attributes of these shaders that aren't supported are listed below.

HyperMan Limitations/ Unsupported Features

- Maya Glow Intensity
- All Mental Ray Attributes
- Per surface ray tracing options
- Hair tube shader
- Layered Shader
- Ocean Shader
- Ramp Shader
- Shading Map
- Surface Shader
- Use Background
- Volume Shader
- Area Light
- Spot Light
- Volume Light
- Paint Effects
- Maya Cloth
- Maya Fur
- Maya Fluid
- Hair

HyperMan Known Issues

Error Message "MTOR" not found... This message will appear when the MTOR plugin is not loaded, the MTOR commands used within HyperMan are not recognisable by the MEL interpreter until the plugin is loaded. Solution, load the plugin before you access the RenderMan render globals tab.

Error Message "SlimCmd" not found... this happens in similar circumstances as the above problem . The solution is to type "source hyperShadeToRenderMan" in the script editor.

Error Message ...Too Many Children... This is a bug, HyperMan will continue to run as normal. There are some cases where the render globals tabs conflict with each other and some elements are hidden. In this case please save your file and restart Maya.

Problem, the render hasn't changed from the previous time. This is when the RenderGlobals tab hasn't updated properly. The solution is to click any UI element in the tab a couple of times.