RenderMan For Artists #05

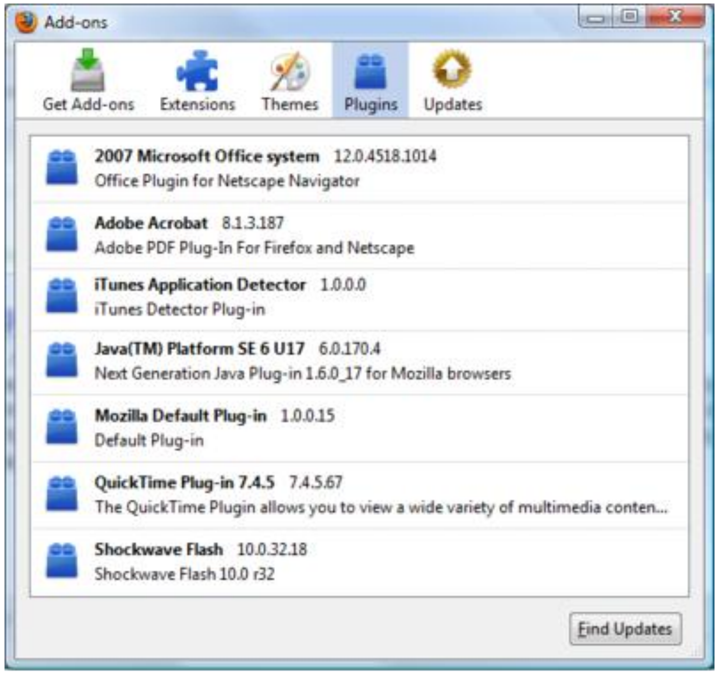
Ri Filter Plug-in

Wanho Choi
(wanochoi.com)

Plug-in?

- A small add-on software component
  - It adds specific capabilities to a larger software application.
  - It enables customizing the functionality of an application

- Examples
  - Adobe Flash Player in web browsers
  - QuickTime Player in web browsers
  - FumeFX in 3DS Max
  - Qualoth in Maya
  - etc.



2007 Microsoft Office system 12.0.4518.1014
Office Plugin for Netscape Navigator

Adobe Acrobat 8.1.3.187
Adobe PDF Plug-In For Firefox and Netscape

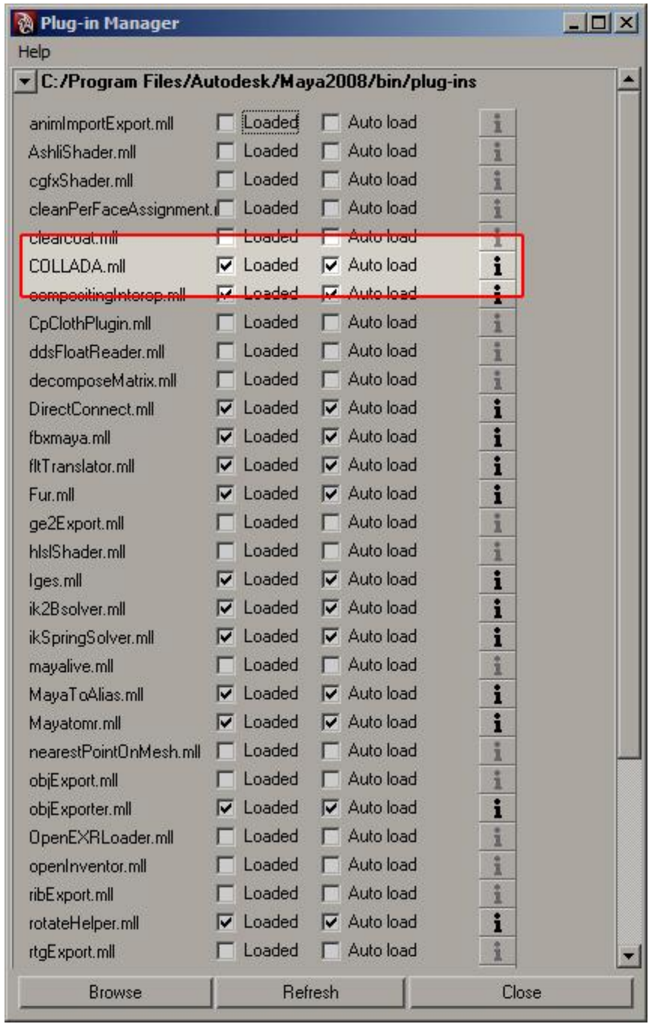
iTunes Application Detector 1.0.0.0
iTunes Detector Plug-in

Java(TM) Platform SE 6 U17 6.0.170.4
Next Generation Java Plug-in 1.6.0_17 for Mozilla browsers

Mozilla Default Plug-in 1.0.0.15
Default Plug-in

QuickTime Plug-in 7.4.5 7.4.5.67
The QuickTime Plugin allows you to view a wide variety of multimedia conten...

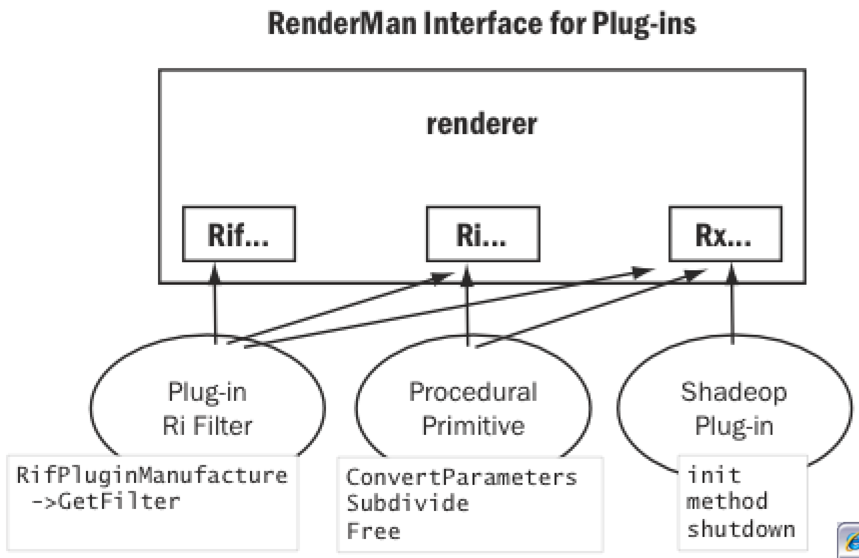
Shockwave Flash 10.0.32.18
Shockwave Flash 10.0 r32

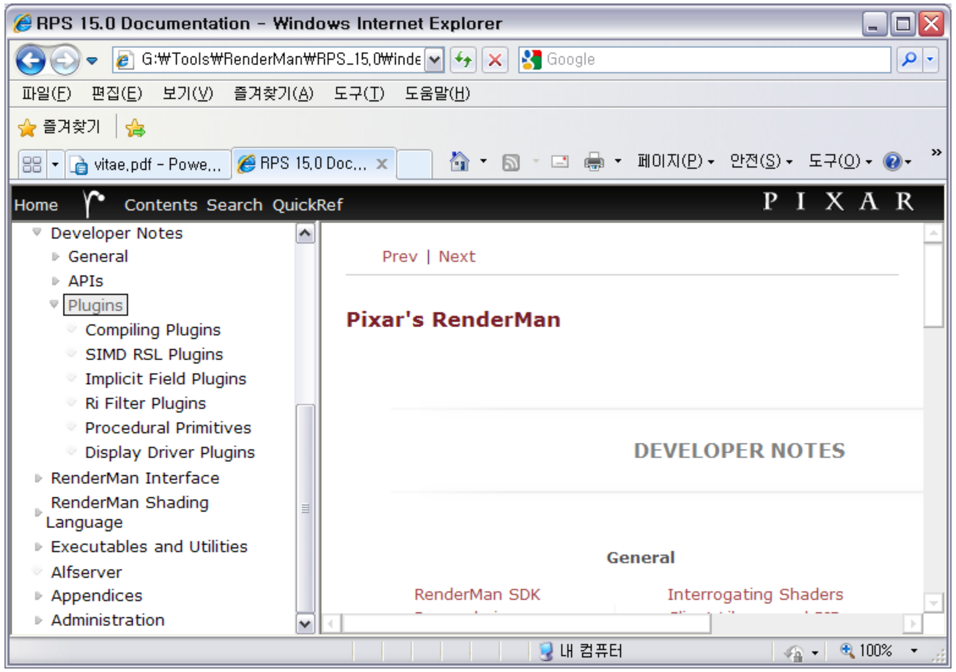


C:/Program Files/Autodesk/Maya2008/bin/plug-ins

animImportExport.mll	☐ Loaded	☐ Auto load
AshliShader.mll	☐ Loaded	☐ Auto load
cgfxShader.mll	☐ Loaded	☐ Auto load
cleanPerFaceAssignment.m	☐ Loaded	☐ Auto load
~~clearcoat.mll~~	~~☐ Loaded~~	~~☐ Auto load~~
COLLADA.mll	☑ Loaded	☑ Auto load
~~compositingInterop.mll~~	~~☑ Loaded~~	~~☑ Auto load~~
CpClothPlugin.mll	☐ Loaded	☐ Auto load
ddsFloatReader.mll	☐ Loaded	☐ Auto load
decomposeMatrix.mll	☐ Loaded	☐ Auto load
DirectConnect.mll	☑ Loaded	☑ Auto load
fbxmaya.mll	☑ Loaded	☑ Auto load
fltTranslator.mll	☑ Loaded	☑ Auto load
Fur.mll	☑ Loaded	☑ Auto load
ge2Export.mll	☐ Loaded	☐ Auto load
hlslShader.mll	☐ Loaded	☐ Auto load
Iges.mll	☑ Loaded	☑ Auto load
ik2Bsolver.mll	☑ Loaded	☑ Auto load
ikSpringSolver.mll	☑ Loaded	☑ Auto load
mayalive.mll	☐ Loaded	☐ Auto load
MayaToAlias.mll	☑ Loaded	☑ Auto load
Mayatomr.mll	☑ Loaded	☑ Auto load
nearestPointOnMesh.mll	☐ Loaded	☐ Auto load
objExport.mll	☐ Loaded	☐ Auto load
objExporter.mll	☑ Loaded	☑ Auto load
OpenEXRLoader.mll	☐ Loaded	☐ Auto load
openInventor.mll	☐ Loaded	☐ Auto load
ribExport.mll	☐ Loaded	☐ Auto load
rotateHelper.mll	☑ Loaded	☑ Auto load
rtgExport.mll	☐ Loaded	☐ Auto load

RenderMan Plug-ins

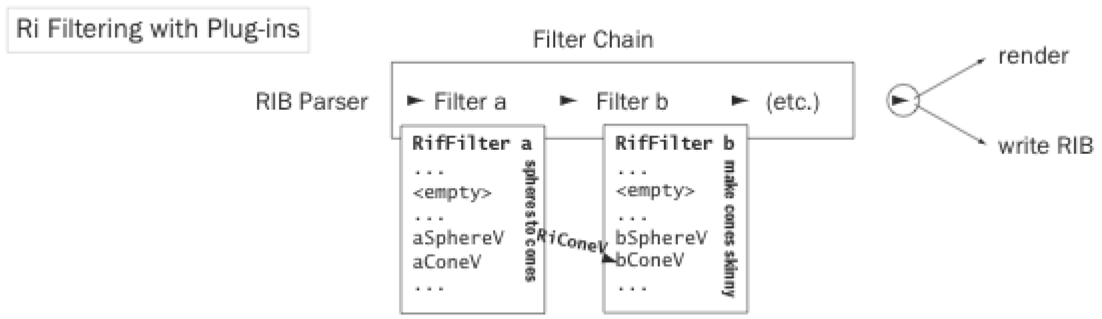

RenderMan Interface for Plug-ins
renderer
Rif...
Ri...
Rx...
Plug-in Ri Filter
Procedural Primitive
Shadeop Plug-in
RifPluginManufacture ->GetFilter
ConvertParameters Subdivide Free
init method shutdown



Ri Filter Plug-in

Parsing a RIB streaming → Searching for patterns that match some criteria → Replacing these patterns with a more preferable, possibly empty, pattern → Rendering

- It is often convenient to construct chains of RIB filters and the result is a powerful back-end RIB processing system that can solve production pipeline problems in a unique way, due largely to its position in the pipeline at the "mouth" of the renderer.



Example – Empty RiFilter Plug-in

```
#include <RifPlugin.h>

class myFilter : public RifPlugin
{
   private:

      RifFilter m_filter;

   public:

      myFilter();
      ~myFilter();

      RifFilter& GetFilter();
};

myFilter::myFilter()
{
   m_filter.Filtering = RifFilter::k_Continue;
}

myFilter::~myFilter()
{
}

RifFilter& myFilter::GetFilter()
{
   return m_filter;
}

RifPlugin* RifPluginManufacture( int argc, char** argv )
{
   return new myFilter();
}
```

- All RiFilter plug-ins must be derived from **RifPlugin**.
- **RifPlugin** is a pure-virtual class.

RifPlugin.h

```
class RifPlugin
{
   public:
      virtual ~RifPlugin() {}
      virtual RifFilter& GetFilter() = 0;
};
```

- All RiFilter plug-ins must have its own dispatch table as a private member variable, which is **RifFilter** structure.

RifFilter.h

```
struct RifFilter
{
   enum DefaultFiltering { k_Continue, k_Terminate };
   …
   RtToken (*Declare)( char* name, char* declaration );
   RtVoid (*WorldBegin)();
   RtVoid (*WorldEnd)();
   …
};
```

- Your subclass can have any number of **RiFilter** objects and can choose which to return based on its own state.
- The **GetFilter()** member returns a pointer to that dispatch table.
- RiFilter plug-ins must implement the **RifPluginManufacture()** procedure. This procedure constructs an instance of a **RifPlugin** subclass.

How to comile & execute

- How to compile
  - g++ -fPIC –I$RMANTREE/include –o myFilter.o –c myFilter.cpp
  - g++ -shared myFilter.o –o myFilter.so

- How to rendering using filter
  - prman –rif myFilter.so –rifargs –rifend test.rib

- How to generate filtered .rib file
  - catrib –o filtered.rib –rif myFilter.so –rifargs –rifend test.rib

RifFilter

- The RifFilter data structure contains a constructor, some versioning information, a filtering mode (DefaultFiltering), and function pointers for every function in the Ri Interface.
- When a given Ri function is parsed, the overridden callback is called instead.
- Then if the DefaultFiltering is set to k_Continue, the next filter in the chain is called. If the mode is set to k_Terminate, then no further calls are made and the parsing terminates.

RifFilter.h

```
struct RifFilter
{
   enum { k_UnknownVersion = 0, k_CurrentVersion = 1 };
   enum DefaultFiltering { k_Continue, k_Terminate };
   short Version; /* the version of the table */
   void* ClientData; /* a place for the plug-in to hang its hat */
   char Reserved[64]; /* for future use */
   DefaultFiltering Filtering; RifFilter ();
   RtToken (*Declare)(char *name, char *declaration);
   RtVoid (*FrameBegin)(RtInt frame);
   RtVoid (*FrameEnd)();
   RtVoid (*WorldBegin)();
   RtVoid (*WorldEnd)();
   ...
};
```

Example – Empty RiFilter Plug-in

```
#include <iostream>
#include <RifPlugin.h>

class myFilter : public RifPlugin
{
   private:

      RifFilter m_filter;
      int m_width, m_height;
      float m_ratio;

   public:

      myFilter( int w, int h, float r );
      ~myFilter();

      RifFilter& GetFilter();

      int width() const { return m_width; }
      int height() const { return m_height; }
      float aspect() const { return m_ratio; }

   private:

      static RtVoid myFormat( RtInt w, RtInt h, RtFloat r );
};
```

```
myFilter::myFilter( int w, int h, float r )
{
   m_filter.Filtering = RifFilter::k_Continue;
   m_filter.Format = myFormat;

   m_width = w;
   m_height = h;
   m_ratio = r;
}

myFilter::~myFilter()
{
}

RifFilter& myFilter::GetFilter()
{
   return m_filter;
}

RtVoid myFilter::myFormat( RtInt w, RtInt h, RtFloat r )
{
   myFilter* obj
   = static_cast<myFilter*>( RifGetCurrentPlugin() );

   RiFormat( obj->width(), obj->height(), obj->ratio() );
}

RifPlugin* RifPluginManufacture( int argc, char** argv )
{
   int w = atoi( argv[0] );
   int h = atoi( argv[1] );
   float r = atof( argv[2] );

   return new myFilter( w, h, r );
}
```

How to comile & execute

- How to compile
  - g++ -fPIC –I$RMANTREE/include –o myFilter.o –c myFilter.cpp
  - g++ -shared myFilter.o –o myFilter.so

- How to rendering using filter
  - prman –rif myFilter.so –rifargs 640 480 1 –rifend test.rib

- How to generate filtered .rib file
  - catrib –o filtered.rib –rif myFilter.so –rifargs 640 480 1 –rifend test.rib

How to apply RiFilters as a sequential chain.

- How to rendering using filter
  - prman –rif myFilter1.so –rifargs –rifend –rif myFilter2.so –rifargs -rifend test.rib

- How to generate filtered .rib file
  - catrib –o filtered.rib –rif myFilter1.so –rifargs –rifend –rif myFilter2.so –rifargs -rifend test.rib

References

- **RenderMan Pro Server 15 *Appplication Notes* & *Developer Notes***