How to use Maya2Renderer

- Development environment

see ./readme.txt

- Build

see ./readme.txt

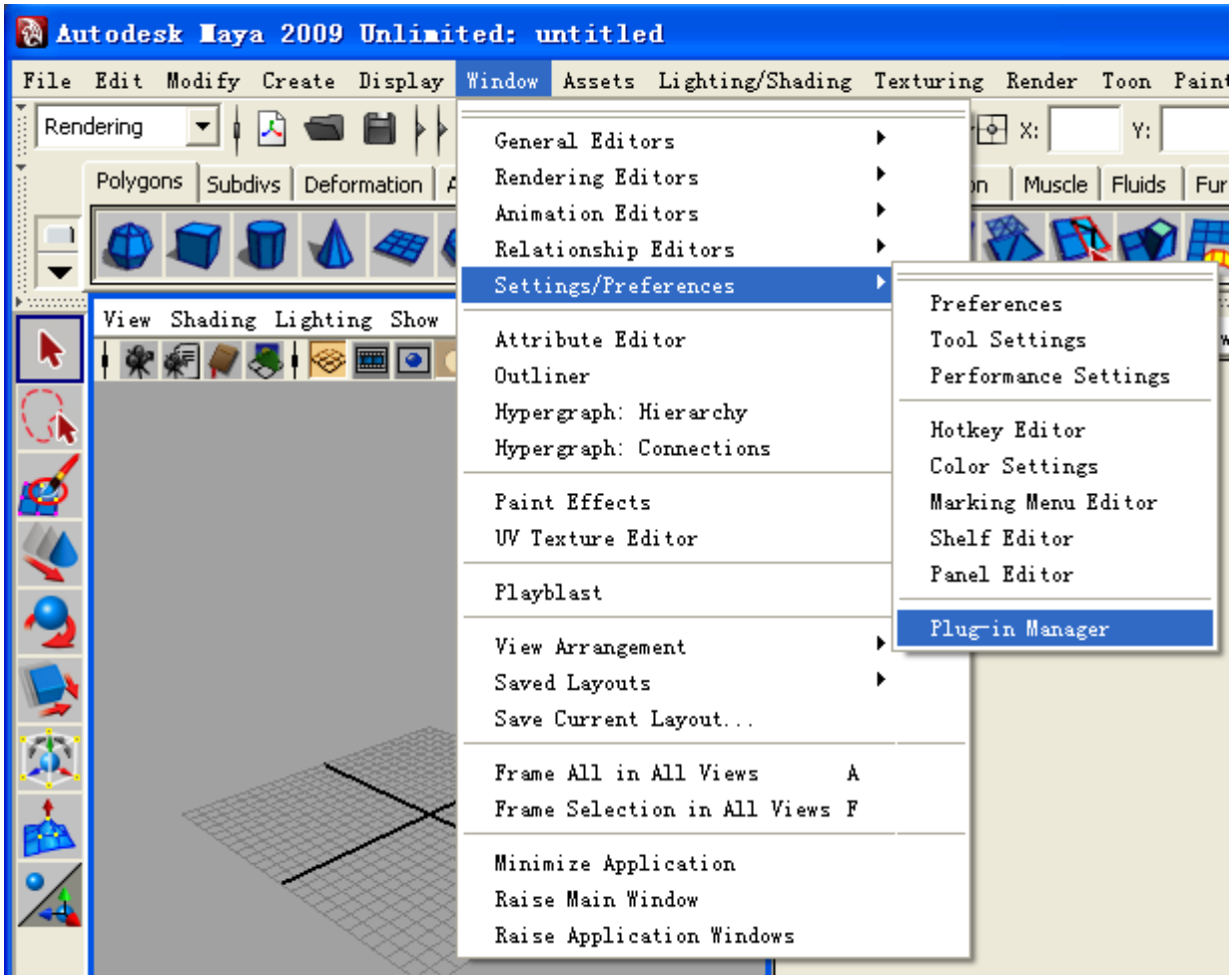
- Install

see ./readme.txt

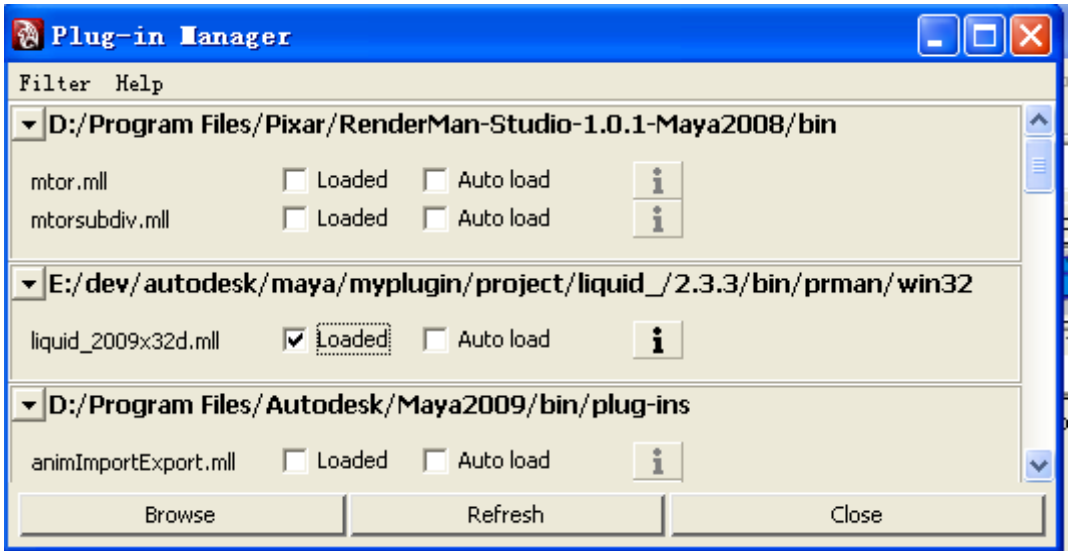
-load liquid plugin

Demo scene is ($liquid_root)/2.3.3/doc/HowToUseMaya2Renderer.ma

1)



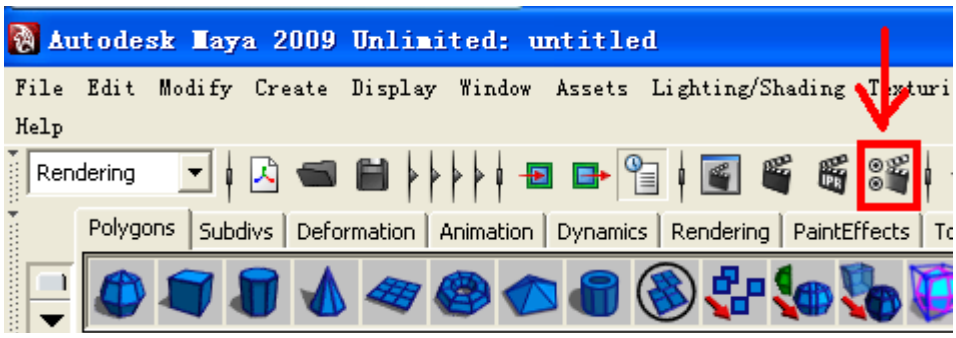
2) select liquid_2009x32d.mll



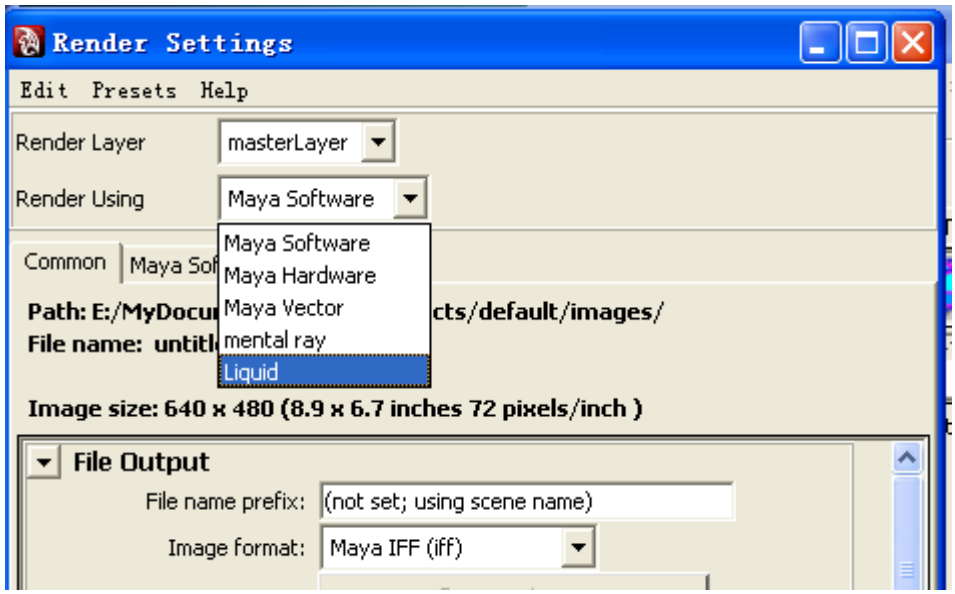
-select liquid renderer

1)

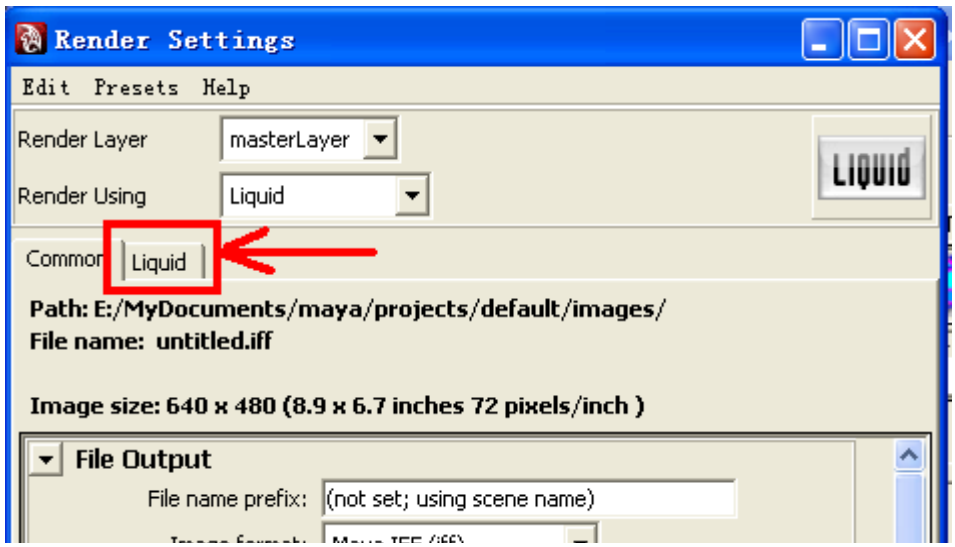
2)

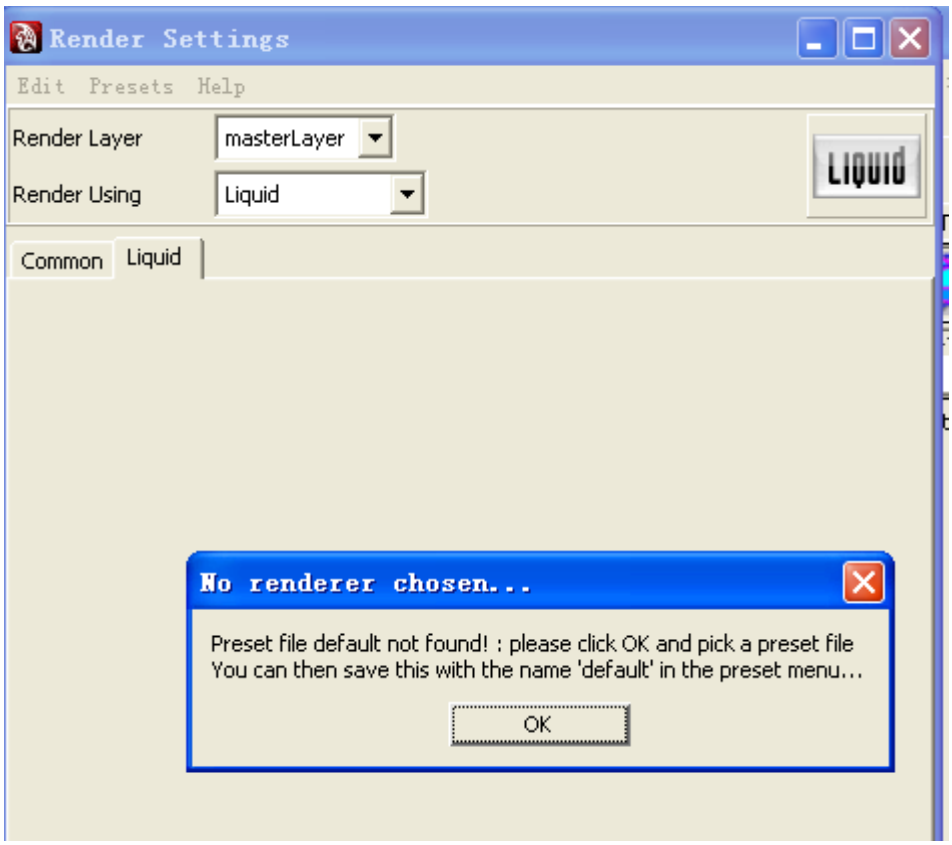

Autodesk Maya 2009 Unlimited: untitled
File Edit Modify Create Display Window Assets Lighting/Shading
Help
Rendering
Polygons Subdivs Deformation Animation Dynamics Rendering PaintEffects

3)

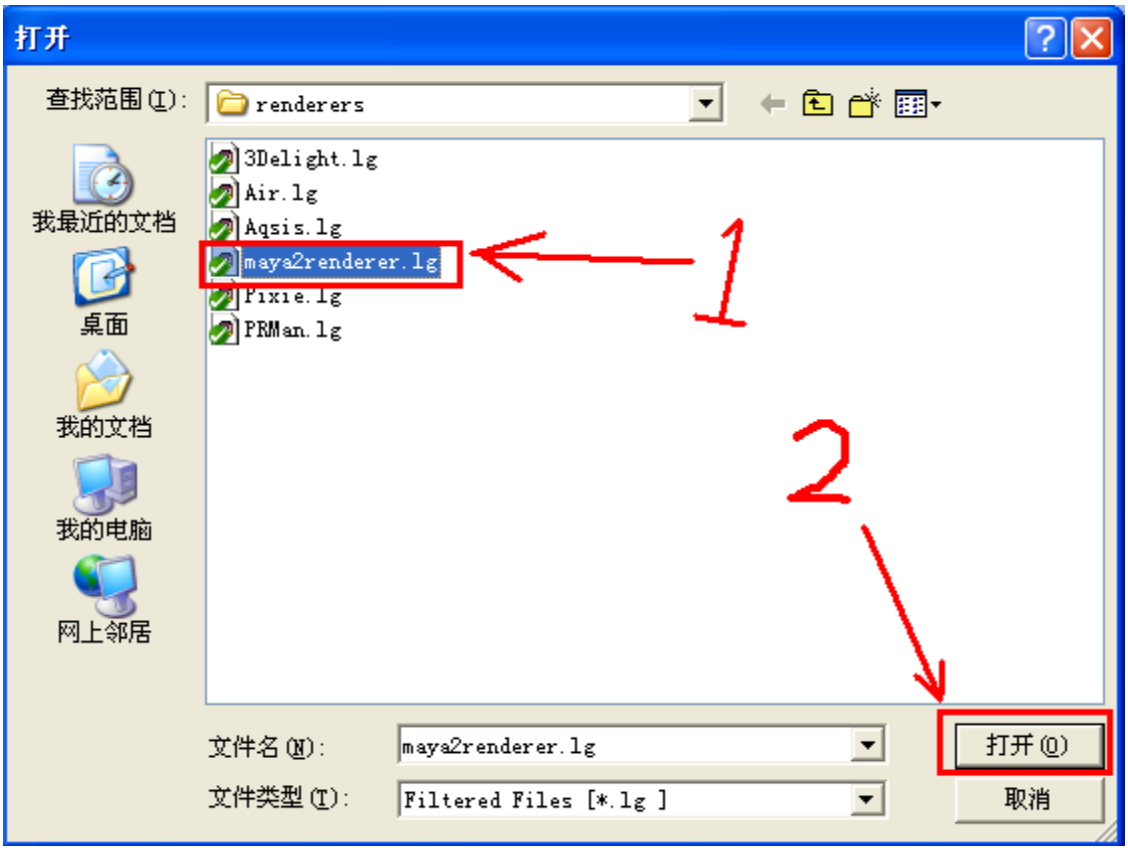

Render Settings
Edit Presets Help
Render Layer masterLayer
Render Using Maya Software
Maya Software
Maya Hardware
Maya Vector
mental ray
Liquid
Common
Image size: 640 x 480 (8.9 x 6.7 inches 72 pixels/inch)
File Output
File name prefix: (not set; using scene name)
Image format: Maya IFF (iff)

4)

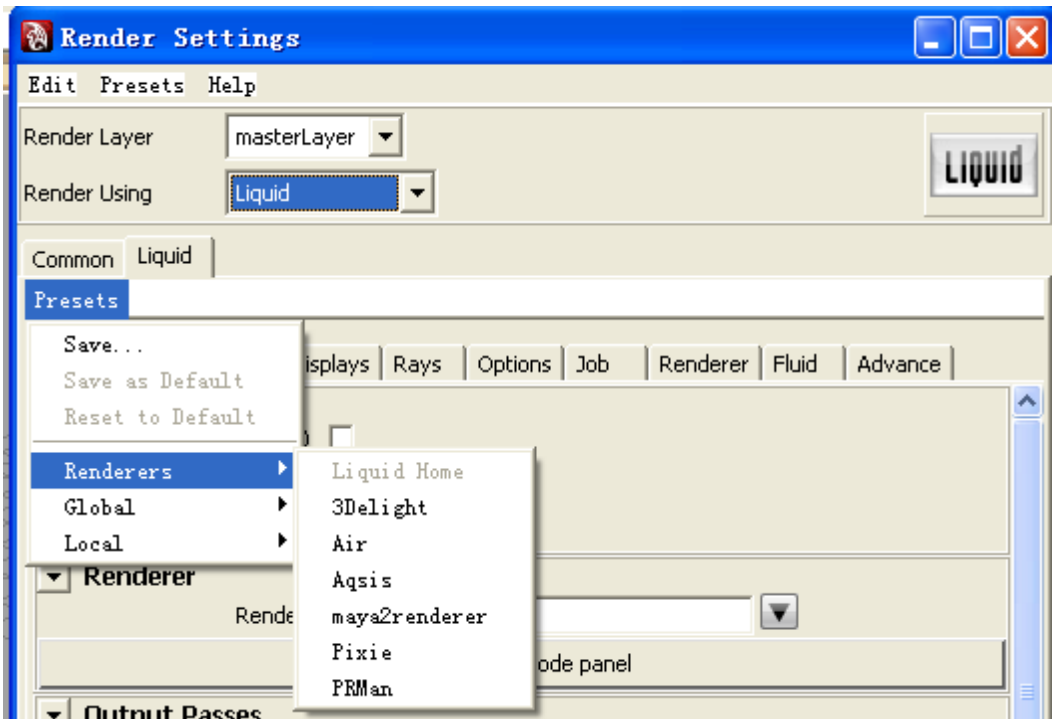

Render Settings
Edit Presets Help
Render Layer masterLayer
Render Using Liquid
LIQUID
Common
Liquid
Path: E:/MyDocuments/maya/projects/default/images/
File name: untitled.iff
Image size: 640 x 480 (8.9 x 6.7 inches 72 pixels/inch)
File Output
File name prefix: (not set; using scene name)



5)



(you can select *.lg in this way:

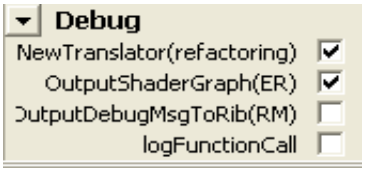


)

-RenderSettings of maya2renderer

Demo scene is ($liquid_root)/2.3.3/doc/HowToUseMaya2Renderer.ma

-Debug



1. NewTranslator(refactoring)
   This is a new feature of maya2renderer.
   I'm refactoring liquidMaya, and put the new export process into liqRibTranslator::_doItNew();
   The original export process is put into liqRibTranslator::_doIt();
   If NewTranslator(refactoring) is checked, liqRibTranslator::_doItNew() is executed,
   otherwise liqRibTranslator::_doIt() is executed.
   If you set NewTranslator(refactoring) to true, you are using maya2renderer,
   If you set NewTranslator(refactoring) to false, you are using liquidMaya,

2. OutputShaderGraph(ER)
   This is a new feature of maya2renderer.
In liquidMaya you MUST export the material of the mesh before you render that mesh.
In maya2Renderer, if you set OutputShaderGraph(ER) to true, you don't have to convert the material manually. The materials will be converted automatically.

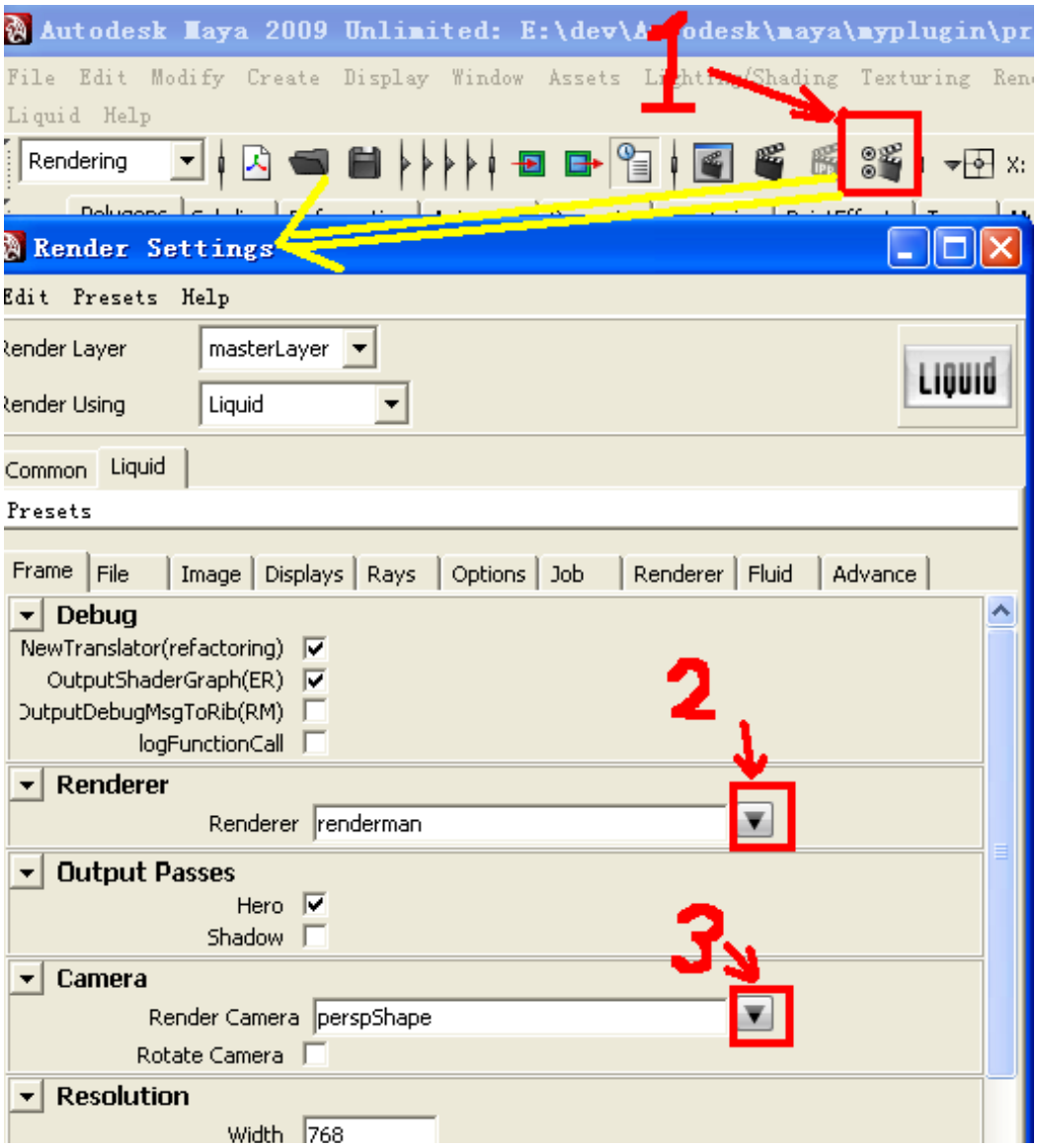
3. OutputDebugMsgToRib(RM)

   This is a new feature of maya2renderer. _liqRIBMsg(const char* msg) will output the msg to rib file.

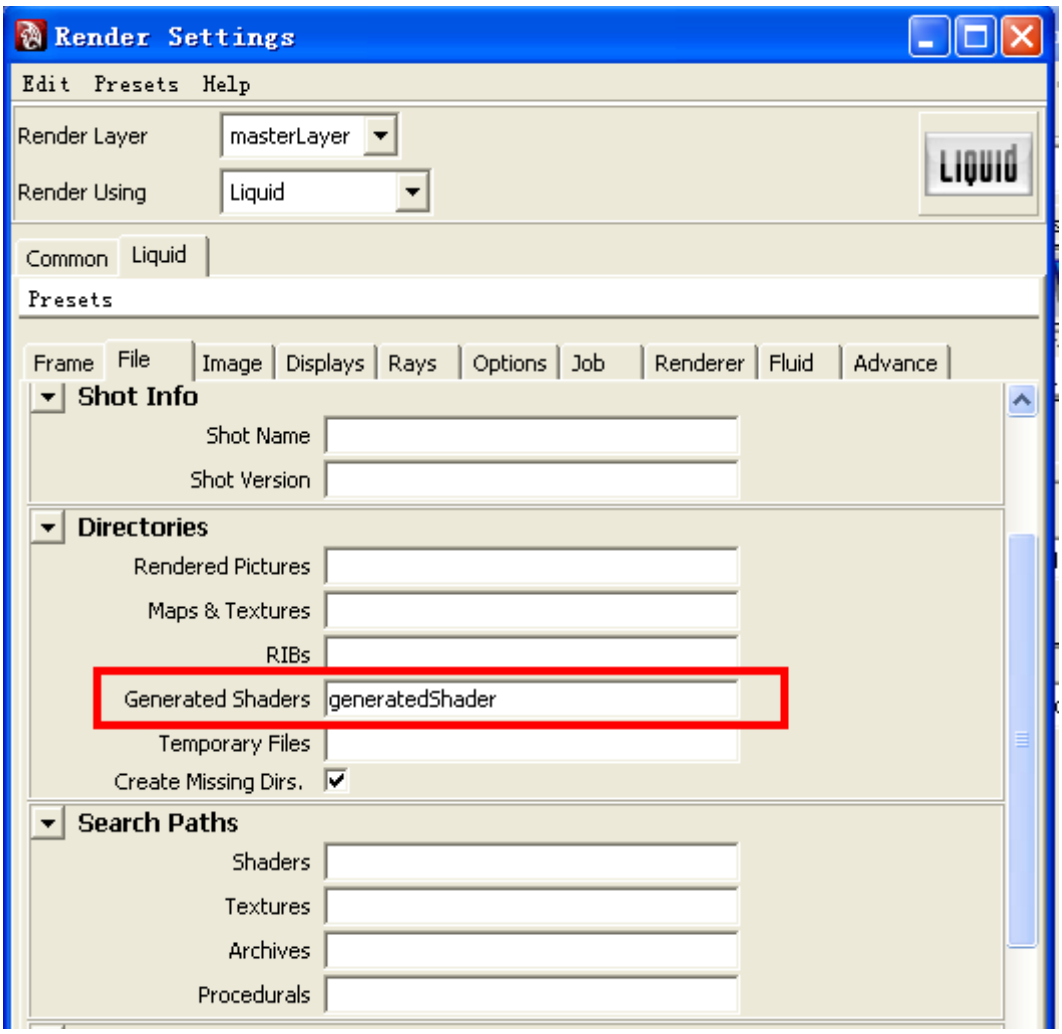
4. logFunctionCall

This is a new feature of maya2renderer. _logFunctionCall(const char* msg) will output the msg. It is useful to trace the call stack in batchrender mode.

7) select "Renderer" and "Camera" in "Render Settings"
"Renderer" MUST be setted to "renderman" in liquidMaya,(in maya2renderer, you can set the "Renderer" to another renderer, e.g. elvishray)

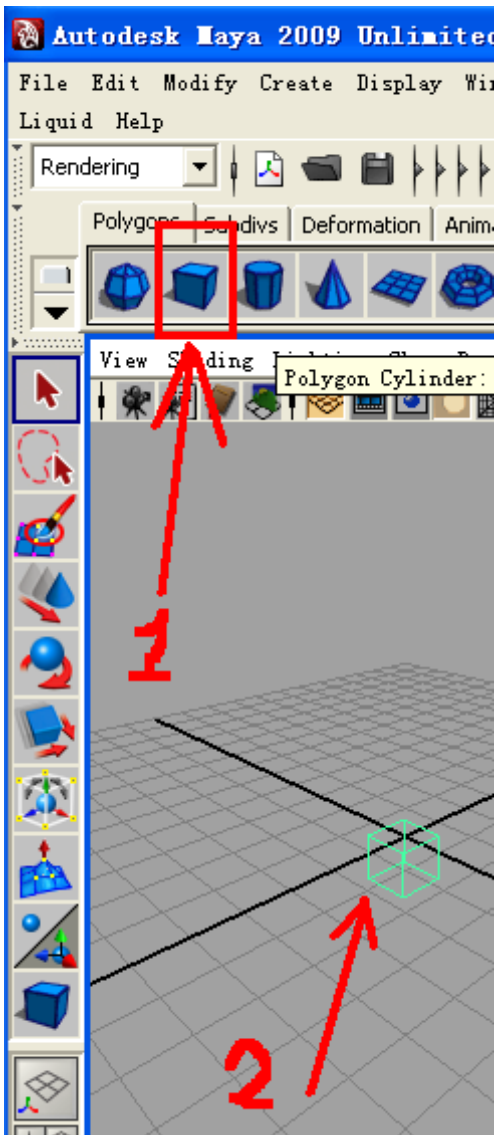


8) set GeneratedShaders directory

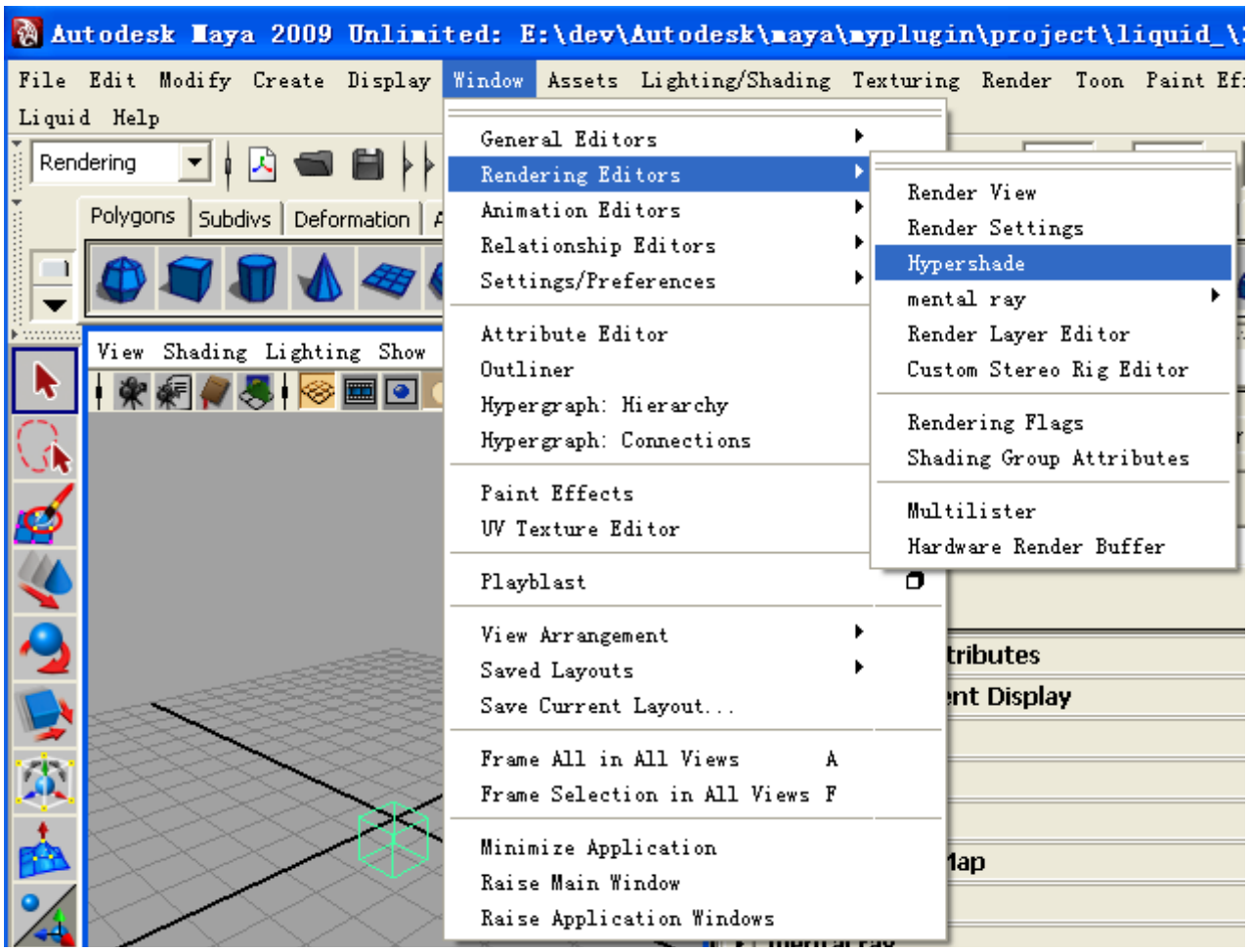


-Create the scene

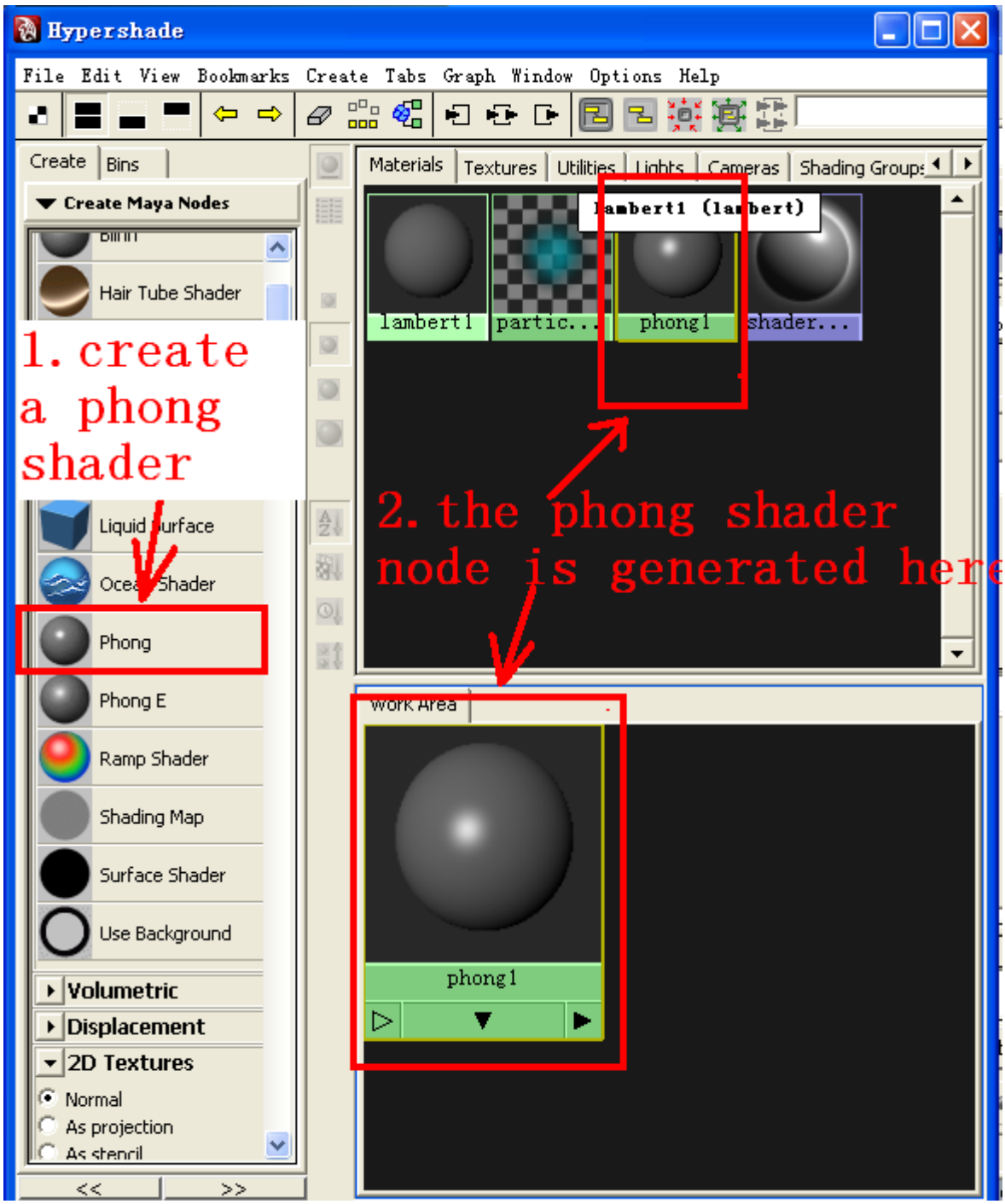
1.create a box



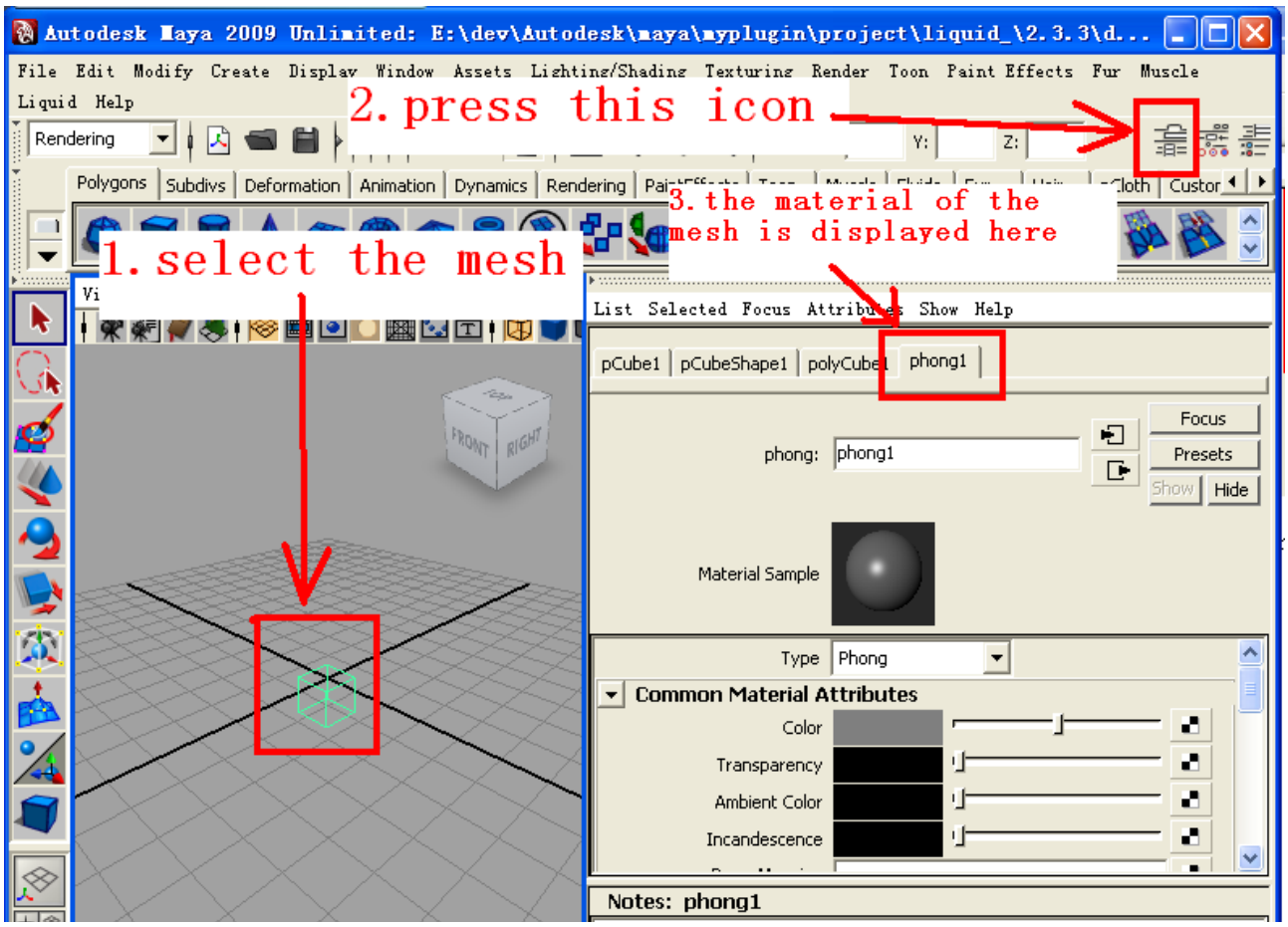
3. assigned a material to this box



4. let's create a phong shader



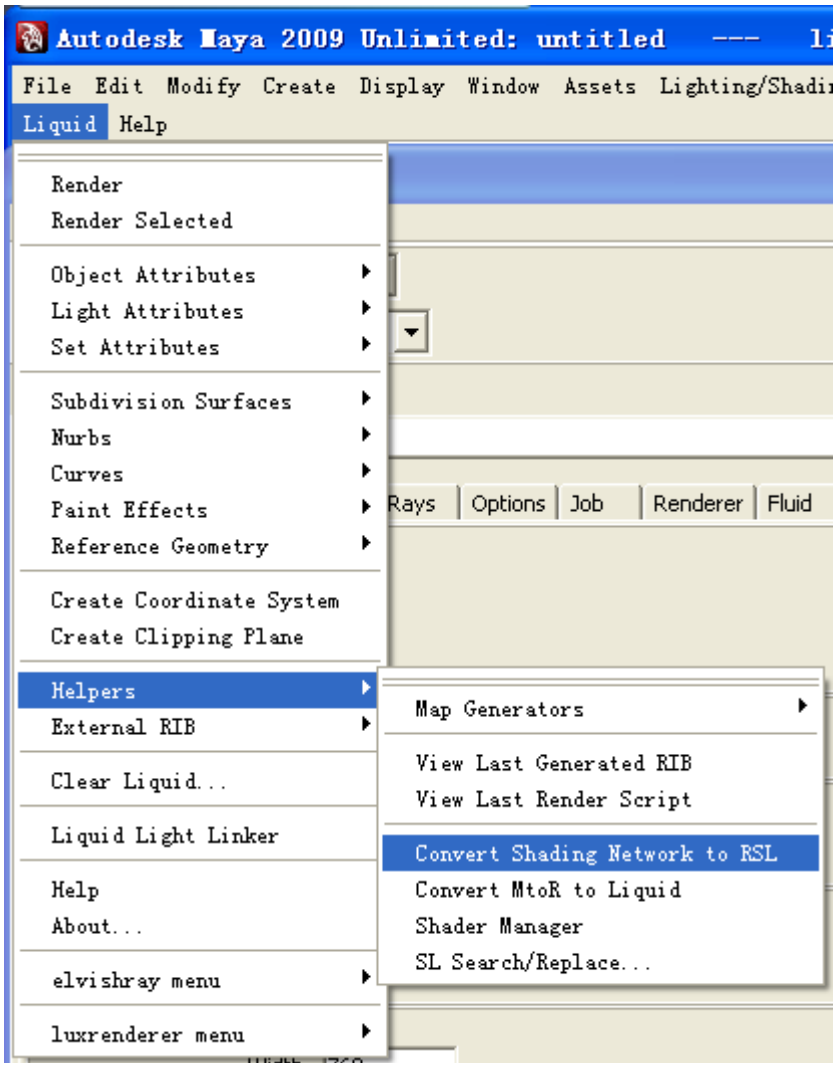
5.select the mesh, then press(not click) the right mouse button on the phong1 node, and select "Assign Material To Selectedion."
6.you can see the phong1 is assigned to the mesh.



-export the material of the mesh

In liquidMaya you MUST export the material of the mesh before you render that mesh.This is the steps:

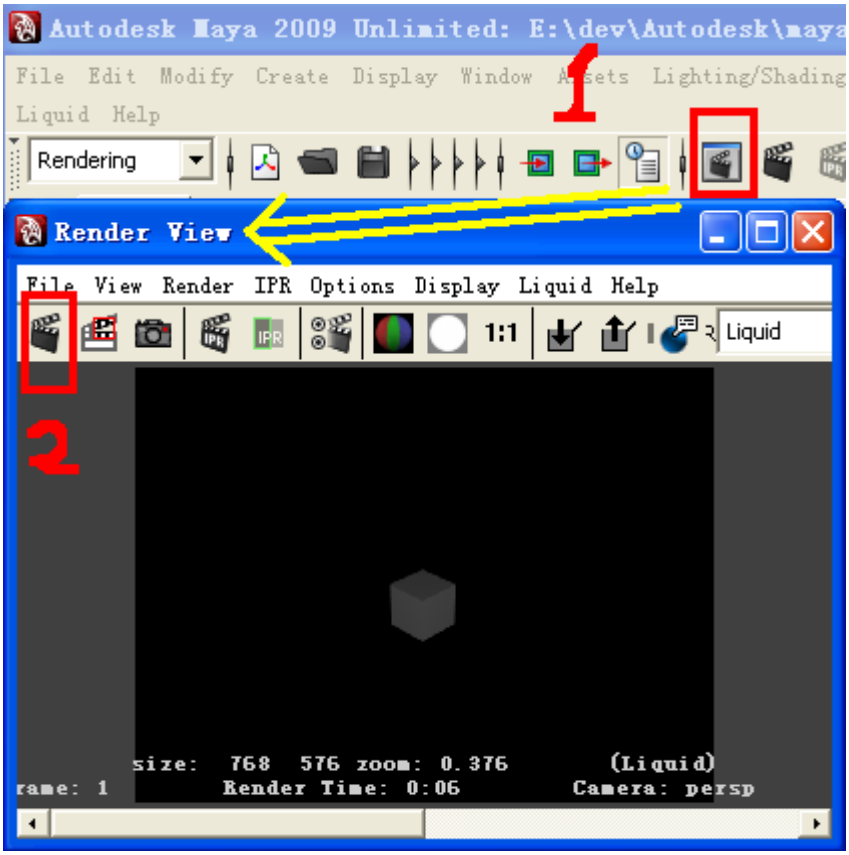
- make sure "GeneratedShaders" directory is set.
- select the mesh which you want to render
- Menu --> liquid -->Helpers --> Convert shading network to RSL:



phong1.sl and phong1.slo will be generated at E:\MyDocuments\maya\projects\default\generatedShader\

-Render

Press the "RenderView" button, and press "Render" button:



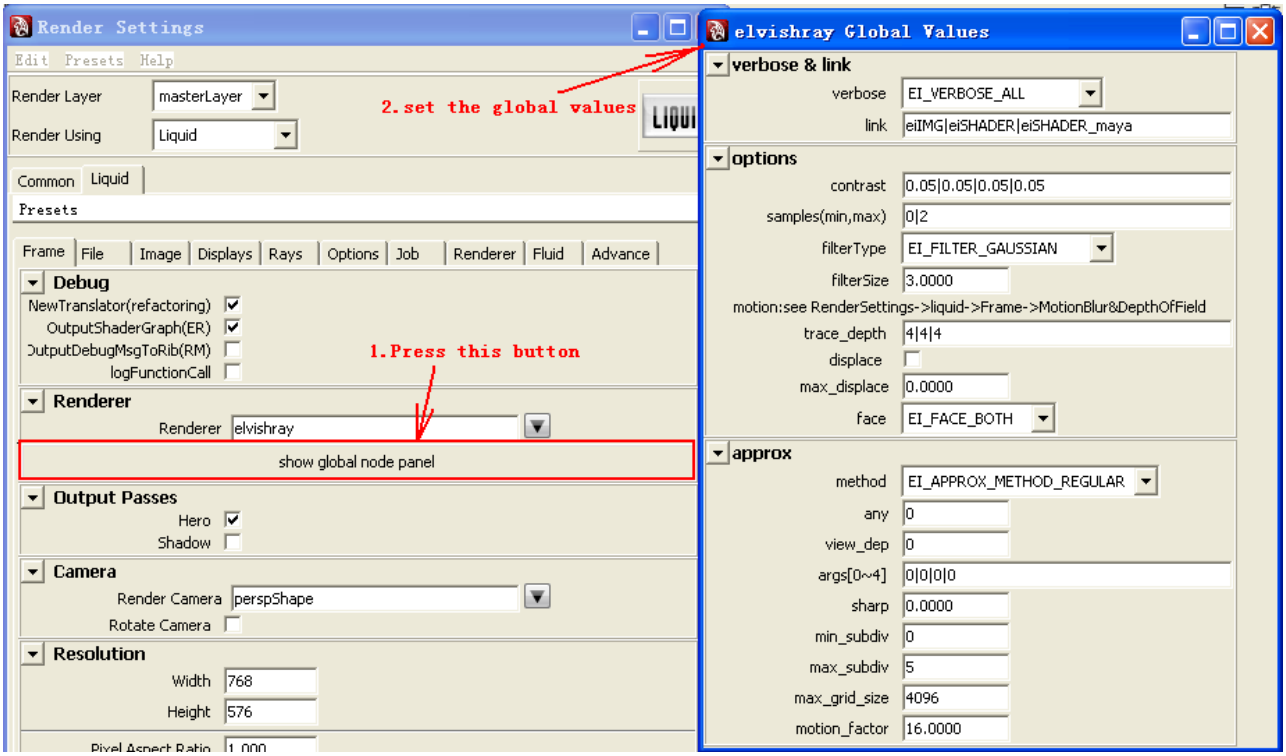
The rib file will be generated at E:\MyDocuments\maya\projects\default\rib,
(e.g. E:\MyDocuments\maya\projects\default\rib_perspShape.0001.rib)

-Renderer

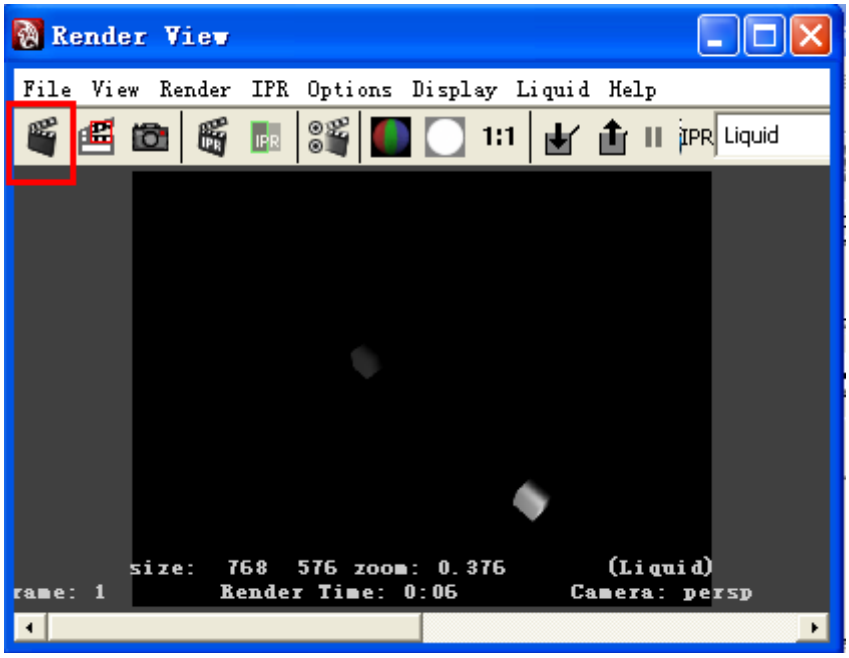
You can select other renderers in maya2renderer, e.g. elvisyray.

-set global values

Press the "show global node panel" button, it will show the global values for this renderer.



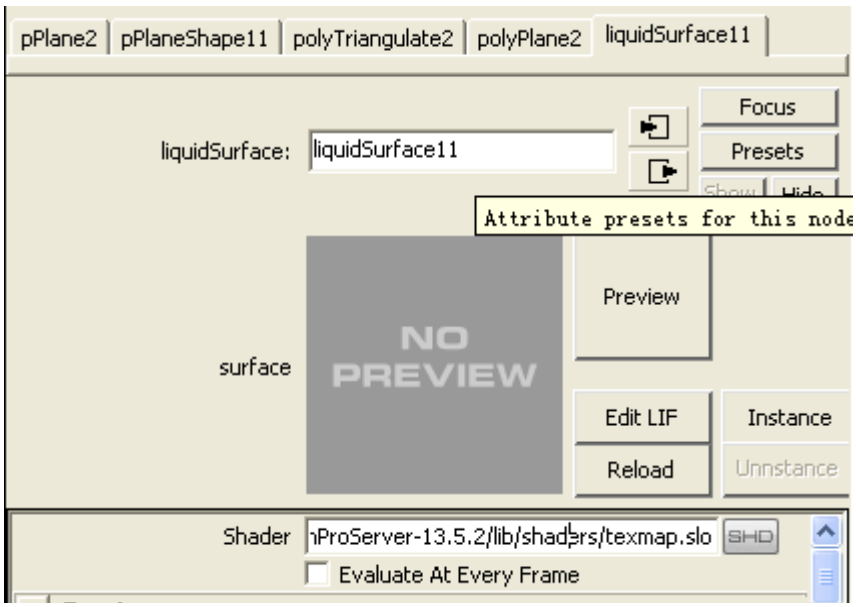
Render



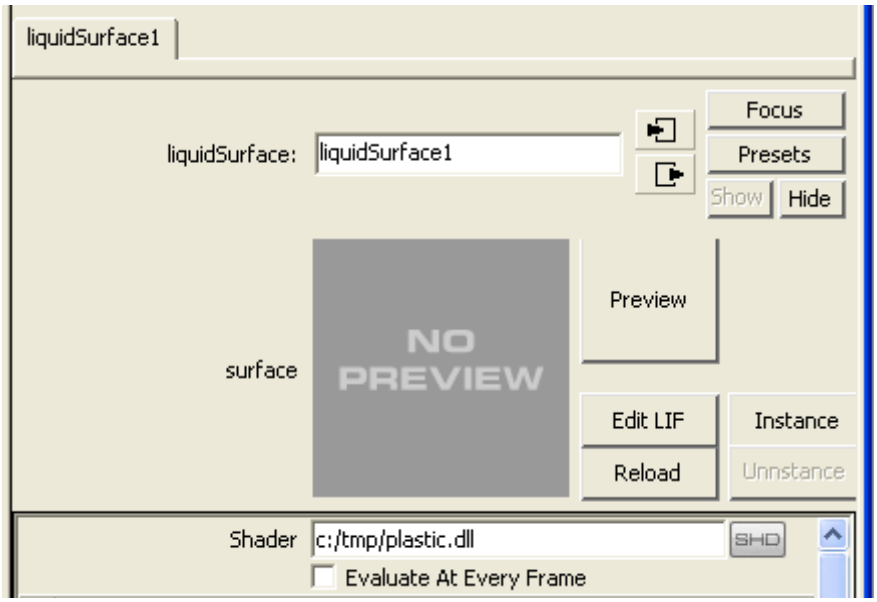
It will generate file:///<MayaProjectDirectory>\rib\<SceneName>.er which trace the invoking of ERAPI

- liquidSurfaceShader

– In liquidMaya, liquidSurfaceShader is used for user-defined surface shader(it is *.slo for prman)



– In maya2renderer, liquidSurfaceShader is extended to supported user-defined shader for other



renderers.
If you use prman, you set the Shader to a slo file, and don't have to do anyting else.

If you use elvishray, e.g. In liquidSurface1, if you set Shader to c:/tmp/plastic.dll, you **have to create** c:/tmp/plastic.pl. Because maya2renderer will search c:/tmp/plastic.pl, and read this pl file to generate shader parameters automatically.

- PL file

pl file contains shader type, shader parameter list.
pl file must locates at the same directory as the shader file. For elvishray, the shader file is a dll file.

pl file is used to generate the UI of parameters of user-defined shader automatically in Maya. The following image show this process in details

%liquidRoot%\2.3.3\doc\HowToGenerateShaderParameterUI\HowToGenerateShaderParameterUI.png

or
https://github.com/maya2renderer/maya2renderer/blob/master/2.3.3/doc/HowToGenerateShaderParameterUI/HowToGenerateShaderParameterUI.png

Here is an example of pl file:

```
#shaderType // shaderType: surface, volume, displacement, light. shaderType must be set before shadername
surface
#shaderName //shadername must be set after shaderType
liquidchecker
//shader parameter list. One line for each parameter,
#Name:Details:Type:IsOutput:Defaults:ArraySizes//Attributes of each parameter, and the attributes are seperated by colon.
frequency:uniform:float:0:8.5:-1
mode:uniform:float:0:0.0:-1
mode2:uniform:float:0:0.5:-1
```

- Refactoring

- Global variables

Put global variables to struct liqGlobalVariable(in liqGlobalVariable.h)
define struct liqGlobalVariable liqglo(in liqGlobalVariable.cpp).

maya2renderer provides specified gobal panel for each renderer, see previous section set global values.

class liqRibTranslator has many interfaces which has postfix '__', these interfaces are refactored from original interfaces. I put all these interfaces into liqRibTranslatorNew.cpp.

- main atchitecture

See ./main_architecture.jpg (https://github.com/maya2renderer/maya2renderer/blob/master/2.3.3/doc/main_architecture.jpg). I list only a few classes in maya2renderer in this diagram. The main translation process is shown in ./main_translation_process.jpg(https://github.com/maya2renderer/maya2renderer/blob/master/2.3.3/doc/main_translation_process.jpg).

Class RendererInterface
This class provides basic interfaces for your renderer. Mjost of these interfaces are fat interface, and will be refactored in the future.
If you want to add your new renderer, e.g. appleseed, into maya2renderer, you should define namespace appleseed, and let class appleseed::Renderer inherite from liquid::RendererInterface

```
namespace appleseed{
    class  Renderer : public liquid::RendererInterface{
```

```
        };
    }
```

ShaderOutputVisitor
I use the visitor pattern to translate the internal maya shader nodes for the new renderer. For example, ShaderOutputVisitor provides the following abstract interfaces: visitBlinn(), visitLambert(), visitPhong() and etc.

RSL::Visitor inherits from ShaderOutputVisitor, RSL::Visitor visits each shader node and output the renderman rsl shader to a sl file. I rename the extention of this sl file to sl_my. sl_my file can be compile by renderman shader compiler. RSL::Visitor use RSL::OutputHelper to add parameters of the shader.

ERCall::Visitor is responsible for constructing shader for elvishray from maya shader node. ER::Visitor uses **ERCall::OutputHelper** to add shader parameters.

ER:Visitor and **ER::OutputHelper** are designed to output the shader into a log file, so I can check whether there is something wrong.

See ./ExportShaderGraph.jpg(https://github.com/maya2renderer/maya2renderer/blob/master/2.3.3/doc/ExportShaderGraph.jpg) for more details.

Class tHeroRibWriterMgr is refactored from the liquidMaya::liqRibTranslator. tHeroRibWriterMgr::write() contains the translation process of the hero pass. See ExportHeroPass.jpg (https://github.com/maya2renderer/maya2renderer/blob/master/2.3.3/doc/ExportHeroPass.jpg)for more details.

Class tShadowRibWriterMgr is refactored from the liquidMaya::liqRibTranslator. tShadowRibWriterMgr::write() contains the translation process of the shadow pass. See ExportShadowPass.jpg(https://github.com/maya2renderer/maya2renderer/blob/master/2.3.3/doc/ExportShadowPass.jpg) for more details.

- NOTE

Maya2renderer doesn't provide API for your renderer, because the general framework is not stable enough now. Many interfaces need to be refactored.

shader graph must be exported before exporting mesh.

- export shader graph

In liquidMaya, you have to convert the shader graph manully before you render the scene.

In maya2renderer, you don't have to convert the shader graph to manully. The shader graph will be exported in each frame.

In liquidMaya, shader graph is exported manually once. In maya2renderer, the shader graph is exported automatically in each frame.

In order to translate maya shader graph for other renderers, I rewrite the following mel files with c++:
liquidConvertShadingNetworkToRSL.mel,
liquidWriteRSL_materials.mel,
liquidWriteRSL_textures.mel,
liquidWriteRSL_utilities.mel
These mel files are obsoleted, and class **ConvertShadingNetwork** handles the translation of the shader graph. The public interfaces of **ConvertShadingNetwork** are ported from mel.

When we traverse the shader graph, whether a plug of a shader node can be output is determinded by **validConnection** which is defined as a map:

```
std::map<ShaderName, ValidPlugins> validConnectionMap;
```

For example, in lambert shader , we need to export color, transparency, ambientColor, incandescence, diffuse, outColor and outTransparency. So we add these plugin names to validConnectionMap, like this:

```
MStringArray validConnection;

//lambert
validConnection.clear();
validConnection.append("color");
validConnection.append("transparency");
validConnection.append("ambientColor");
validConnection.append("incandescence");
validConnection.append("diffuse");
validConnection.append("outColor");
validConnection.append("outTransparency");
validConnectionMap.insert(std::make_pair("lambert", validConnection));
```

This code segment is placed in ShaderValidConnection::setValidConnection();

The following images show more details:
./ExportShaderGraph.jpg
(https://github.com/maya2renderer/maya2renderer/blob/master/2.3.3/doc/ExportShaderGraph.jpg)
./How_to_output_a_ER_shader.jpg
(https://github.com/maya2renderer/maya2renderer/blob/master/2.3.3/doc/How_to_output_a_ER_shader.jpg)
./How_to_output_a_RSL_shader.jpg
(https://github.com/maya2renderer/maya2renderer/blob/master/2.3.3/doc/How_to_output_a_RSL_shader.jpg)

./convertShadingNetworkToRSL_er.jpg (https://github.com/maya2renderer/maya2renderer/blob/master/2.3.3/doc/convertShadingNetworkToRSL_er.jpg)
./convertShadingNetworkToRSL_rsl.jpg (https://github.com/maya2renderer/maya2renderer/blob/master/2.3.3/doc/convertShadingNetworkToRSL_rsl.jpg)

- export user-defined shader

liqShader::tokenPointerArray contains the user-defined shader parameters.

The following images show more details:
./user-defined_shader_er.jpg (https://github.com/maya2renderer/maya2renderer/blob/master/2.3.3/doc/user-defined_shader_er.jpg),
./user-defined_shader_rm.jpg (https://github.com/maya2renderer/maya2renderer/blob/master/2.3.3/doc/user-defined_shader_rm.jpg)

- limit on the string parameter of the user-defined shader
Assuming that we define a string variable, strA, in the user-defined shader.
- if strA is used to specify a texture name(e.g. string strA="/texturepath/a.tex"),
then prefix 'texname' must be added to the variable name(e.g. string texnamestrA="/texturepath/a.tex").
- if strA is **NOT** used to specify a texture name,
the prefix 'texname' must **NOT** be added to the variable name.
In this case, if prefix 'texname' was added to strA, e.g.string texnamestrA="teststr", maya2renderer will output 'string texnamestrA="teststr.tex"' to rib file, but the texture teststr.tex doesn't exist, and it would lead an error.
see liqShader.cpp line306 for more details.

```
// [2/1/2012 yaoyansi]
// This is a limit in liquid(maya2renderer).
// - If the attribute name starts with 'texname',(e.g. texname, texname0, texname_0, and etc.)
//   it is a texture name or texture full path,
//   so we MUST append '.tex' to the plug value.
// - If the plug is not a texture name or texture full path,
//   DO NOT let the attribute name starts with 'texname'.
```

- the string value cann't contain '|' which is assigned to global variable $gSeperator.

- export mesh

todo

- export light

todo

- log system

Mel log

trace log

Shadow pass is refatored from liquidMaya, and is not extended for elvishray.

Each test case contains *.rib ,*.er, *.log, they are logs for that test case.

- TodoList

Some interfaces in class Renderer are fat interface, they will be refactored.

%liquidroot%/todolist.txt

- How to extend maya2renderer to support a new renderer

Maya2renderer provide a *newrenderer*(nr) module as a template to support a new renderer.

Now let's extend maya2renderer to support appleseed(as) renderer.

1. Copy %LiquidRoot%\2.3.3\include\liqConfig_template.h to %LiquidRoot%\2.3.3\include\liqConfig.h
and add macro for your new renderer, e.g. _USE_APPLESEED_

2. copy %LiquidRoot%\2.3.3\src\newrenderer*.* to %LiquidRoot%\2.3.3\src\appleseed*.*, replace the prefix 'nr' int the file name with 'as'. e.g. rename nr_renderer.h to as_renderer.h, and nr_CreateMenus.mel to as_CreateMenus.mel.

3. Rename liqGlobalsNodenewrenderer.mel to liqGlobalsNodeappleseed.mel;
In liqGlobalsNodeappleseed.mel, replace 'newrenderer' with 'appleseed'

4. in as_AEliquidInstanceFlags.mel,as_CreateMenus.mel and userSetup.mel, replace 'newrenderer' with 'appleseed', and replace 'nr' with 'as'.

5. In *.h and *.cpp,
replace 'namespace newrenderer' with 'namespace appleseed',
replace 'nr_' with 'as_',
replace _USE_NEWRENDERER_ with _USE_APPLESEED_

6. In as_renderer.cpp, in Renderer::ribPrologue_begin(...), replace '.nr' with '.as'.

7. replace your maya.env with %LiquidRoot%\2.3.3\bin\maya.env,

or add LIQUID_AS_MEL = $LIQUID_ROOT/2.3.3/src/appleseed/mel to your maya.env, and append $LIQUID_AS_MEL to MAYA_SCRIPT_PATH.

8. In %LiquidRoot%/2.3.3/mel/registerLiquidRenderer.mel, add "appleseed" to $renderer, like this:

```
global  string $renderer[] = {"renderman","elvishray","luxrenderer","appleseed"};
```

9. In %LiquidRoot%/2.3.3/mel/registerLiquidRenderer.mel, append the following code in function liqGlobalsNodeRendererMain(string $cmd):

```
 else if($rnd=="appleseed"){
 liqGlobalsNode_appleseed($cmd);
 }
```

10. In %LiquidRoot%\2.3.3\src\renderermgr.cpp,

```
void RendererMgr::createFactory(const std::string& renderername)
        {
                CM_TRACE_FUNC("RendererMgr::createFactory("<<renderername<<")");

                ...
                else if(renderername=="appleseed"){
                        m_factory = new appleseed::Factory();
                }
                ...
        }
```

11. automation test