mrLiquid User Manual

v0.8.0

Table of Contents

Installation

mrLiquid ships with a custom installer that should make installing it a breeze under any platform.

If you have previously installed a demo version, it is recommended that you first uninstall it, to make sure the Maya.env and maya.rayrc files do not end up pointing to the wrong demo plugin and shaders.

If you downloaded your software and find it as a .zip file, first you need to decompress this file. Under modern Windows, this can be done just by selecting the folder and clicking on Extract. Under Unix, most GUI window managers may offer a similar option.

If you can't decompress it with your window manager, download a utility to unzip files. Common utilities that can do this are winzip, zip, pkzip, ark and others.

Once you uncompress the zip archive, you will find different installers for different platforms. Run the installer corresponding to the platform you are installing in.

If you are installing the software for multiple users, it is recommended you run the installer as root or administrator. Under Unix systems, you can often do so by using the command "sudo" before the name of the installer, like:

```
$ sudo ./mrLiquid-0.8.0-Install
```

Under Windows, you may need to log out and log in again as an administrator or power user.

Once the installer starts, you will be requested for a place to install the software. The default location is to install all of filmaura's software in a subdirectory called filmaura.

You can also choose whether to do a typical install or a custom install. A custom install may offer you more control on the different actions the installer does, but most users will probably be fine with a typical install.

The installer will install the software where you indicated. It will also try to create or modify your Maya.env and maya.rayrc files for all maya versions, so that the plugin and shaders are located correctly.

Under Unix, you will also get an environmen.sh bash file in the mrLiquid directory for each maya version.

If you later need to run the plugin as a different user than the one you installed it with, you

should copy these settings to the other user's Maya.env and maya.rayrc file.

The environment.sh file settings are recommended to be installed globally and they supercede the need to have Maya.env installed.

For more information about Maya.env and maya.rayrc, please refer to your Maya manual.

The installer also installs some mel files that are named identical and override some of Maya's standard mel scripts. For safety, these are installed within the mrLiquid directory, and not within the maya install directory. That is, the maya original files remain untouched.

If you have made custom modifications to any of these files, you will need to merge your modifications with the modifications needed for mrLiquid.

The files overridden are:

```
layerEditor.mel
buildShaderMenus.mel
createAndAssignShader.mel
createMRImageFormatControl.mel
createRenderNode.mel
hyperShadePanel.mel
mentalrayCustomNodeClass.mel
mentalrayCustomNodeUI.mel
miItemsNode.mel
renderCreateBarUI.mel
```

Text 1: mrLiquid's mel file overrides

Licensing mrLiquid

Before you can use mrLiquid, you will need to also install its licensing software as well as obtain a license for it.

The current licensing system used to license mrLiquid is called Reprise License Manager or RLM. Together with your mrLiquid installer, you should have received an installer for the license manager, too.

You must install this, by following a similar procedure to installing mrLiquid.

The license manager can be installed on the same machine as mrLiquid or in a completely different machine.

Once installed, the license manager will create a new daemon or server for your machine, which will automatically start whenever you reboot your machine.

You can manually start the license server by using the correct procedure for starting services or daemons for your OS. Under Unix systems, this involves running:

```
$ /etc/init.d/rlm_daemon start
```

Under Windows, this involves going to the Services Control Panel, clicking on the Reprise License Server service and clicking on the start button.

You can also open a console and type the full path to the rlm executable, like:

```
$ %PROGRAMFILES/filmaura/rlm/3.0B4/Windows/win32/bin/rlm.exe
```

Once the license manager is installed, the status of the license manager can also be checked on the web, by pointing your web browser to:

http://localhost:9000 on the license server machine.

Hosts on the network can also check the status of the license server if your network has a proper DNS lookup or you know the IP of the license server machine. In that case, you would do something like:

http://192.128.4.53:3000

or:

http://licserver:3000

When you bought mrLiquid, you should have received an email telling you how to generate a license for your software. Once you generate the license, you will receive it in your email as a text file. You must place this text file in the location where the license manager is installed (the bin/ directory where the filmaura executable resides).

Here's an example of such a license:

```
########################################################################
# License File for:
#     pedro paramo
#     llano en llamas studio
#
########################################################################

HOST licserver1 ANY 2764
ISV filmaura /opt/filmaura/rlm/3.0B4/Linux/x86_64/filmaura

########################################
#
# Licenses for Film Aura's products
# DO NOT MODIFY ANYTHING BELOW THIS LINE
#
########################################
#
# DEMO License for mrLiquid v0.8.0 - Expires Sat Dec 15 07:34:55 CST
2007
#
LICENSE filmaura mrliquid 2007.11 15-dec-2007 1 hostid=0015f24f1389
    share=uh:1 customer="pedro paramo" contract=42 type=" eval demo"
     sig="....."
```

Text 2: Example of license file

If your license manager is not on the same machine as mrLiquid, you must also define an environment variable to specify the machine on the network where it is before starting Maya. Under Unix's bash, this is done with:

```
$ export RLM_LICENSE=2764@yourmachine
```

Under Windows, environment variables can be set in a .bat file or by going to Environment Variables in the Systems Control Panel.

For more information about RLM and license management, please refer to the RLM_UserManual.html that shipped with your copy of the software.

Demo Limitations

If you are testing or evaluating a copy of this software, you must be aware that the demo version has some features removed.

The mrLiquid version demo currently has these limitations:

- Only a single node-locked license is allowed to run simultaneously.
- Batch rendering is disabled.
- Linux rendering is not optimized.

Starting Maya for mrLiquid

Under a Unix platform, you are responsible to set up your shell environment properly before starting the software.

Under a bash shell, this can usually be done by:

```
$ source <install>/<platform>/<arch>/maya8.5/environment.sh
$ /usr/autodesk/maya8.5/bin/Maya8.5
```

where 8.5 is just the version of maya you want to run and <install> is the directory where you installed mrLiquid and <platform> and <arch> are the platform and architecture directories,like Linux-x86/i686.

Under Windows, as long as your Maya.env and maya.rayrc file are located inside your maya/<version>/ directory in your user profile directory, you should be all okay to start maya from any shortcut.

In addition, on Windows, a .bat file is also provided, similar to the environment.sh of Unix. It is used, like:

```
> <install>/<platform>/<arch>/maya8.5/environment.bat
> %PROGRAMFILES%/Autodesk/maya8.5/bin/maya.exe
```

How to Use mrLiquid

Load the plugin into maya from the plugin manager:

Window->Preferences->Plug-in Manager

Select the mrLiquid plug-in from the list of plug-ins. If mrLiquid does not appear in the list of plug-ins, make sure your **MAYA_MODULE_PATH** has been set correctly. By default, this is done in your Maya.env file and in your environment.sh for Unix systems.

The plugin, can work as either a full Mayatomr replacement or as a complement to it. The default behavior is to act as a complement to it, so Mayatomr will get loaded automatically too (if it has not been loaded yet).

If you have rubyMEL installed, you might also want to optionally load it, too, so that mrLiquid can use ruby. If you like the plugins enough, you can set it to autoload.

Once the plug-in is loaded, it will create a new menu in the Rendering section called mrLiquid. From there, you can control all of mrLiquid's options.

mrLiquid currently works in conjunction with a standalone renderer such as the mentalray standalone renderer and will not render anything by itself.

For mrLiquid to find the mental ray standalone, you must have it on your path. By default, mrLiquid will look for the command named “ray” (Unix) or “ray.exe” (Windows).

To change this, the variable **MRL_RAY** can be set in your environment, in Maya.env or within maya itself, by doing something like:

```
putenv("MRL_RAY", "C:/Program Files/mentalray3.4/bin/ray.exe");
```

From then on, mrLiquid will try to start mrLiquid using that version of the ray executable.

Warning:

Albeit the ray executable can be changed, it should still be compatible with the shaders used for the Maya version you are using. It is ill advised to run a Maya that uses a Mayatomr with a mentalray version like 3.4 and a mentalray standalone version that is, for example, 3.5 or viceversa.

Custom shader Installation

Just as with Mayatomr or the standalone, mrLiquid can use any custom shader. To have the custom shaders work, you need to set the **MI_CUSTOM_SHADER_PATH** variable to point to both the .mi file as well as the .so or .dll shader.

How does mrLiquid find shaders?

mrLiquid itself will rely only on the variable **MI_CUSTOM_SHADER_PATH** to find the shaders for the mentalray stand-alone. The paths listed there must list the paths to the .mi files AND the path to the .so or .dll file of the shader.

From that variable, the variables **MI_RAY_INCPATH** and **MI_RAY_LIBRARY** are set up so that the stand-alone can render.

The maya shaders are located from the environment variable MI_ROOT. If MI_ROOT is not set, mrLiquid will locate the maya shaders based on MAYA_LOCATION.

How does Maya create the GUI for shader?

To create the GUI to the shaders, Maya relies on the variable **MI_CUSTOM_SHADER_PATH**. The paths listed there must list the paths to the .mi files.

How does Mayatomr find shaders?

Sadly, Autodesk's approach to have Mayatomr find the shaders is much more complex. Mayatomr relies on a special configuration file called **maya.rayrc** file, which must list both the .mi file and the .so or .dll file. This configuration file is usually insider the user's maya prefs directory.

If the maya.rayrc file does not exist, newer versions of Mayatomr will look for shaders only in the maya mentalray install directory.

From Maya2008 onwards, Mayatomr now works similar to mrLiquid and can use

MI_CUSTOM_SHADER_PATH to locate both the shader binary and the include files.

How does the mental ray stand-alone find shaders?

The stand-alone can rely on reading the maya.rayrc file (by manually adding it), by reading a .rayrc file (which it reads by default) or by setting the environment variables **MI_RAY_INCPATH** and **MI_RAY_LIBRARY**. It will also look for shaders in **MI_ROOT/**include and **MI_ROOT/**lib, if **MI_ROOT** is defined.

Using the environment variables is the recommended approach. When mrLiquid calls the stand-alone, you don't need to set any variables. mrLiquid will set **MI_RAY_INCPATH** and **MI_RAY_LIBRARY** path automatically from the value of the **MI_CUSTOM_SHADER_PATH** variable.

IPR

Unlike maya or mentalray's ipr, mrLiquid's IPR works, for the time being, a tad outside the context of maya.

To start it, go to the **Rendering->mrLiquid->IPR.**

In there, you have three options: **Normal**, **Two-Pass** and **Progressive**.

Normal will just render with whatever settings you have in your render globals.

The **Two-Pass** option will render the IPR in two passes. A first rough pass and a pass with your render global settings.

The **Progressive** option starts an ipr which will begin rendering at a rough setting and progressively increment its render settings after each idle time until it reaches what you set in the render globals.

The rough settings for both the two-pass and the progressive algorithm can be changed from within **Rendering->mrLiquid->IPR->Options**.

Ideally, the IPR should keep working properly with almost anything you do (but check KNOWN BUGS to see some current issues).

You can do a lot with it:

- new/import/save/load/add reference scenes.
- create objects
- model objects
- change light/shader/camera/render settings parameters
- change render camera
- reset crop regions
- change render settings, including motion blur
- move lights, cameras, objects
- change light linking
- add/remove approximations
- change/deform objects (either manually, with bones, or deformers)
- attach and assign new materials and shaders
- change frames

- change object and instance flags
- change render layer
- hide/template/unhide objects
- duplicate objects or instances
- rename objects/shaders
- reparent objects or shapes
- add groups
- create/delete/edit render layers

If at any point the IPR gets out of sync with what you see in the wire frame, you can stop it and restart it.

Image Viewer

When mrLiquid starts rendering, it will try to open a new image viewer, in addition to sending the 8-bit image results back to Maya's Render View.

Using another viewer has a number of benefits compared to Maya's Render View, as it allows you to do things like check the alpha channel while rendering.

mrLiquid by default will search your **PATH** variable for the following image viewers:

- The value of the **MRL_VIEWER** environment variable.
- mrViewer
- imf_disp

Using mrViewer as your image viewer for rendering is the best alternative, as it offers you full floating point renders, gamma lookups, 3D LUTs, CTL support, cloning, compositing and many other features. For more information about mrViewer, please visit Film Aura's website at: http://www.filmaura.com.

If no stand-alone image viewer can be found, mrLiquid will use **imf_disp**, which is mental ray's stand-alone image viewer that ships with maya.

If your facility already has a custom viewer, you can use it instead by setting the environment variable **MRL_VIEWER** to the name of the executable. Note that the image viewer must be able to read OpenEXR images as well as connect to mental ray through a socket by using a stub file. If you want to show mental ray shadow maps, the viewer must also support the format.

Additions Shader Features

One big feature of mrLiquid's shading networks is the ability to allow any of mrLiquid's special string variables in any attribute that takes a string.

Thus, for example, a maya_file texture node could be set to load the texture ${OBJNAME}.iff. And when the scene is rendered with mrLiquid, this single shader will expand itself to multiple copies of the shader – one for each object and the ${OBJNAME} keyword will be replaced by the name of the object.

Additional Texture Features

mrLiquid offers some enhancements in handling of textures compared to Mayatomr. It can use the additional attribute makeMipMaps defined in the mentalrayOptions node to determine whether textures in the current render layer are converted to mipmaps or ripmaps. If the attribute does not exist, the environment variable **MRL_MIPMAPS** is checked. If set, mipmaps are generated. If not set, textures are handled just like Mayatomr.

When ripmaps are used, mrLiquid can optionally use its own mrl_exr_file shader instead of maya's file shader, without you having to modify any single file in the file requesters. The benefit of mrLiquid's exr file reader is that it antialiases properly on all rays using ray differentials and takes advantage of ripmaps to give you better stats. To use the mrl_exr_file reader instead of maya's own file shader, set the environment variable **MRL_EXR_FILE** to 1.

The first time a texture heavy scene is rendered, it can thus take some time for mrLiquid to convert all textures to the OpenEXR format. But afterwards, only those textures that have changed on disk will be reconverted.

OpenEXR textures are saved in the directory specified in the **-ripmapDir** flag to the mental command. If not set, the location used is the same as the original texture directory. The files will be named like the original texture, but receive an additional "_r.exr" extension (standing for ripmap exr). In case of maya file nodes, mrLiquid will check the texture wrapping modes and may also append a "_c" (clamp), "_p" (periodic) or "_m" (mirror) for U and V. This will make seams invisible if the texture is tiled or wraps.

These textures are then fed to mrLiquid's exr shader. This shader performs texture filtering using ray differentials and will handle texture wrapping properly, unlike Mayatomr. It often renders faster than using ripmap .map files.

The exr file reader reports a message every time a file texture is first opened, to point to the name of texture being opened, as well as its file descriptor. File descriptors are the number of simultaneous open files that an OS can handle. Some good operating systems like linux allow this value to be manually configured, while others like Windows have the maximum number locked to a certain value. By default, mrl_exr_file will recycle file descriptors after 512 textures are read.

When the render finishes, with the mrl_exr_file reader, you will also receive texture statistics, like:

```
PHEN 0.2  info : ---------------------------------------------------
PHEN 0.2  info :                    Texture Statistics
PHEN 0.2  info : ---------------------------------------------------
PHEN 0.2  info : Total Texture Data:       23440.00 Mb
PHEN 0.2  info : Ideal Texture Memory:     23440.00 Mb
PHEN 0.2  info : Peak  Texture Memory:     23440.00 Mb
PHEN 0.2  info : Texture Flushes:                 3
PHEN 0.2  info : Peak Textures #:                 2
PHEN 0.2  info : Texture Accesses:         12345350
PHEN 0.2  info : Texture Block Misses:          234
PHEN 0.2  info : ---------------------------------------------------
PHEN 0.2  info :                Texture Mip-Map Histogram
PHEN 0.2  info : ---------------------------------------------------
PHEN 0.2  info : 0.8|0.2|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|
PHEN 0.2  info : -----------------------------------------------------
```

These statistics can give you an overview of the texture access in your scene and can help pin-point problems with your network or your scene.

By default, the mrl_exr_file shader uses a 16Mb cache to store its texture memory (16384 bytes). This cache can be changed by setting the environment variable **MRL_TEXTURE_MEMORY** to a new value in bytes.

The total texture data is the number of megabytes that the frame needs to render. It encompases all texture pixels read during the course of the frame. In a perfect world with unlimited memory, your **MRL_TEXTURE_MEMORY** should be set to the value reported here.

Peak Texture Memory is the maximum amount of memory used at any one time to render. If the texture cache has become full, the value will be equivalent to your **MRL_TEXTURE_MEMORY** setting. If the cache never gets filled, it will be smaller.

Once the texture cache is filled, the mrl_exr_file shaders will flush the oldest accessed data to make room for the new data. The Texture Flushes setting reports how many times this had to be

done. The more number of flushes you have, the slower your scene will render.

Ideal Texture Memory is an average of the texture memory used divided by the number of texture flushes. It represents a guess of what your MI_TEXTURE_MEMORY should be set to for best and average performance.

Peak Textures # is the maximum number of textures read during the render of the frame.

Texture accesses is how many times the shader tried to sample the texture and texture block misses is how many times in those accesses the mrl_exr_file shader had to fetch a new texture block from disk. Ideally, you want the ratio between texture accesses and texture fetches to be as big as possible.

Finally, the texture mipmap histogram is a textual representation of what levels of the ripmap were accessed more often, with higher quality levels on the left. For example, for objects that are really far, you should fetch much lower quality maps.

Additional Render Globals Features

Motion Blur

In addition to all the Mayatomr motion blur options, mrLiquid supports the integer attribute **motionBlurType** in the mentalrayOptions node. When this attribute is added, motion blur can be controlled between forward motion blur (0), midpoint motion blur (1) or backwards motion blur (2).

Midpoint motion blur means that the current frame is considered to be in the middle of the movement for the frame. Forwards motion blur considers the current frame as the start for the motion blur for the frame. And backwards motion blur considers the current frame as the end of the motion blur for the frame.

While midpoint motion blur is arguably more correct, forwards motion blur can often be faster and less prone to issues with particles or similar dynamic elements.

mrLiquid's default motion blur is to use forwards motion blur.

Additional Shader Features

mrLiquid ships with a collection of shaders some of which help with IPR while others add actual helpful functionality. The shaders named as gg_* can be used either under mrLiquid or Mayatomr and are improved versions of those shaders freely available as part of mrClasses. Shaders marked as mrl_* are in general internal shaders that mrLiquid uses to provide all of its functionality and are not meant to be used by users.

gg_showinfo

This shader allows you to easily verify a huge number of settings in your objects or in your scene. It can be used to track rendering problems, texture filtering issues, memory leaks and can be overall very useful for debugging new shaders.

Here's a list of what it can show and how the other settings effect the results:

Triangles	Shows the triangle tesselation mental ray uses on an object with a colorful pattern.
Boxes	Shows how mental ray boxes the tesselation into different groups of triangles.
Orientation	Shows you the orientation of the surface (green is front facing, red is backfacing).
NdV	Shows the dot product of the surface normal and the view direction.
P	Shows the point position. The X,Y,Z position is mapped into colors based on the X, Y,Z min and max values.
N	Shows the (bumped) normal orientation. The X,Y,Z position is mapped into colors based on the X, Y,Z min and max values. Usually you will want to set min to -1, -1, -1 and max to 1,1,1.
Ng	Shows the geometric normal orientation (after displacements). The X,Y,Z position is mapped into colors based on the X, Y,Z min

	and max values. Usually you will want to set min to -1, -1, -1 and max to 1,1,1.
dPdu	Shows the derivative of P over the surface's u parameter (state->derivs[0]). Only valid on surfaces and subds with derivatives. The value is mapped into colors based on the X, Y, Z min and max values.
dPdv	Shows the derivative of P over the surface's v parameter (state->derivs[1]). Only valid on surfaces and subds with derivatives. The value is mapped into colors based on the X, Y, Z min and max values.
dP2du2	Shows the second derivative of P over the surface's u parameter (state->derivs[2]). Only valid on surfaces and subds with second derivatives. The value is mapped into colors based on the X, Y, Z min and max values.
dP2dv2	Shows the second derivative of P over the surface's v parameter (state->derivs[3]). Only valid on surfaces and subds with second derivatives. The value is mapped into colors based on the X, Y, Z min and max values.
dP2dudv	Shows the second derivative of P over the surface's u and v parameter (state->derivs[4]). Only valid on surfaces and subds with second derivatives. The value is mapped into colors based on the X, Y, Z min and max values.
Tangent U	Shows the U tangent used for bump mapping. Only valid on surfaces and meshes that have bump mapping tangents. The value is mapped into colors based on the X, Y, Z min and max values.
Tangent V	Shows the V tangent used for bump mapping. Only valid on surfaces and meshes that have

	bump mapping tangents. The value is mapped into colors based on the X, Y, Z min and max values.
ST	Shows the surface's s and t texture parametrization (usually badly named in 3d packages as uv parametrization).
dPds	Shows a calculated and discontinous derivative of P over s. The value is mapped into colors based on the X, Y, Z min and max values.
dPdt	Shows a calculated and discontinous derivative of P over t. The value is mapped into colors based on the X, Y, Z min and max values.
Barycentric	Shows the barycentric coordinates of each tesselated triangle.
Time	Shows the time of each sample. Only valid when motion blur is active.
Motion	Shows the motion vector of the intersection point. The value is mapped into colors based on the X, Y, Z min and max values. Only valid when motion blur is active.
Instance	Colors each instance object with a different color.
Object	Colors each shape object with a different color.
Label	Shows mental ray label or tag values with a different color.
Material	Shows each material (surface shader) with a different color.
Raster Unit X	Shows the raster unit or pixel footprint in screen X that mental ray calculates using

	mi_raster_unit (for camera rays only). The value is mapped into colors based on the X, Y, Z min and max values. As it is a very small value, you should set the max value accordingly.
Raster Unit Y	Shows the raster unit or pixel footprint in screen Y that mental ray calculates using mi_raster_unit (for camera rays only). The value is mapped into colors based on the X, Y, Z min and max values. As it is a very small value, you should set the max value accordingly.
EWA dPds	Shows the tangent vector for s that mental ray calculates for its elliptical filtering. The value is mapped into colors based on the X, Y, Z min and max values. As it is a very small value, you should set the max value accordingly. This will show you how broken elliptical filtering is in mental ray.
EWA dPdt	Shows the tangent vector for t that mental ray calculates for its elliptical filtering. The value is mapped into colors based on the X, Y, Z min and max values. As it is a very small value, you should set the max value accordingly. This will show you how broken elliptical filtering is in mental ray.
dNds	This is a calculated and discontinous derivative of N over s. The value is mapped into colors based on the X, Y, Z min and max values.
dNdt	This is a calculated and discontinous derivative of N over t. The value is mapped into colors based on the X, Y, Z min and max values.
Raster Area	This is the raster area calculated from mental

	ray's mi_raster_unit on camera rays only. It is somewhat equivalent to Renderman's area(P).
Raster on Triangle	This option colors objects in green whenever the projection falls onto the same triangle of state->point if Rx falls outside, colors red. If Ry falls outside, colors it blue.
Incidence In Tangent	This shows the incident vector (state->dir) in tangent space. The value is mapped into colors based on the X, Y, Z min and max values.
DsuDsv	This shows an approximation to calculate Du(s) + Dv(s) and Du(t) + Dv(t).
dPdx	This shows the ray differential of P over screen's x derivative. Works on all rays. The value is mapped into colors based on the X, Y, Z min and max values.
dPdy	This shows the ray differential of P over screen's y derivative. Works on all rays. The value is mapped into colors based on the X, Y, Z min and max values.
dNdx	This shows the ray differential of N over screen's x derivative. Works on all rays. The value is mapped into colors based on the X, Y, Z min and max values.
dNdy	This shows the ray differential of N over screen's y derivative. Works on all rays. The value is mapped into colors based on the X, Y, Z min and max values.
MipMapLOD	The values of dPdx and dPdy are combined as an area to select a single value to be used as a mipmap lod. mrl_exr_file will use this value to select the correct OpenEXR ripmap.

Memory Use	This value maps memory usage as a ramp between green and red. The value is mapped into colors based on the X, Y, Z min and max values, where a value of 1.0 is assumed to be 256Mb.

gg_cellnoise

This shader implements cellnoise, similar to that of renderman.

gg_vcellnoise

This shader implements cellnoise on three bands, similar to that of renderman.

gg_perlin

This shader implements Perlin's new improved noise without lattice inflections function and with optional periodicity.

gg_vperlin

This shader implements Perlin's new improved noise function without lattice inflections on three dimensions and with optional periodicity.

gg_worley

This shader implements Steve Worley's original noise function.

gg_vworley

This shader implements Steve Worley's original noise function on three bands.

gg_parallaxbump

This shader implements parallax bump mapping which is a form of bump mapping often used in games.

gg_noise

This shader implements the renderer's original noise function wth optional periodicity.

gg_noises

This shader implements fractal brownian motion and turbulent noise with filtering using frequency clamping. For more information, see Advanced Renderman.

gg_tracegroup and gg_select_tracegroup

These set of two shaders provide the functionality equivalent to trace sets or trace groups. That is, they allow reflections and refractions to be manually controlled by the user on an object-to-object basis. mrLiquid provides menus options to create these shaders automatically and add trace groups to almost any shader in your scene.

Trace Groups

mrLiquid provides the ability to create trace groups or trace sets. What this functionality gives you is a more fine grained control on what reflects, refracts or traces what. While Maya and mental ray offer the ability to turn off raytracing features on objects and instances, doing so effects the whole scene. For example, it is impossible to have one object appear in the reflection of another object but be excluded in the reflection of another one.

mrLiquid models its trace groups on Renderman trace sets, but further expands their functionality. With mrLiquid's trace groups, you can link trace features based on object lists, instance lists, instance groups (sets), and materials. That is, you can, for example, exclude all objects that are part of a set, or a certain material besides actually listing either each object or instance manually.

Tutorial

- Create a scene with a ball, a cube and a plane reflecting the ball and cube.
- Select the ball to reflect/exclude. Go to **mrLiquid->Trace Group->Create Trace Group**. Rename the trace group to something meaningful.
- Select the plane's material or object to have it recognize the previous trace group. Use **mrLiquid->Trace Groups->Use Trace Group-><name>**
- Play with the trace group settings (try excluding it to see the ball's reflection disappear).

> **Note:**
>
> Albeit mrLiquid's trace groups allows hiding objects from reflections, this is done with a shader trick, meaning that raytrace intersections are still performed. There's currently no speed benefit in rendering when using trace groups to smartly exclude objects from raytracing in your scenes.

Additional Light Features

Cropped Shadow Maps

mrLiquid provides you with additional light features on top of what Mayatomr offers. One of those features is the ability to crop shadow maps. Cropping shadow maps allows you to often use lower resolution maps than those that mentalray generates automatically based on the cone of the light and the scene size.

To crop the shadow map of a light, first make sure to create a light with a mental ray shadow map (Maya shadow maps will not work). Then, change one of your views to view the scene from the point of view of the light.

Finally, go to **mrLiquid->IPR->Crop Region Tool** and then, in the view with the pov of the light, draw a rectangle covering the area where you want the light to cast shadows. This will add some additional attributes to the light representing a crop region in float coordinates (between 0...1). If your shadow map area is smaller than the cone of the light, your shadow quality will dramatically improve without having to increase the shadow map resolution. Feel free to keep changing the light's shadow map settings, including resolution or animate the values of the crop region.

Inherited Light Attributes

An additional burden placed on custom light shaders is that the color of the light in the custom shader is not represented in the opengl drawing of the scene unless you carefully make the main light's color match either manually or through an expression.

mrLiquid supports an extra attribute called **miInherit** that you can add to a light shader (and by default it will get added automatically albeit due to a bug in the maya gui this may not be seen).

When this boolean attribute is on, any parameter in the custom shader that has the same name as a parameter from the main light's attributes will then be driven by the main light's attributes.

What this means is that if your custom shader has, for example, a “color” parameter, you can and should use the main light's “color” attribute to drive its color. A similar thing may happen with an “intensity” parameter.

This trick makes your opengl view match more closely what you get with the renderer at the end, without you having to write any expressions which saves time, memory and improves your scene's speed.

Note, however, that one drawback of this is that if you switch to render your scene back with Mayatomr, it will not recognize the **miInherit** attribute and will take the values for the custom shader from the main gui for the custom shader and not the light's attributes.

Object Lights

mrLiquid gives you a simple way to create object lights. That is, area lights whose shape is determined by another object in your scene.

To create an object light, under maya8.0 or earlier, create a point light.

For maya8.5 or later, create an area light.

Then go to the area light settings and switch the area light type to “User”.

To switch the user area light to an actual object area light, add a message attribute on the light's shape called **miAreaObject.**

To that attribute, using Maya's Connection Editor or the Hypergraph, connect the message of a valid object's shape, like a Nurbs Surface or a Mesh.

Light Sets

mrLiquid supports grouping lights in sets and passing those onto shaders or using those sets for selective lighting.

While maya offers light linking, using light sets can be more efficient on complex scenes, as the

light set has to only be spit once, while maya's light linking expands all lights for each object.

It is important that any light set used for lighting only contains lights or the results may be unpredictable, probably resulting in a crash of the renderer.

To create a light set, go to **Window->Relationship Editors->Sets**. From there, you can use select several lights and then click on the **+ (plus)** button to create a new light set for them.

Once the light set is created, you can use the **Window->General Editors->Connection Editor** to attach the light set to any shader parameter that takes a light list.

Particles

mrLiquid can render most of maya's particle types including: points, streaks, multipoints, multistreaks, sprites, clouds, blobbies and instances.

Currently, it does not handle neither tube, spheric nor numeric particles.

Unlike Mayatomr, sprite images can be animated.

mrLiquid's particle shaders, rather than embedding the particle data in the .mi file like Mayatomr, will use a maya particle cache. This means that if you have already created a particle cache for your simulation, mrLiquid can render the particles almost immediately, only needing to save a small .mi file.

Limitations:

- Instance cycles will not match maya unless an expression or a keyframe animation is used.
- Blobbies may not match Mayatomr exactly, as mrLiquid uses its own shader for the calculation.
- multipoints and multistreak particles may also not match the viewport exactly, as the random number generator is different.

Fur

mrLiquid allows you to render maya fur by doing a quick approximation of what you see on the screen. While the results may be different than those produced by the maya software or by Mayatomr, mrLiquid's algorithm is much faster.

Hair

Besides Mayatomr's standard hair rendering options, mrLiquid adds three additional hair generation methods. mrLiquid will automatically split hairs into different subobjects based on a threshold (a default of 100,000 hairs).

These methods can be accessed by the miHairType property in the instance of the hair object.

However, for some of these hair methods to work, it is needed that the hair was created with NURBS splines, and not just PaintFX hairs as it is the default in maya.

The different hair render types are:

Guide Hairs

This option renders just the spline guide hairs. It can be useful to render quick simulations of the hairs or during test renders.

This option requires hairs created with NURBS splines.

Emulated Clumps

This option generates hair like maya but using mrLiquid's internal algorithms. This often results in hair being generated faster than with Maya, but the results may not match exactly what you see in the Maya render or your viewport.

This option requires hairs created with NURBS splines.

Interpolated

Interpolated hairs is a different form of hair generation that instead of generating hairs in “clumps”, like Maya does, it interpolates hairs between 3 guide hairs. This type of hair generation is often much more efficient for flat hair or fur types.

This option requires hairs created with NURBS splines.

PaintFX Clumps

This option uses Maya's Paintfx engine to generate hairs. This makes hairs match very closely the output of the Maya render engine, but it can take longer to calculate. This is the type of hair that Mayatomr generates by default.

This option requires hairs created with PaintFX.

Additional Phenomena Features

One of the highlights of mrLiquid is the ability to easily create complex phenomena from any shading network in your maya scene. This helps to simplify your maya scene as well as it allows you to easily create a library of phenomena for special effects.

To create a phenomenon, open the hypergraph or hypershade and carefully select the node you want to use as the root of the phenomenon.

Then go to **Materials->Create Phenomena** or **Export Phenomena**.

If a valid root was selected, a popup will appear with all the shaders further up the tree as well as the parameters for them. From this requester, you can choose which parameters you want to expose as the interface to your phenomenon. The default behavior is to expose ALL parameters.

mrLiquid's phenomena .mi file gets saved into your user's presets directory (ie. the value of **internalVar -userPresetsDir**). A custom user interface for the phenomenon created is saved in your user script directory (ie. the value of **internalVar -userScriptDir**).

The difference between **Create** and **Export Phenomena** is that the export option just exports a new phenomenon while the create phenomenon will export and replace all your shaders with a single node corresponding to the newly created phenomenon.

If you want to share your phenomenon with other users, it is recommended you move the phenomenon's .mi file and its .mel interface to some public .mi and script directory respectively.

Note that due to the way Maya refreshes its MEL AETemplates, you will not see a change in the interface if creating a phenomenon multiple times with different user parameters. You will need to exit and restart maya for the changes to be visible.

Limitations:

mrLiquid's phenomena can be relatively complex, as textures, lights, cameras, multiple shaders and multiple shader types are supported. Note that some of these features are not totally exposed properly in Mayatomr, so some of the phenomena mrLiquid can create may not be renderable in Mayatomr.

Also note that any string parameter with mrLiquid expressions like ${OBJ} in them, will be baked in into the phenomenon and will not be able to change on the fly anymore.

The **Material->Expand Phenomena** allows you to expand one or more phenomenon in your scene to retrieve the shading network corresponding to the phenomenon, roughly doing the opposite of Export Phenomena. Note that albeit mrLiquid is pretty smart about expanding phenomena, there may still be some phenomena that it is not able to expand successfully.

PaintFX

mrLiquid can render most types of PaintFX, including those using lines and meshes. It will try to use shaders that closely emulate the look that the maya renderer uses.

mrLiquid will not render PaintFX's glows or those PaintFX properties that do 2d effects like blur or smears.

Render Layers

mrLiquid takes advantage of Maya's render layers, but it also adds several benefits to it, including manual MEL overrides.

By default, when mrLiquid starts, it will automatically create several default render layers. The render layers created can be controlled in the file **mentalSetupDefaultRenderLayers.mel**.

The default mental ray layers are for debugging and optimization of your scene, like a tesselation render layer and are not renderable by default.

mrLiquid also ships with several simple predefined layer types, which you can manually create by accessing the **mrLiquid->Render Layer->Create** menu. These render layers will automatically try to parse your scene and assign the needed shaders or changes to the render globals.

Render Layer Overrides

In addition to all the maya overrides that you can generate for each render layer in recent maya versions, mrLiquid also allows the creation of layer overrides.

By selecting any render layer and adding a message attribute called “mrOverrides”, you can then connect mrLiquid render layer overrides.

mrLiquid's mentalRenderLayerOverride nodes consist of pre and post commands or actions. These commands are run right after maya assigns the shader and render global overrides.

Pre commands are free to modify the maya scene any way they see fit. However, it is important that for each pre command created, an equivalent post command is created to try to revert the changes.

As they are based in scripting, mrLiquid's render layer overrides can be a tad more cumbersome to use than maya's render layer overrides. However, they can be much more efficient when a change need to effect hundreds of objects (for example, assign the same shader to all objects in the scene).

Also, mrLiquid's overrides can modify things the maya render layer overrides cannot currently modify, like camera render settings.

Baking Geometry

As your scene complexity increases, it is not uncommon to find your maya scene taking a long time to open or worse, not being able to fit in memory anymore.

To alleviate this problem, mrLiquid offers the ability to bake geometry into static .mi files, animated object .mi files or animated assembly .mi files (maya 3.6).

Static .mi files are simple include files and will only improve the opening of the maya scene. They can contain any type of geometry, shader or primitive that mental ray supports. When an .mi include file is loaded by mental ray, it is all loaded in memory immediately, as if the elements had been created by the translator.

Object .mi files, on the other hand, allow them to be used as file objects which allow mental ray to flush each of the objects in and out of the scene. They are somewhat equivalent to Delayed RIB Archives in Renderman, but only a single object is stored in them. Also, other elements like lights cannot be stored.

In mental ray 3.6, assemblies were introduced. Assemblies are somewhat equivalent to Nested Delayed Rib Archives in Renderman. They allow any combination of lights, cameras, shaders and objects. Like mental ray file objects, the renderer can flush them in and out of the scene as memory requires.

To bake geometry, go to **mrLiquid->Bake->Geometry**. A requester will pop up allowing to select what type of baking you want to do (Include Files, File Objects or Assemblies) and whether you want to bake an animated sequence. You can then also choose to delete/replace the baked objects in your scene.

If you bake a set of objects as file objects or assemblies and replace them, mrLiquid will create some special shape types called mentalFileObject or mentalFileAssembly which should draw a bounding box corresponding to each object baked (or all of the objects in case of an assembly).

Environment Variables

mrLiquid supports some environment variables to control its behavior.

All of these can be set in the shell window, user environment or within Maya by using putenv().

The variables are:

VARIABLE	*USE*
MRL_NO_MAYA2MR	When set, use mrLiquid as Mayatomr replacement.
MRL_RAY	Location and name of mental ray stand-alone executable.
TEMP	Location of user's temporary directory.
MI_CUSTOM_SHADER_PATH	Location of additional .so/.dll/.mi shaders, including those of mrLiquid.
MI_ROOT	Install location of mental ray stand-alone.
MAYA_LOCATION	Install location of maya.
MRL_SWATCH	When set, mrLiquid will use an additional stand-alone license to generate the swatches when MRL_NO_MAYA2MR is set. Otherwise, no effect.
MRL_VIEWER	If set, it specifies the image viewer to use to monitor renders as they happen. This viewer must be capable of reading OpenEXR files as well as connecting to mental ray by using one of its stub files.
MRL_MIPMAPS	If set to a value of 1, it will set mrLiquid's default behavior of handling textures by creating mipmaps and ripmaps. The mentalrayOptions' boolean makeMipMap overrides this default setting.
MRL_EXR_FILE	If set to a value of 1, mrLiquid will render textures using its own mrl_exr_file shader, replacing all maya_file shadre nodes.
MRL_TEXTURE_MEMORY	This integer value controls the amount of texture memory to use for ripmaps when using the mrl_exr_file shader.
MRL_USE_CCMESH	Set this to 1 or 0 to determine whether meshes with subdivision approximations are spit as ray3.5+ ccmesh (Catmull-Clark) meshes or as mentalray hierarchical

VARIABLE	*USE*
	subdivision. If not defined, the default behavior is to emulate the default behavior of Mayatomr for that maya version.

Before loading mrLiquid or Mayatomr, the environment variable:

```
MRL_NO_MAYA2MR
```

can be set. When this variable is set and mrLiquid is loaded, it will NOT attempt to load Mayatomr. This allows you to use mrLiquid as a complete replacement to it. In this mode, the "mental" command will be registered as "Mayatomr" as well as all other commands and Mayatomr nodes. Do *NOT* try to load Mayatomr afterwards, as it will crash maya.

Starting with v0.8 of mrLiquid, several standard maya MEL scripts that for maya's gui that are hard-coded are provided as replacements to also support mrLiquid.

The Menus

Upon loading the plugin, you will find a new mrLiquid menu in the Rendering section of Maya.

The first option in the menu is:

Render Globals

This option brings up mrLiquid's render global settings panel.

Approximation Editor

This option brings up mrLiquid's approximation editor. Similar to Mayatomr's Approximation Editor, it also allows you to create flagged approximations.

Render Layers

Within this submenu, you will find several common operations to be used on render layers.

- ***Create***

  - ***Default Render Layers***

    This option creates mrLiquid's default render layers. These render layers are the same that are created upon launching a mrLiquid render. They are determined by the content of the **"mentalCreateDefaultRenderLayers.mel"** file.

  - ***Ambient Occlusion***

    Adds an ambient occlusion render layer. An ambient occlusion render pass is a raytracing pass that measures how visible a point in a surface is taking into account an hemisphere around the normal of the point being shaded. This can be used to create darker shadows in crevices, soft contact shadows and dirtmap passes.

  - ***Beauty***

    Adds a beauty render layer. A beauty pass is usually the final render layer run and the one that should create a final image for compositing or for outputing to film or video.

- ***Cast Shadows***

A cast shadow render layer is one that only renders the shadows of objects as a black and white or colored image. This is often useful in compositing to properly add the shadow of a 3d object to a background image shot on stage or a set.

- ***Compositing***

A compositing render layer is one that only composites images or samples passes from previous renders. Its main use is to manage scenes of large complexity or to use your 3d package as a very simple compositing package.

- ***Final Gather***

A final gather pass is one that just renders out the final gather (global illumination) contribution of the scene. Final gather, as all forms of global illumination, can often be a time consuming render, so isolating it as a separate render layer can be beneficial

- ***Shadow Maps***

A shadow map pass is one that pre-calculates all shadow maps in your scene, saving them to a disk file. If you have multiple beauty render layers or scenes of great complexity, this can speed up your render iterations.

• *Set Render Layer*

Similar to changing layers through maya's layer editor, this submenu allows you to easily switch from one render layer to another.

• *Validate Render Layers*

mrLiquid's render layers support, in addition to all the features of maya's render layers, mel overrides. This option will iterate through all render layers and verify that their mel override is valid mel code.

• *Create Render Layer Scenes*

This option allows you to save each renderable render layer with all its objects as a different maya scene. This feature is probably no longer used much but it can still be useful to manage very large render scenes, allowing you to conserve some memory.

- ***Export Render Layers***

  This option allows you to save each renderable render layer as a maya scene. This feature allows you to easily share custom render passes among different users, shots, or projects.

Samples Pass Files

Here you will find options to manage mrLiquid's samples passes. Samples passes are somewhat akin to saving out multiple frame buffers, but instead of creating images they create samples pass files. These samples pass files can then be composited or operated upon in a similar way as images, but since the operations are done on samples, it can avoid some problems often seen in depth composites. Note that, unlike rendered images, the sample pass format is proprietary to mental images, so other applications will probably not be able to read it.

- ***New Pass File***

  Creates a new pass file. By default, the pass file receives the name of the node and it is added only to the current render layer.

- ***New Merge Pass File***

  Creates a new merging sample pass. By default, this pass is added only to the current render layer and it will composite all previously created sample pass layers.

Frame Buffers

Within this submenu you will find options to manage mrLiquid's frame buffers. Frame buffers are additional images that your renderer will create while rendering. These additional frame buffers can be used for special effects as well as for saving out new images to isolate different parts or render layers of your final image. For some types of render passes and with appropiate shaders, they can be used to avoid having to re-render the whole scene each time as a new render layer.

- ***New Frame Buffer***

  Creates a new frame buffer. This frame buffer is not saved to disk.

- ***New Frame Buffer and Output***

  Creates a new frame buffer and a corresponding output pass to save the frame buffer to disk.

Trace Groups

Under this submenu, there are several options to allow you to control trace groups. Trace groups allow you to have some objects reflect or refract others in a selective manner, giving you more control than just turning on or off an object's render flags and avoiding multiple render layers.

- ***Create Trace Group***

  This option creates a new trace group for every selected object or instance.

- ***Use Trace Group***

  This submenu allows you to apply a trace group to the selected material or objects.

Lights

Here you will find options useful for dealing with lights in your scene.

- ***Show Saved Shadow Map***

  This option allows you to view, in a separate viewer, the saved shadow map of a selected light. This image will also update interactively upon IPR changes or similar.

- ***Toggle Area Lights***

  This option will toggle on and off all the area lights selected (or all of them if none are selected) for a faster preview. Note that if you save your scene with the area lights turned off, this option cannot revert those lights back to area lights.

- ***Toggle Shadows***

  This option toggles on and off the quality of shadows for faster previews. Note that if you save your scene with the shadows at a low quality, this option cannot revert those lights back to the original quality.

Materials

From within this submenu, you can invoke several operations to deal with materials (shading groups and shaders).

- ***Create/Export Phenomena***

  Allows you to group any shading engine or shading network selected into a single phenomenon node. When run, a requester will appear allowing you to select which parameters to expose as part of the phenomenon and whether you will also want to replace the shading network with the single phenomenon.

  Phenomena are created within the user's presets dir (internalVar -userPresetsDir) and the corresponding MEL interface scripts are created within the user's scripts dir (internalVar -userScriptsDir).

- ***Expand Phenomena***

  Replaces selected phenomena with the corresponding shading network.

- ***Export Materials***

  This option will create a new maya scene containing all the current maya materials. It can be useful to share materials across different projects or users.

IPR

From this submenu you control mrLiquid's IPR (Interactive Photorealistic Render), which is a form of interactive render preview.

- ***Normal***

  Starts or switches to a normal IPR mode, using all settings as defined in your render globals.

- ***Two-Step***

  Starts or switches to a two-step IPR mode. The first pass is a rough pass as defined in your IPR options and the second pass corresponds to your settings in your render globals.

- ***Progressive***

Starts or switches to a progressive IPR mode. IPR begins as a rough pass with the settings defined in your IPR options and progressively keeps re-rendering until it reaches the settings used in your render globals. Any change you do in the scene while the render is running resets the process.

- ***Crop Region Tool***

This tool allows you to easily create a crop region for any of your render cameras for previewing or for any light with shadow maps for speeding up their shadowmap generation and quality.

- ***Options***

Brings up the IPR render options. From within that requester you can specify what the IPR rough pass means and whether IPR will consider rendering all layers and renderable cameras.

- ***Image Viewer***

Opens up a new image viewer, showing the IPR image in progress.

- ***Pause IPR***

Allows you to pause the IPR.

- ***Continue IPR***

Allows you to continue the IPR after it has been paused.

- ***Stop IPR***

Stops the IPR.

Render

From this submenu you control mrLiquid's Renders.

- ***Render Current Frame***

  Starts a render for the current frame.

- ***Stop Render***

  Stops the render of the current frame.

Batch

From this submenu you control mrLiquid's batch rendering.

- ***Create YAML File***

  mrLiquid's ruby render scripts can use a .yaml file for submitting jobs in batch. A .yaml file is a very simple text file that contains key->value pairs.

  This option allows you to easily create such a .yaml file. The yaml file is saved in the same directory of your scene file.

  For more information on ruby's batch render scripts see BATCH RENDERS section.

- ***Render Timeline***

  Creates a YAML file and then launches mrLiquid's ruby render scripts in the background to render the timeline. This will actually fire up a separate maya in the background which will then load your scene again, so it can be memory intensive.

Bake

From here, you can create different baking passes. Baking allows you to pre-calculate textures or geometry for speeding up rendering or to allow handling larger scenes.

- ***Textures***

  Bakes shading and lighting to textures, in a similar way to Mayatomr. However, unlike Mayatomr, you can also bake frame sequences.

- ***Geometry***

  Bakes geometry objects into .mi files for file object reads or for $include.

Console (Windows only)

- ***Clear Console***

During verbose renders, your maya console can easily get filled with messages. This option uses the windows api to clear the maya console window.

Unload

- ***Unload mrLiquid***

This option allows you to unload the mrLiquid plugin. Note that any specific mrLiquid render node will be removed.

- ***Unload Mayatomr***

This option allows you to unload both the mrLiquid plugin as well as Mayatomr. Note that any specific mrLiquid or Mayatomr render node will be removed.

About

This option will give you some information about mrLiquid and rubyMEL (version, programmers, beta-testers, etc).

Variables in String Expressions

All string expressions (shader paths, output images, string parameters, requesters, custom text, etc) that mrLiquid translates can also accept a number of variables. These are:

VARIABLE	*DESCRIPTION*
$PDIR	Current Project Directory
$SCN	Scene name (without .ma /.mb extension)
$RNDRPASS	Name of current render pass
$PREVPASS	Name of previous render pass
$RNDR	Render directory (-output flag)
$TXT	Texture directory (-textureDir flag)
$MIDIR	Directory for mi files (-miDir flag)
$FGMAP	Directory for fg files (-fgmapDir flag)
$SHMAP	Directory for shadow map files (-shmapDir flag)
$PHMAP	Directory for photon map files (-phmapDir flag)
$NAME	Short name of current entity being translated (shader name, usually)
$OBJNAME	Name of current object being translated. If object is |pCube1|pCubeShape1, $OBJNAME is "pCubeShape1"
$OBJPATH	Path of current object being translated. If object is |hello|pCube1|pCubeShape1, $OBJPATH is "|hello|pCube1"
$INSTNAME	Name of current instance being translated. If object is |hello|pCube1|pCubeShape|, $INSTNAME is "pCube1".
$INSTPATH	Path of current instance being translated. If object is |hello|pCube1|pCubeShape|, $INSTPATH is "|hello".
$LABEL	Label of current object being translated as specified by miLabel attribute.
@	Current Frame. Multiple @@@ indicate a padded frame number.
${...}	Any MEL global variable. So, ${gMentalRay} would refer to the global MEL variable $gMentalRay.
$(...)	Any Environment variable. So $(MI_DIR) would refer to the environment variable MI_DIR.

You can also use Maya7's %character syntax to specify:

VARIABLE	*DESCRIPTION*
%c	Camera name
%l	Render Layer name (same as $RNDRPASS)
%s	Scene name (same as $SCN)
%i	Instance name (same as $INSTNAME)
%o	Object name (same as $OBJNAME)
%[0-6]n	Frame number (same as one or more @)

Special Attributes

These are attributes you can create to give you additional functionality not present in Mayatomr. If mrLiquid is run without Mayatomr, these attributes will often be created automatically.

If running Mayatomr, you create these attributes using addAttr, like, for example:

```
addAttr -at bool -ln makeMipMap mentalrayOptions;
addAttr -multi -at message -ln miApproxList nurbsSphereShape1;
```

Some of the menu options in mrLiquid's menu may also create some of these attributes automatically.

Node	Attribute	Type	Description
miDefaultOptions	makeMipMap	bool	if available and set, every texture read will be automatically converted to a mipmap .map file during scene translation. The translator will automatically output the name of the resulting .map file and will keep comparing the dates of the .map file against the original file to re-update the .map file if needed.
	shadowMapOnly	bool	Specifies that the render (or pass) will render only shadow maps.
	motionBlurType	int	Specifies the direction of how motion blur is calculated. 0 - Forwards motion blur 1 - Midpoint motion blur [Default] 2 - Backwards motion blur Suggestion: use Forwards Motion Blur as it can help speed up translations in some cases.
	lightMapActive	int	Allows you to change the light map settings. Values: 0 - lightmapping is off. 1 - lightmapping is on. [Default] 2 - lightmapping is only thing rendered.
	photonMapOnly	bool	Allows you to render just photon maps.
	photonAutoVolume	bool	Allows you to turn on/off autovolume for photonmapping.
	regionRectX	int	
	regionRectY	int	
	regionRectWidth	int	
	regionRectHeight	int	Allows you to set a fixed rendering crop region for rendering in batch. Values are in pixels.
	exportFluidFiles	bool	If set, it allows you to spit fluid data files onto $TEMP for slightly faster fluids, instead

Node	Attribute	Type	Description
			of into the .mi2 file.
Approximation nodes	miTrace	bool	
	miVisible	bool	
	miShadow	bool	
	miCaustic	bool	
	miGlobIllum	bool	These set the approximation as a flagged approximation
	noSmoothing	bool	For fine approximations of meshes, turn off smoothing.
Nurbs, Mesh, Subd Shape	miApproxList[]	multi message	allows you to hook flagged approximation nodes (any type: surface, diplacement, etc). Note that no checking is done to verify those nodes are valid.
	miLabel	int	Allows labeling (tagging) the shape (not the instance)
Transforms	miFinalGather	bool	
	miReflection	bool	
	miRefraction	bool	Allows controlling these in the instance. They work exactly like the miGlobillum flag of the transform.
Custom Light Shaders	miInherit	bool	This gets created automatically, but there's a bug in maya7's AE editor that will have it not appear. This boolean flag, if set, allows your custom light shader to inherit parameters from the light's shape if those parameters are named identically. For example, if your custom shader has a "color" attribute and miInherit is on, the "color" parameter of the shader is not used, but the one in the light's shape is. The benefit of this is that you can get your opengl lights' preview to match your final render preview, even with custom shaders.
Point Light	miAreaObject	message	Connect a mesh/subd/nurbs transform to the light and make

Node	Attribute	Type	Description
			the point light's area type be of type user. This allows you to have an object area light.
Shading Groups and Shaders	miMultiShader	bool	When this flag is set, the shading group/shading network will be treated as a multi-object shading network. What this means is that each object will get a copy of the shading network, named: shader-objectname The main reason for this is to allow $OBJNAME expressions. In fact, as soon as mrLiquid detects an $OBJNAME or similar expression in some shader parameter, it will automatically create the attribute and set it to on for other objects (or for future translations). NOTE: this flag is still not IPR friendly as of yet.
Cameras	miLensShaderList	multi message	attach a list of lens shaders to this attribute.
	miSamplesPassList	multi message	attach a list of render pass nodes (samples passes) to this attribute.
Render Layer	mrOverrides	multi message	attach a list of mentalRenderLayerOverrides nodes to a render layer.

Batch Renders

To do batch renders, you can use either maya's standard render command or the included Ruby Rendering Framework.

Maya's Batch Render Framework

You can use the standard maya command:

```
render -r mr2  <opts>...
```

or:

```
render -r mi2  <opts>...
```

to start a render of mrLiquid with maya open (mr2) or to spit .mi files (mi2). All other options should be equivalent to Mayatomr's render -r mr and render -r mi.

When using the maya batch render framework, all settings are taken from the scene file as it was saved.

Ruby Render Framework

This framework uses a single .yaml file (which is a very simple text file of “key -> value” pairs) to contain all the settings used in renders (and potentially, comps).

It currently works for rendering with maya, maya2mr and mrLiquid's mental command and it allows changing on the fly most settings, without you having to worry about opening and re-saving the maya file.

The main interest in using a yaml file is to simplify the number of flags passed to a render command. Also, it simplifies the use of the command over the render queue for large facilities or for a large number of users.

To use the ruby rendering framework, you need to install rubyMEL first.

This includes the rubyMEL plugin, the RubyMaya (RubyMaya.rb/mayalib) module for ruby, and

the assorted ruby scripts for rendering.

If you followed the automatic install, you should have the scripts:

```
RNDR.rb
ALL.rb
MAYA.rb
```

inside rubyMEL's **bin/** directory. You should eventually set your **PATH** environment variable to point to them, so they are easy to access.

You should also have a **demo/** directory with a simple scene and a sample YAML file.

So.... to run the demo, you should open a console or shell window and run (for Unix):

```
> cd <rubyMEL dir>
> source maya<maya_version>/environment.sh
> cd demo/
> ../bin/RNDR.rb SHOT.yaml 5
```

The above should render a single frame (5) of a simple sphere animation. For windows, replace the “source ...” line with “envinronment.bat”.

Under windows, it is very important to try to use the correct ruby version (as set in the environment file), as each maya version links against a different (and incompatible) compiler version. Using a ruby version with an incompatible compiler can lead to unpredictable behavior or crashes.

If you don't get a render, see the section on **Aura/Facility.rb**. You probably have installed maya or ruby somewhere different than the default location.

The actual frame(s) will be placed inside a special temp directory, which is named with the process id number and inside a subdirectory with the name of the pass rendered. This allows you to run tests and keep all versions.

The other reason for rendering to a local temp directory is that it helps the network when working with a render queue.

By default, the Ruby Rendering Framework is set up as if Rush was your render queue so it will automatically assume you are rendering on the network if RUSH_FRAME is set.

The **Facility.rb** file contains settings that easily allow changing those defaults to support other render queues.

Inside the **lib/** directory of mrLiquid, you will also find two additional yaml files, named **FACILITY.yaml** and **SHOW.yaml**.

These two files are used to set FACILITY and SHOW settings. Usually, you put the **FACILITY.yaml** file somewhere in your network where all machines can see it.

And you put the **SHOW.yaml** also in the network, but in a place specific to the show or commercial you are working on. That way, you can have multiple commercials or shows rendering with different settings (image resolutions, etc).

All the settings established in **FACILITY.yaml** or **SHOW.yaml** can also be overriden in the **SHOT.yaml** file, to have specific shots with specific settings.

RNDR.rb

Okay, so you rendered a simple demo frame. Let's say you want to move the frame to its final destination. This can be done by using:

```
> RNDR.rb -m SHOT.yaml 5
```

The frame will still be rendered first into the temp directory. But, afterwards, the frame will be moved to its final destination (by default: renders/layer/).

To change the directory destination, in your .yaml file you can use:

```
RNDR:
  dir:
```

(see **SHOT_FULL.yaml** or the chapter YAML Reference).

The default dir and prefix settings are stored in the **FACILITY.yaml** file.

Also, in that file, the **RENDER_FRAME** setting is the name of an environment variable that can be used in batch renderqueue renders to automatically set the -move flag. By default, this is set to RUSH_FRAME, to detect when the script is running under Greg Ercolano's Rush render

queue. You will probably need to change this to set it to your facility's render queue.

The image prefix is set up by default to be:

```
SHOT.version.layer  if the SHOT environment variable is defined
scene.version.layer if not
```

This gives you renders like:

```
ta320.1.beauty.0020.exr
```

By default, when images are moved, they will be placed in:

```
renders/#{layer}/#{res}/
```

where layer is the name of the render layer and res is one of 'hi' or 'lo'.

What exactly means to be 'hi' or 'lo' is determined by what you usually set in the **SHOW.yaml** file. By default, hi is set to film res 'Academy' and lo is set to 'NTSC'.

What Academy and NTSC means in terms of resolution, pixel aspect, etc. is defined in the **FACILITY.yaml** file. In general those are relatively common terms which you should not need to modify.

The RNDR.rb command supports taking frame ranges and frame numbers as parameters. The syntaxes supported are:

```
START-END
START to END
Example: RNDR.rb 10-15  20 to 25
         [10, 11, 12, 13, 14, 15, 20, 21, 22, 23, 24, 25]

START-END/STEP
START to END step STEP
Example: RNDR.rb 10-20/5  30 to 50 step 5
         [10, 15, 20, 30, 35, 40, 45, 50]

START-END batch BATCH
Example: RNDR.rb 10-20 batch 5 14 34 12
         [10, 15, 20, 34]  (14 12 are ignored as are in batch)

START-END countdown DOWN
Example: RNDR.rb 10-20 down 10
         [10, 20, 15, 12, 14, 16, 18, 11, 13, 17, 19]
```

Statistics

As the different passes of the render finish, you will see messages such as:

```
------------------------------------------- J: 00:00:07 T: 00:00:19
```

These indicate the JOB time (for example, spit mi files) and the TOTAL time so far.

Finally, at the end of the render, you will some simple timing statistics:

```
RNDR: ------------------------------------ J: 00:02:50 T: 00:03:10
RNDR: TOTAL RENDER TIME:  00:02:50  FRAME AVG:  00:00:08
RNDR: ---------------------------------------------- T: 00:03:10
```

Debugging Batch Renders

All the .rb commands support several debug levels. To change the debug level, change the environment variable **DEBUG_LEVEL** to a number from 1 to 9.

You can do that with (cmd.exe, for example):

```
> set DEBUG_LEVEL=5
```

Under bash, you would do:

```
> export DEBUG_LEVEL=5
```

Finally, RNDR.rb -h will give some basic usage description.

MAYA.rb

This script allows you to open maya with the scene taken from the .yaml file provided and with all of its environment variables set in the same way as the RNDR.rb command does.

This is just a simple convenient utility.

ALL.rb

This script runs multiple commands. For example, if you created several scenes, run as RNDR, RNDR2, etc. ALL.rb allows you to run all of them.

Aura/Facility.rb

Okay... if you move the SHOW.yaml or FACILITY.yaml to a new location, you should edit the Ruby file "Aura/Facility.rb" to tell the ruby scripts where to look for that stuff (among other things).

The "Aura/Facility.rb" file should be placed inside your ruby installation.

With the windows automatic install, for example, this is in:

```
<mrliquid install dir>/lib/ruby/1.8/Aura/Facility.rb
```

This file also contains other settings, like what maya version to use by default or where maya is installed.

Creating a YAML file from Maya

If you have rubyMEL properly installed, you can create a .yaml file from maya directly for batch rendering.

Go to:

mrLiquid->Batch->Create YAML file

A new .yaml file will be placed in the same directory where you saved your scene file. You can then execute that file using the RNDR.rb command.

Alternatively, you can also just do:

mrLiquid->Batch->Render Timeline

which will proceed to create the .yaml file and then run the RNDR.rb command locally on your machine.

Note that this means having two maya open, so it can be heavy memory wise and use an additional batch license.

YAML Reference

```
# This a complex and dummy example showing all the options available.
# Settings set here override settings set in SHOW.yaml and FACILITY.yaml.
#
# Setting in RNDR or RNDR2 override settings in ALL.
#
# Common settings to all passes
ALL:
      res:  lo              # resolution to render (either 'lo' or 'hi')
                            # what 'lo' and 'hi' means is defined
                            # in SHOW.yaml
      frames:     1-20          # frame range to render
      passes: ['RNDR', 'RNDR2'] # passes to render (commands: RNDR.rb, RNDR2.rb)

#
# Render pass, using command RNDR.rb
#
RNDR:
      # Scene to render
      # Can be a full path to the scene or a scene file inside the
      # MAYA_PROJECT environment directory.
      scene:            ball.ma

      # Renderer to use (maya, mental, maya2mr, etc)
      renderer:   mental

      # Render thru .mi file
      mifile:      1

      # Export verbosity
      export_verbosity: 2

      # Render verbosity
      verbosity:      2

      # List of render layers
      layers:      ['beauty', 'shadow']

      # List of cameras to render
      cameras:    ['persp', 'top']

      # Directory where to place images
      dir:      images

      # Prefix for image names (you can use simple ruby expressions)
      # The default prefix leads to images named like:
      #     ball.1.amboccl.0200.exr
      prefix:      "#{scene}.#{version}.#{layer}"

      # Overrides for any maya attr.  These will be run using mel's setAttr
      # setAttr miDefaultFramebuffer.format 7;
```

```
      # setAttr defaultRenderGlobals.useMayaFileName 0;
      overrides:
            miDefaultFramebuffer.format: 7
            defaultRenderGlobals.useMayaFileName: 0
#
# Render pass, using command RNDR2.rb (usually a symbolic link to RNDR.rb
# but it can be a completely different command)
#
RNDR2:
      # Scene to render
      scene:      ball_two.ma
      renderer:   mental
#
# Comp pass, using command COMP.rb (not written yet)
#
COMP:
      # Script to render (Nuke)
      script:           ball_two.nk
```

Table 1: Comprehensive list of YAML attributes

Known Bugs

- Cropping tool is not 100% accurate under some camera Fit options.
- You can animate the showing/hiding of objects if doing incremental saving of .mi files (ie. several frames into a single .mi file), but the objects need to be visible in the first frame or you need to spit all objects (non-visible ones too).

  Workaround: either spit all non-visible objects or spit frames into separate .mi files.
- If using multiple renderers in the same scene, there is the potential for conflict. Some renderers add their own set of attributes to the light nodes and shaders.
- Tube Particles are not supported yet.
- Particles Numeric and Particles Spheres are not supported.
- Particles Cycle Objects do not match maya due to a different random number generator. They will match if an expression is used, thou.
- Changing resolution to a bigger resolution than the one we started with during IPR crashes imf_disp (mray imf_disp bug).
- If running as a stand-alone without Mayatomr, mrLiquid's mapVisualizer does not know how to display photon information.
- Under some Maya versions (mainly 2008), unloading Mayatomr can result in a crash. This has been traced to a bug in the OpenEXR library.

FAQ

Q: How do I do a render or IPR only selected objects?

The short answer is that you don't. For IPR, use **Display->Hide Unselected Objects** (alt+h by default) and **Display->Show Last Hidden.**

For final renders, create a render layer with the objects you want to render.

If scripting, you can use “mental -active” to spit out an .mi file with active objects only.

Q: How do I do test renders with smaller resolutions (like Render View: Options->Test Resolutions)?

Just change your resolution in the common Render Globals (Render Settings). You can do this even while in IPR. Note, however, that imf_disp has a bug that will crash it when resolutions are changed, so you may need to reopen it if imf_disp is your render viewer. mrViewer does not suffer from that bug.

Troubleshooting

Q: How do I use the license server across a firewall?

A: If you want to serve licenses across a firewall, generally you will need the license servers to have known port numbers in order to allow your firewall to pass requests on these ports.

The license server consists of two servers: the rlm server and the filmaura server.

The rlm server itself is always at a known port number (contained in the license file on the SERVER or HOST line).

Typically, rlm starts up the filmaura server with dynamic port numbers which are not known before startup time. Requests to the filmaura server are local to the server machine, so enabling all ports on the server machine for requests at 127.0.0.1 is the simplest way to fix the issues.

However, for a more paranoid approach, it is possible to have RLM assign fixed port numbers to any of the filmaura server.

In order to do this, you need to specify the port number for the ISV server on the ISV line. The port number is the fourth parameter in the isv line:

ISV filmaura [filmaura.opt] [port-number]

In order to specify the port number, you must also specify an options file for this filmaura server.

Once you have specified the port number, instruct your firewall to allow connections to both the port number on the SERVER line (for rlm) and the port numbers on any filmaura lines.

Q: On Windows, RLM fails to start the ISV server with a select() error. What's wrong?

A: rlm fails to start the filmaura server, with errors similar to the following written to the rlm log:

```
03/16 04:12 (filmaura) select() failure: Unknown error
<last error repeats many times>
03/16 04:12 (filmaura) Too many errors on main socket, exiting
```

This can be caused by corruption of some values in the Windows registry. Film Aura has no

reason to believe that it is responsible for this registry corruption.

Microsoft has published the following article on how to correct the registry when this occurs.

Note that the error message indicating the problem is different in the article than the RLM error message indicating the problem, but the underlying cause is the same.

http://support.microsoft.com/default.aspx?scid=kb;en-us;817571

With newer versions of XP, the bug can also be fixed by running:

```
> netsh winsock reset catalog
```

and then rebooting.

As an additional help, rlm on windows now ships with a small utility called WinsockFix, which when run allows you to fix this and other winsock problems.

Q: mrLiquid complains about a IlmImf.dll or IlmImf.so

A: This most likely means a conflict between an IlmImf.dll version you have installed somewhere and the one shipped with mrLiquid. You can fix the conflict if you make sure the **PATH** (Windows) or **LD_LIBRARY_PATH** (Unix) variables list the directories of mrLiquid first.

Alternatively, you can also try removing or moving the conflicting library. Or removing the path to the other library from those environment variables before starting Maya or mrLiquid.

Q: Maya does not load or find mrLiquid's shaders

A: This is often a problem with your Maya.env file or the value of MI_CUSTOM_SHADER_PATH.

A known bug in Maya makes it not override MI_CUSTOM_SHADER_PATH if such a variable has been defined previous to starting maya. Undefine MI_CUSTOM_SHADER_PATH and Maya.env's settings should take over. Or optionally, add the path listed in Maya.env to your configuration script that sets MI_CUSTOM_SHADER_PATH.

You can further verify Maya loaded the proper shader path, by typing in the script editor:

```
getenv MI_CUSTOM_SHADER_PATH;
```

If mrLiquid's install location is not present, something is broken in Maya.env or your MI_CUSTOM_SHADER_PATH value.

Q: Mayatomr cannot load mrLiquid's shaders

A: This is often a problem with your maya.rayrc file. Make sure that Mayatomr is reading the correct maya.rayrc file when it is first loaded. Watch out for any errors that show up.

Then, watch out for any errors that show up when you start the render with one of mrLiquid's shaders.

You may be missing the auxiliary library mrLibrary in a library path, perhaps.

For Mayatomr to locate all the mrLiquid auxiliary libraries, the environment variable **LD_LIBRARY_PATH** must have been set up correctly for the shell you started maya from for Unix systems. This is usually done by sourcing the **environment.sh** file that the mrLiquid installer installs by default.

Under Windows, the **PATH** variable is used instead.

Q: mental ray stand-alone cannot load mrLiquid's shaders

A: This is often a problem with your environment settings. mental ray stand-alone uses the environment variables **MI_RAY_INCPATH** to locate shader .mi files and **MI_LIBRARY_PATH** to locate the shaders themselves (.dll or .so files).

Whe mrLiquid calls the stand-alone these variables are set automatically from the value of **MI_CUSTOM_SHADER_PATH**. For debugging and information, the values of these variables is also printed out the first time you start a render or an IPR.

However, if you started the stand-alone yourself, you will need to set up your environment correctly.

These variables must have a path to mrLiquid's shaders for the correct version of mental ray you are running.

Under Unix, these variables are usually set in the **environment.sh** file. Under Windows, a similar **environment.bat** file is provided.

For stand-alone to load all the mrLiquid auxiliary libraries, the environment variable **LD_LIBRARY_PATH** must also have been set up correctly for the shell you started maya from

for Unix systems. This is usually done by sourcing the **environment.sh** file that the mrLiquid installer installs by default.

Q: Under Linux, firing up the IPR or the render hangs Maya (either completely or briefly for some seconds).

A: This problem arises when forking a command under Maya on Linux and the command is not found on disk.

With mrLiquid, this often means that your **MRL_RAY** variable is not set or is set to point to an invalid mental ray executable for the OS.

Q: When rendering, I get warnings like:

```
warn   302005: mayabase.mi, line 189: while defining declaration
"maya_blinn": ignoring redeclaration of function maya_blinn
```

This warning most likely means a conflict in your ray stand-alone environment. More than likely a .rayrc or maya.rayrc file is being loaded at the start of the render with absolute paths.

With mrLiquid, the use of .rayrc or maya.rayrc files is no longer recommended. mrLiquid automatically adds **$include "mayabase.mi"** or similar for each shader used in your scene. You only need to have your MI_CUSTOM_SHADER_PATH set correctly. As the $include paths are different between mrLiquid and the .rayrc file, mentalray will incorrectly load the shader shader (or shader definition) twice.

Q: When rendering, I get warnings like:

```
API  0.0  warn   302017: [stdin], line 37: illegal argument
"refractionBlurLimit" for function call to "maya_options"
```

This warning indicates an inconsistency between the .mi file generated by mrLiquid and the shaders or .mi files being used to render the scene.

If the warning is about a maya shader, this most likely indicates a conflict with your environment.

For example, you used mrLiquid to generate a maya2008 compatible .mi file, but are using an

older mental ray standalone compatible with maya8.5 only.

Be sure to set up MI_CUSTOM_SHADER_PATH correctly and use compatible mental ray and shader versions.

Q: Rendering with a mrLiquid shader in mrLiquid works fine but in Mayatomr I get a black frame and a warnings like:

```
PHEN 0.4  error  051011: shader "gg_showinfo" not found
```

This most likely indicates that the shader could not be loaded for some reason. For example, most mrLiquid shaders require the library libmrLibrary to be loaded first. If this library cannot be loaded, the shader will not render and, under some operating systems, you will not get a good error message why loading failed.

For the mrLiquid library to be loaded, it must have been manually loaded in your user's maya.rayrc file or the **PATH** (Windows) or **LD_LIBRARY_PATH** (Unix) variables must be pointing to the location where libmrLibrary is located.

Q: Help! mrLiquid works fine, but my custom shaders crash it. Mayatomr, however, renders fine all of my shaders.

This most likely indicates that your shaders have a subtle bug in that they don't know how to handle light instance groups correctly. If your shader code does something like:

```
mi_query(miQ_INST_ITEM, NULL, *iter, &light);
mi_query(miQ_LIGHT_SHADER, NULL, light, &shader);
```

where *iter is a light instance iterator or pointer, you have a subtle bug here. It is possible in light lists for light instances to point to instance groups of lights, and mrLiquid takes advantage of that for efficiency (while Mayatomr doesn't).

Here's a correct version of the above:

```
mi_query(miQ_INST_ITEM, NULL, *iter, &light);
miScene_types type = mi_db_type( light );
if ( type == miSCENE_LIGHT )
{
   mi_query(miQ_LIGHT_SHADER, NULL, light, &shader);
}
else if ( type == miSCENE_GROUP )
{
   miInteger nkids = 0;
   mi_query(miQ_GROUP_NKIDS, NULL, light, &nkids);

   for(int k = 0; k < nkids; ++k)
   {
      miTag kid;
      mi_query( miQ_GROUP_KID, NULL, light, &kid, k );

      type = mi_db_type(kid);
      if(type == miSCENE_LIGHT)
      {
         mi_query(miQ_LIGHT_SHADER, NULL, kid, &shader);
      }
   }
}
else
{
  mi_error("Invalid light");
}
```

Q: When rendering or using the IPR, I am getting a message similar to:

```
[BUG] Unknown exception thrown in xxxx.
[BUG] Please report, providing scene if possible
[BUG] and brief description of what you did.
```

A: This message indicates a somewhat serious bug in mrLiquid which may effect its operation afterwards. If you see the message, please report the bug to software@filmaura.com, with a description of what you did, and ideally, a scene to reproduce the problem.

It is recommended that if you see this message, you stop the IPR and restart it again.

Q: When starting the IPR or a render, Maya complains with the warning:

```
Could not open IPR viewer.  No IPR image "/usr/tmp/ipr.exr".
```

A: This warning means that the image or image stub that imf_disp or other viewers use to connect to the mental ray stand-alone could not be found. This often means that the render failed for some reason.

This could have caused due to a bug in mrLiquid or due to a crash in the stand-alone renderer.

Open the console window and there may be some additional information available.

If you believe you have stumbled into a mrLiquid bug or problem not listed here, please report it at: software@filmaura.com.

Try to give precise instructions on how to reproduce the bug and, ideally, attach a simple scene file to showcase the problem.

Licensing Information

mrLiquid relies on several public domain libraries for its operation. These are the licenses to them:

OpenEXR

Keyword Index