Blender and Renderman
Building a production pipeline for professional use.

Ted Gocek
animate1978@gmail.com
www.blendertorenderman.org
September, 2010

Figure 1: Project Widow's main character with shadow mapped fake SSS and DoF (rendered using 1.6.0 dev build)

Table of Contents

Introduction.................................................................................................................................2
Why Renderman?.......................................................................................................................4
  History...................................................................................................................................4
  RenderMan Interface Specification.........................................................................................5
  Required capabilities.............................................................................................................6

  Optional advanced capabilities ............................................................................................6
Production Requirements.......................................................................................................6
  Feature Set...........................................................................................................................6
  Flexibility............................................................................................................................7
  Robustness..........................................................................................................................7
**Introduction**

“Project Widow” began in May 2009 as a proof of concept short film demonstration, something that could quickly be produced and delivered to the world. Blender and Renderman together could be a very powerful, low cost solution for visual effects and animation and at that time it was decided to really try to prove it. The project quickly grew large, in scale and in complexity, what was supposed to take three months turned into nearly two years and was compounded with some very difficult technical barriers. However much was developed during this time, not only in the assets of the short film but in the knowledge of figuring out solutions to problems, the software we used itself and the skills to get this completed.

The mission was clear and we did not want to be on that list of dead projects that had potential but none the less faded into nothing. Here I am presenting how to build an efficient, film quality animation or visual effects pipeline using open source tools, with focus on Blender and Renderman. It is a production pipeline that equals those of much larger scale in terms on methods, functionality and it is considered “stable”.

The reason for the Blender to Renderman research and development is because a large majority of “Hollywood” visual effects and animation houses rely on Renderman for a good percentage of the shots they create. Renderman has proven to be more than capable of handling the sheer volume of information is has to process per frame, within an acceptable render time and has been able to produce fully generated feature length films. Not to mention Pixar's flagship product, PRMan, has been in existence for over 20 years and with it's use in some of the hugely successful visual effects heavy films early in the 1990's and beyond, has been the key success of CG in film and the catalyst of the current visual effects and animation industry. It has history and production proven experience in the rendering end of productions, Blender on the other hand has only been used for a fraction of that screen time and has never rendered a frame for a big budget effects shot. Blender sadly does not have the credentials yet to be considered as a pipeline tool, or so say some voices in the industry, it has been said that it never would be used for productions because the internal renderer cannot meet industry demands and in a way they are right, despite all it's power and innovation the internal renderer has flaws that do matter when it comes down to pulling off that incredible effects shot that cost a studio millions to produce and hundreds of manpower, multiplied by thousands of render time hours for the average length movie.

This argument can continue forever, there is no side that is truly right. Producers don't care about innovation or open source achievements, they want what works all the time and quickly enough, something that has proven to be reliable, so usually the thought process of “If it works for you it works for me”, unless the tool is used only by a small team in house to achieve a task they are working on, it is considered a huge risk to introduce new tools to a working pipeline. Only a few tools or open source code have been used in such a manner and only recently have more studios come forth with more tools and libraries released to the open source community. In short the mindset that one tool can be used for everything needs to go, there are many tools and programs used to complete a shot and once you get past that, only then can you see how even now we have the same power, tools and skills that the professionals use and have used for some time now. What it takes is some imagination, skill and some really talented programmers if you want to get fancy. There needs to be solid links between pipeline steps and the Blender to Renderman pipeline has been optimized and tested to make sure that each pipeline step can be completed, from modeling to final frame.

Blender has been around for a very long time, only in the past 6 years has it really gotten a lot of attention due to it's release into the open source community, as well as the open movie projects, though in some people's eyes Blender is still considered a “hobbyist” application, which if viewed as such then it is, unless one takes the time to fully explore it, where Blender's power really shows. This is the driving force behind Project Widow; to prove that Blender's powerful modeling and animation capabilities can be meshed together with Renderman's powerful rendering features, using production methods that rival the professionals, all on a non budget, non profit short film. It can be done, our team is doing it, some of us created the means even.
In the early days of the Blender to Renderman effort there was little out there to go on. Blender was only a few years into the open source world, Aqsis and Pixie functioned well but still had issues and with a couple of basic RIB exporters such as the Blenderman script, our work was limited to basic still frames at best. At one point one of the only options to do some really powerful shader work was done after RIB was exported and using the tool Cutter, hand edit the files until the final image was decent. While there was some improvement in the pipeline, this is not a very efficient way to work with Renderman, and not acceptable for professional work by any stretch of the imagination.

Five years have passed since I first came up with the idea to help drive Blender and Renderman development and much has changed, now we believe that this suite of tools are capable of true production work. This is after countless hours discussing with developers and artists, hundreds of thousands of lines of code changes in all the software used, sleepless nights testing capabilities and trying to pinpoint annoying bugs. Not to mention countless collective hours of rendering test images. Now that things work and things were stable enough to truly work on something with merit, Murphy's Law dictates that everything this depends on will be totally and forever changed.

When Blender 2.5x series was announced it was clear that we too had to evolve and try to work harder than ever to bring our wishes and requests to the table and say “Hey we will do the hard work, you just need to make it possible for us to do it” and we think we did a good job because the Blender developers added the RenderAPI, something that was highly desired by not only us but every other user of external renderer's as well as the script developers. Now it’s just a matter of proving to the pros that this is worth a look at too, a task that actually is harder to do than getting the key people in the community to ponder and even work with.

Blender, Aqsis and RIBMosaic are going through their own refactoring, for their own reasons and at their own paces. It is merely coincidental that they all happened at the the same time or within a close approximation. RIBMosaic's future relied on being programmed from scratch again in order to function in Blender. Aqsis needed to be able to handle current rendering methods like ray tracing, global illumination, point based occlusion and more in order to survive in the open source world and in the CG world altogether, now that Pixie's development has come to a near dead halt. Blender needed to be able to provide a RenderAPI so that external renderer's could access the same data that the internal renderer could. Some of Blender's data prior to the refactor were inaccessible, most notably the particle system, thus other parts like animated curves were not able to be rendered either.

The purpose of this is to help understand the needs of Blender in a professional visual effects house for instance. There are a good number of visual effects professionals that have been actively contributing to the open source world, as well as those that are trying to incorporate tools into their own pipelines, so there is effort on both sides of the field even if this number is small in comparison to the rest. The Blender to Renderman pipeline is one thing that could bring even more attention and respect from the professional field of visual effects and animation.
Why Renderman?

RenderMan is not a program in itself, but a rendering interface developed by Pixar in the early 1980's and released publicly in 1989 as RiSpec 3.1. RenderMan is often confused with PRMan, which is Pixar's commercial RenderMan compliant rendering program. RenderMan is actually deeply rooted with Pixar's history, and in many ways birthed the studio and software developer.

History

In the beginning of the 1980's Ed Catmull had brought a bunch of graphics programmers to the Computer Division of LucasFilm, the goal was to computerize the special effects industry in three ways: to replace the optical film printer with digital compositing; replace the physical film editing station with digital off-line editing; and to replace miniature effects with computer generated animation. During that time the fastest computer generally available was the VAX 11/780. The Computer Division undertook three hardware projects: the Pixar-2D, the EditDroid and the ambitious Pixar-3D. The Pixar-3D was a giant image synthesis hardware pipeline. The software prototype of this rendering pipeline was given the name of REYES, which stood for "Render Everything You Ever Saw", but also the name of a California based park. The "Genesis Effect" sequence in Star Trek: The Wrath of Khan was done with an early version of the software, as well as the stained glass man sequence in Young Sherlock Holmes.

Around this time John Lasseter had joined LucasFilm for a few months, on behalf of Ed Catmull's request, to experiment with computer animation. Several months later the first Renderman based short "The Adventures of Wally and Andre B" was completed. Pixar was spun off of Lucasfilm in 1986, when Steve Jobs bought the division.

The Pixar-3D hardware project was eventually named the "Reyes Machine". This machine was an expensive piece of hardware, with a series of special purpose boards that each accomplished one part of the Reyes rendering pipeline, leading to the RM-1 Project, which was a lower cost hardware machine that thought to replace the very expensive "Reyes Machine". Eventually the idea to have a standard interface like Silicon Graphics had with their hardware (GL), and started to develop the next generation GL, called the "Rendering Interface," or RI.

At some point in time the idea that the entire Reyes Machine would be small enough to be compared to a Sony Walkman, that one could have video piped directly to video sunglasses (it was the 80's). This was the birth of RenderMan. After that the "RM" in the RM-1 was simply understood to mean "RenderMan", and Pixar started to try to sell RM-1 machines to anyone who need high speed rendering services; mainly Industrial Light and Magic. It was decided by an advisory group of 20 big name companies in the computer graphics industry to announce that "RI" was the PostScript of 3D. There was only one snag when Steve Jobs decided the "Rendering Interface" was not "snazzy" enough. They needed to come up with a new name. So thus all the hardware and software turned into RenderMan; RenderMan Interface, RenderMan-1 Hardware, RenderMan Toolkit. Sadly the hardware never made it out of beta testing, and it was killed by the inexorable march of Moore's Law and intristic unsupportability of special purpose hardware.

In an attempt to demonstrate the merit's of the Pixar Image computer, Lasseter completed the "Luxo Jr" short, which not only impressed computer scientists, but the film industry as well. "Luxo Jr" was nominated for "Best Short Animated Film" by the Academy Awards in 1986.

Pixar released it's next product that was the software version of the Pixar 3D hardware, dubbed PhotoRealistic RenderMan. Over the next several years Pixar sold over 100,000 licenses of the software that was generally not marketed for consumers.

In 1988 Pixar released the RiSpec 3.0 to the public. While this was a document that was publicly available, the only available software using the RiSpec was PRMan until the mid 90's when a student by the name of Larry Gritz wrote BMRT (Blue Moon Rendering Tools), a free alternative to the pricy Pixar software.

Over the next decade several commercial and free rendering software packages became available, all RenderMan compliant, some having features not found in PRMan (such as BMRT's ability to ray trace at the time). Some have been made for personal use, while others such as 3Delight and AIR made for professional production.

RenderMan has been credited for bringing computer graphics to the big screen, with it's speed, accuracy and expandability. Since the early 90's more and more films are being made with computer animation and effects in them, most
of the time using PRMan. Since 1995 and the release of Pixar's "Toy Story", once again PRMan proved to the film industry that full CG films can be done. [1]

**RenderMan Interface Specification**

What set the RISpec apart from other standards of the time was that it allowed using high-level geometric primitives, like quadrics or bicubic patches, to specify geometric primitives implicitly, rather than relying on a modeling application to generate polygons approximating these shapes explicitly beforehand. Another novelty introducing by the RISpec at the time was the specification of a shading language (RSL).

The Renderman Shading Language allows material definitions of surfaces to be described in not only a simple manner, but also highly complex and custom manner using a C like language. Using this method as opposed to a pre-defined set of materials allows for complex procedural textures, new shading models and programmable lighting. Another thing that sets the renderers based on the RISpec apart from many other renderers, is the ability to output arbitrary variables as an image—surface normals, separate lighting passes and pretty much anything else can be output from the renderer in one pass.

**Required capabilities**

For a renderer, in order to call itself "RenderMan-compliant", it must implement at least the following capabilities:

- A complete hierarchical graphics state, including the attribute and transformation stacks and the active light list.
- Orthographic and perspective viewing transformations.
- Depth-based hidden-surface elimination.
- Pixel filtering and anti-aliasing.
- Gamma correction and dithering before quantization.
- Output of images containing any combination of RGB, A, and Z. The resolutions of these files must be as specified by the user.
- All of the geometric primitives described in the specification, and provide all of the standard primitive variables applicable to each primitive.
- The ability to perform shading calculations through user-programmable shading
- The ability to index texture maps, environment maps, and shadow depth maps
- The fifteen standard light source, surface, volume, displacement, and imager shaders required by the specification. Any additional shaders, and any deviations from the standard shaders presented in this specification, must be documented by providing the equivalent shader expressed in the RenderMan shading language.

**Optional advanced capabilities**

Additionally, the renderer may implement any of the following optional capabilities:

- Area light sources
- Bump mapping
- Deformations
- Depth of field
- Displacement mapping
- Environment Mapping
- Global illumination
- Level of detail
- Motion blur
- Programmable Shading
- Special camera projections
- Spectral colors
- Ray tracing
- Shadow depth mapping
Production Requirements

Visual effects houses and animation studios rely on Renderman not because of it's popularity, there are plenty of other popular rendering software that have been around and in use for quite some time. Renderman was designed for one purpose, film rendering, so the rendering specification is built around the requirements of film, what technicians and engineers need and to do so reliably. Below are some of these reasons Renderman has been relied on for years.

Feature Set

The feature set of Renderman is quite extensive, even the most basic requirements of “RiSpec compliant” is impressive. Basic features like motion blur and DoF is a must. The input of a variety of geometry, not just polygons but NURBS, points, curves and other parametric curved surfaces. Atmospheric effects, shadowing, texture mapping and control information for post processing compositing.

Flexibility

Renderman is a highly flexible rendering specification. It contains a shading language, which allows users to have complete freedom to define how the renderer will produce an image. Renderman is capable of producing the most “photorealistic” of imagery for film, yet it is also capable of recreating the look of traditional cel animation, or even abstract rendering. Flexibility also encompasses the image settings of Renderman, one can adjust the amount of sampling and detail of a rendered image, even individual elements or objects can have their own setting and allows for complete control to increase rendering performance.

Robustness

The ability to handle 100,000 frames of images is a must for a production renderer. It must not crash, it must be capable of handling large amounts of data and be able to do this over and over again with as little problems as possible. We live in the real world and absolute perfection is impossible but the REYES algorithm was designed for the purpose of film, it was designed to handle the massive amount of data that is produced for film effects, it was designed to be able to handle the needs of technical directors and visual effects supervisors.

Performance

Rendering hundreds of thousands of frames is no easy feat and a production renderer needs to be able to meet the demands of it's users and be able to do this without complaining too much, to do it in a timely matter because films generally do not take a decade or so to finish, so the window of time to render is reduced to mere months and that is only after everything else is finished first. Being able to be used in the pipeline from start to finish with the same level of control, power and performance is a must as well because a vast majority of user preview their work before submitting it for approval. In the final stages the renderer needs to be able to perform, the final frames are what your audience will see so your success or failure rides of that.

Image Quality

Lastly the quality of the image is the most important of any renderer, however in Renderman this is compounded because the majority of the imagery produced will be seen on a very big screen, so the image quality needs to match up to the quality of film. Film is a very high resolution format, more so than HD even, so the requirement of absolute fine detail is a must. Having the ability to control the level of detail or quality of the image is also required. While rendering at film resolution is possible it seems a bit of a waste to preview changes at that scale, so having the ability to have a preview setting with lower quality and smaller resolutions is just as important as having that fine detail.

Visual effects house and animation studios swear by the power of Renderman, however most of them usually talk
about Pixar's Renderman, PRMan when the name is mentioned. However as we have seen in action, there are other rendering applications that can work just as well. This is why Pixar developed the specification in the first place, so that other programs or tools can access the Renderman API, it is just up to the developer to make the renderer as close to the “standard” image, performance and innovation benchmark; PRMan.

One part of the reason for better Renderman integration is the shading system itself. In Blender the artist is restricted to a pre-defined set of shading and lighting models, however in the Renderman world, it is the shader that defines the image and the renderer just interprets it to make a long story short. With a robust shading language it is possible to create any type of surface volume or displacement one can think of, either using texture maps or procedural methods. This makes it possible to create whole new shading models if they wanted but if someone wanted to do the same thing in Blender it would require one to learn C++ and even then the shader is only usable in Blender, Renderman Shader Language source code is generally able to compile on any Renderman software (provided the software has the functionality). While working with Renderman is not always an easy road, the pay off is that the imagery produced can be awe inspiring and well.... it kind of already has been.

The Widow Pipeline

A pipeline can mean many things in a visual effects or animation studio, anything from concept art, software, methods or even rendering, in this case the pipeline is referring to the 3D process from model to render. In a professional studio environment there are many tools that are used to help with this process, some of them in house tools, some of them commercial software that has been heavily modified and then some of them have been open source. Studios are also in turn releasing some of their own tools or libraries as open source, such as ILM, Imageworks and Disney, so there already is at least an attempt to drive innovation rather than stifle it. This is an important link as the open source community and the professional industry can develop technology that is compatible with each other. This is one of the reasons that Pixar had released the RiSpec to the public, that allowed others to develop their own Renderman based software, thus this is how Aqsis and Pixie came into being. This is also how OpenEXR ended up in not only open source applications but nearly every single commercial application that is used for visual effects work.

When studios develop a pipeline many things have to be considered before production begins, otherwise situations will arise where this can grind even a short film to a temporary halt until those problems are resolved. This could take hours, days, weeks and in worst case scenarios even months, which will cause time, creative and financial loss. It will not matter if the pipeline is built in a studio or if it is an internet based collaboration, the pipeline has to be solid and stable from the ground up. This includes hardware, operating systems, programs, libraries and any other little miscellaneous scripts that may end up being used during a production.

System Requirements

Software Requirements.

Sadly there are requirements in order to run the Blender to Renderman pipeline, simply because the software that is stable and production ready are considered obsolete in the open source world, the newer versions are not stable and anything below a certain version just will not work at all. In other words the pipeline is touchy but once configured right usually will run without problems. At the core of the pipeline lies these five pieces of software, the rest of the pipeline is customizable and can be built as big or as small as needed. One thing to remember is Python makes up a good chunk of the entire pipeline and having the right version is an absolute must, this is why Project Widow is not using the latest Blender 2.5x builds.
| **Python 2.5.4**  
http://www.python.org/download/releases/2.5.4/ | There MIGHT be debug issues for developers if we don't all use the same Python version. Blender and Mosaic need at least 2.5.4. Though if using Windows it is REQUIRED to use Python 2.6.2 with Blender 2.49. |
| **Blender 2.49b**  
http://www.blender.org/download/get-blender/ | Depending on the operating system used, either Python 2.5.4 (Linux) or 2.6.2 is to be used. Since Mosaic 0.5.1 was built for Blender 2.49b it will only make sense to just use that version from now on. |
| **Mosaic 0.5.1**  
http://ribmosaic.cvs.sourceforge.net/viewvc/ribmosaic/mosaic/ | The latest version of Mosaic, which at 0.5.1 is the final release for the Blender 2.4x series. This final release contains bug fixes and is designed around Blender 2.49b. |
| **Aqsis 1.6.0**  
http://download.aqsis.org/builds/testing/ | 1.6.0 is the production release of the Aqsis project, though work is well underway in the 1.7.0 developmental branch, including the complete core rewrite and plans for the more advanced and modern rendering methods. |
| **OpenEXR 1.6.1**  
http://www.openexr.com/downloads.html | The reason for OpenEXR is to take advantage of the awesome file format developed by ILM, already development and testing done for Project Widow resulted in Aqsis having the MultiLayer EXR display driver, one that can interface directly with Blender. |

## Hardware.

Most 3D artists do not need to be told how to build a workstation, or at the very least able to understand that the more computing power you have, the faster you can create art. This may not always be the case however this is not an argument on what hardware is better, it is an attempt to fully understand some of the things you will need in order to really harness this particular pipeline design using open source software.

## Workstations

This is always a debate on public forums over what is better than what, issues with what software or hardware conflicts, even down to brand names. However the fact of the matter is that a 3D workstation should be as powerful as you can make it. Depending on the funding level, a pipeline can run on a single workstation, or a network of a dozen or more, either locally in a single location, or distributed over the internet, this pipeline can pretty much work on any hardware built in the past 5 years, though the newer the better.

First thing is to get as many cores in a CPU that funding can allow, this will only be for your benefit. RAM is another must to grab as much as possible, there is no reason other than price to not get as much number crunching power as possible. Then a GPU card, in my experience Nvidia seems to be a bit more Linux friendly than ATI, though the same could be said for Windows.

One area of research that has been of interest to many artists are the real time re-lighting tools, such as the Lpics system that Pixar developed [5], or Lightspeed by ILM [6] and even recently RenderAnts [7], which is of great interest because not only was Elephant’s Dream content used for the presentation material but Blender and RIBMosaic were used to export Blender scenes into RIB. This is an area that will see some use in the future and for certain will become seen more over time.

Other than the obvious high powered system for a workstation make sure to invest in good monitors, as this can affect your rendering output greatly if the display gamma is not the same as the rendered output. It is difficult to really say which model or type is best because I certainly do not know. Larger monitors are always a must, I like widescreen monitors to view the HD resolution frames that Widow will be rendered at, especially if I can get 1920x1080. This is all personal
In the end the workstation will cost per system a lot more than any other system you or your team will be using for whatever purpose. Do not be shocked to see prices of complete systems running close to $1,000 or more depending on hardware configurations. Make it worth the investment but make it upgradable too.

**Renderfarms.**

Renderfarms are common place and often required in order to render film resolution cg imagery, otherwise even 30 seconds of frames could take months to complete using just one computer, depending on the complexity. Most renderfarms are comprised of rows of racks, each rack loaded top to bottom with servers and sadly no matter what kind of deals you get with hardware vendors, or even luck out on eBay, renderfarms are not cheap. If you have the funding to invest in a farm of 100 servers then you have little to worry about, however most of us do not have the kind of money, nor the storage space needed to house the farm, then you need to factor in air conditioning and the money draining electric bill when all them servers start cranking away at the next epic film. We can build smaller scale ones however at a modest budget, there is no getting around the hardware cost and if you have the space to house such a project well then you can build your own renderfarm.

Most renderfarms are custom tailored so there is no true way to build nor develop that part of the pipeline, usually it is just what works with the tools involved. There are many render job managers as well, both open source and commercial, so there are plenty of options to go with. Project Widow is using Dr.Queue for our rendering jobs, which will be done on the Animux renderfarm, a fairly well known renderfarm in the Blender community as it was used to render several short films and tutorial animation sequences. Our landing the reservation of a massive render job on this farm was the result of my work at working out the details of the Blender to Renderman pipeline, I also used this distribution during the early months of production, some of this work produced new tools as well, since it involved the renderfarm itself and how to ensure exported RIB files were being rendered correctly and in the manner that we wanted.

One major factor to consider is architecture uniformity in the renderfarm, one should try to keep all renderfarm hardware in the same configuration as changes in this can result in artifacts of the renders only really noticeable when viewed at full frame rate. The main area of concern is the CPU architecture should remain consistent across the board, each architecture type compiles software a little different and since the rendering software is compiled for that architecture any changes will affect things like procedural texturing and volumetrics. Mixing a 32 bit with 64 bit can do this, or if you have older SPARC and MIPS systems in the farm then these subtle artifacts will show through. The result will look like flickering when using fractal, turbulence or noise functions and when projected on a large screen becomes highly noticeable and distracting, not to mention it just looks awful and unprofessional. If you do have a mixture of hardware then it can be a simple matter of assigning certain jobs to certain render nodes, or using certain software on each type, or even types of jobs like fluid and cloth simulations on one set of nodes and rendering on another at the same time. This is also known as render queue management.
Render Job Managers

Here is a list of the most commonly known render queue management software, both open source and commercial.

**Freeware / Opensource**

<table>
<thead>
<tr>
<th>Software</th>
<th>Website</th>
</tr>
</thead>
<tbody>
<tr>
<td>DrQueue</td>
<td><a href="http://drqueue.org/cwebsite/">http://drqueue.org/cwebsite/</a></td>
</tr>
<tr>
<td>SunGridEngine</td>
<td><a href="http://wwws.sun.com/software/gridware/">http://wwws.sun.com/software/gridware/</a></td>
</tr>
<tr>
<td>Afanasy</td>
<td><a href="http://cgru.sourceforge.net/">http://cgru.sourceforge.net/</a></td>
</tr>
</tbody>
</table>

**Commercial**

<table>
<thead>
<tr>
<th>Software</th>
<th>Website</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deadline</td>
<td><a href="http://www.franticfilms.com/software/products/deadline/overview/">http://www.franticfilms.com/software/products/deadline/overview/</a></td>
</tr>
<tr>
<td>Muster</td>
<td><a href="http://www.vvertex.com/">http://www.vvertex.com/</a></td>
</tr>
<tr>
<td>Qube</td>
<td><a href="http://www.pipelinefx.com">http://www.pipelinefx.com</a></td>
</tr>
<tr>
<td>Rush</td>
<td><a href="http://seriss.com/rush/">http://seriss.com/rush/</a></td>
</tr>
<tr>
<td>Smedge</td>
<td><a href="http://www.uberware.net/">http://www.uberware.net/</a></td>
</tr>
</tbody>
</table>

The main reason for the Widow pipeline specifically using DrQueue for render job management is not only is it stable but it has been used in several films over the years, as well as other applications. [2]

**Companies, Institutions and Professionals**

- NASA Goddard Space Flight Center
- Martian Labs (Linux/IRIX farm running Houdini & other general computation)
- Bren Entertainment (Filmax & Filmax Animation)
- Swedish Film Effect
- Indiana University
- Gener8Xion Entertainment, Inc.
- Christopher R Green
- Verint Systems Canada Inc.
- New Machine Studios (Mac OS X and Suse Linux Server Farm Running Maya)
- Basecamp VFX
- LeyendaVFX Producciones
- Alan Jones (who also provided XSI support and XSI-Splash)
- Ainkaboot Ltd UK (Clustering solutions)
- CRAVE (DBIT, Mumbai)
- Digital Graphics
- University of Wisconsin - Parkside
Disputes over which operating system is better has persisted since the dawn of home computing, though one thing is for certain is that UNIX has been the operating system of choice for the visual effects and animation industry. Since the early days of CG in media there was one company that helped bring up the industry; Silicon Graphics Inc. This company built high powered UNIX workstations and servers that were well beyond the price range of the average household, instead they were marketed to the science, government and entertainment industries that had the funding to afford them and in large numbers at that. During the early 1990’s, when CG visual effects really made it’s first impact in film, Windows was not exactly capable of handling the data required by the likes of Pixar and ILM, by that time most of these high powered servers were 64 bit capable and Windows 3.1 was still a 16-bit operating system, designed for comparatively low powered home PCs. So this was not a suitable OS for a multi million dollar effects shot, let alone a dozen or more and not much has changed since then, even though Windows has become the OS of choice for 3D gaming. It has to do with the versatility of UNIX, there is usually little porting to different flavors such as IRIX, HP/AUX, FreeBSD, Debian, RedHat, Suse and so on, aside from the differences of the various flavors usually the source code itself is left unchanged.

However this is not to say that this pipeline is not possible on Windows, it is just not the primary choice for this pipeline. It was designed to run in a UNIX type environment so there are certain things that are not possible on Windows, or just easier with Linux. For server based tasks UNIX is a recommended choice while in other cases such as modeling or texturing tasks it really depends on the software used at these stages of production, so if there is a single Mac system with the only license for Photoshop, it wont make sense to ditch it for GIMP, as it may be preferred to use Mac OSX for texture work, compositing or editing and other tasks that this OS is renowned for. For the rendering end of the spectrum most should highly consider using an open source OS such as Debian, if only for the reason for reducing cost to hardware alone.
Most rendering applications run on Linux and Renderman is no exception, from Pixar's Renderman and 3Delight, to Aqsis and Pixie. Some take advantage of distributed rendering, where each node is only rendering a small chunk of a frame, such as the case with Pixie and 3Delight. Windows may own the gaming market on home PC's but for the visual effects and animation industry UNIX is the king.

Even in this pipeline certain tools only work in Linux, such as Shrimp, which is the shader tool that has been in use for Project Widow since the summer of 2010, though it is a highly customized version, much of the changes are not in the source code itself but the node library, as well as the RSL function library.

Developing software on a UNIX based OS is also easier to accomplish, as all come with a C compiler and access to development tools and libraries that are fairly easy to port over. Developing software for Windows often involves some major work that may not pay off in the end. The scripting abilities of UNIX shells also play a big factor, not to mention cron jobs and other UNIX specific tools allow for a high level of pipeline automation, reducing the amount of human input as well as error, an automatic backup of a directory every day is a smarter choice than depending on the user to remember.

**Blender**

The whole purpose of this is to prove that Blender is capable of being a part of a studio pipeline, or being the primary tool in an open source pipeline. Either option is possible, for each user of Blender might only need it for a certain series of tasks and that is the beauty of this pipeline, it is very flexible and very little is restricted. This pipeline is designed for Blender 2.49b, though it may be considered obsolete now it is production stable, even in our pipeline much of the bugs and issues seen even a year and a half ago are either resolved or worked around.

One of the major issues with the 2.4x series and below has been certain data is just not accessible, mainly data related to curves and NURBS. During the early days of Project Widow we had run into a problem when trying to figure out how to best represent spider silk strands. The obvious choice was curves, as they are easy to animate and truly looked like spider silk when rendered in Aqsis. However the problems began when the animation of curve points was attempted, the curves retained the same point positions when rendered in Aqsis regardless of what was actually happening. This was a devastating blow to our team, even though it was already suspected, so we had to figure out other ways to make spider webbing. We came up with various options, even curves were used since they rendered fine as long as they were stationary, but in the end any spider silk strands were polygon meshes, some of them cloth simulations, others using bones, some were simple texture maps.

Other issues were not so much real bugs as they are human error, or not understanding certain functions of Blender, or just plain oversight that created problems down the line. Texture maps have been lost on models, due to using Absolute paths by mistake rather than Relative, something that seems backwards, it just makes more sense to use Relative paths by default than Absolute. Human error has been and continues to be the primary cause of technical issues later on down the pipeline, so studios need to take this into account and establish a set of standards and conventions for the entire team to follow otherwise these kinds of issues will persist all the way into post production.

Blender is also highly customizable and it is recommended to change the startup blend file to be set up for Renderman by default. From RIBMosaic preloaded on startup, the various RSL functions in the form of include files, other Python tools that are commonly used for the production at hand and lastly window layout. The window layout can be a huge improvement from the default
This is probably the most open ended part of the pipeline, since the pipeline was designed to work with any RiSpec renderer, from Pixar’s software to the small personal projects like jrMan. In our pipeline we heavily use Aqsis because of our relationship with the developers themselves but the studio or group has the ultimate choice of which RiSpec renderer they will be using. Obviously the best choice would be something that is not only stable but feature rich, however this may not always be the best case, for instance Pixie has a lot in common with Pixar’s Renderman in functionality, or even 3Delight. All three can do ray tracing, global illumination, point cloud irradiance cache and brick map baking capabilities, however Pixie’s development has slowed down considerably in recent years. Aqsis on the other hand has been making huge progress in changing the underlaying framework so that future work can be done in those areas that have been lacking. Studios that want the production proven stability and speed of commercial software are able to use this pipeline just as much as a single person that is just learning about this technology.
**Renderman software most commonly used:**

**Commercial**

- Pixar's Renderman
- 3Delight
- AIR
- RenderDotC

**Open Source**

- Aqsis
- Pixie

Depending on the needs of the pipeline, production and who decides the funding, the rendering software will also influence the pipeline as each have their own unique set of tools and abilities. 3Delight has tools that function similarly to Aqsis but it can do things Aqsis cannot. Pixar has a powerful framebuffer tool, as does Aqsis but each are different in their own way. AIR has a GUI tool that no other have. Even deeper into the actual rendering part, each software have their own unique code to accomplish better rendering methods. If your production requires or desires ray tracing then Aqsis clearly is not the choice one would make, so that limits the software to Pixie or commercial software.

As you can see the choice of renderer has a major impact on the design of the pipeline. The Project Widow pipeline is based off of Aqsis and is specific to it so it takes advantage of the tools included but it also works around the limits as well, even though we are deeply tied to the Aqsis development team we are not directly involved with the source itself.

We did influence some of the enhancements and additions, as well as do some serious testing with large scale scenes but in the end the Aqsis developers ultimately control the development of the software and have every right to reject ideas that do not fit with development targets. Commercial software also function the same way, in it's support plans and ties to development teams in studios, certain ideas are suggested and these studios either develop them, or have the software company add it and then provide the update to them. This is often a tight lipped process, involving non disclosure agreements on both ends. These updates are tested and then included with the next upgrade, along with the bug fixes, so in both open source and commercial markets the process of upgrades, testing and release procedures are very similar, or can be depending on how deep you want to get involved with software development.

Aqsis 1.6.0 is considered “Production Stable”, yet the current release does not have certain additions that were added later to the developmental repository, something we are working with already since this method of shadow maps had been in RIBMosaic for some time. On my system, as well as the Animux renderfarm, are recent builds of Aqsis 1.7.0, which is the developmental version. So in a way the Project Widow software is a mix between obsolete and bleeding edge, referring to Blender 2.49b and the latest builds of Aqsis 1.7.0. This is only because we have setup certain parts of our production to take advantage of the little additions, such as the single point shadow map.

In general the Renderman end is probably the most versatile in the entire bunch, much of the work involved just means setting up environment variables, custom file paths, certain scripts for certain apps. The rendering itself, well you just have to see it for yourself. For instance on my workstation is mainly built for shading, lighting and rendering, the window placement seems to work well for me and with dual monitors this makes it all the much easier to do quick editing.

One important thing to remember when designing this pipeline, is to use whatever Renderman software you choose as your preview renderer during the production. The reason is to keep the mindset to not use the Blender internal for the reason that your are creating content for Renderman, not Blender, you have to think about it from renderer downward. This way after a time, it become natural to use Aqsis for instance, to the point that it takes a second or two more to find the Blender rendering buttons or menu items. Taking a professional approach to Blender to Renderman pipeline design and implementation will pay off in the long run.
Imaging

In the world of 2D image editing there are two applications that come to mind, Photoshop and GIMP. One of them is a production proven commercial application that has a very long history that is deep seeded in the visual effects industry and ILM. The other is an open source application that started as a school project and over the past 15 years has evolved into what it is today. Both are powerful in their own way, they both have their pros and cons and they both have been used in productions of varying degrees. However the one thing that Photoshop has that GIMP does not is the ability to work with more than 8-bit color. This is where the small fork of GIMP called Cinepaint comes in, as it can and has been used in big budget visual effects work, simply because it was developed for UNIX systems since Photoshop was not built for anything other than Mac and Windows, it was born out of necessity. This was before the visual effect backed WINE project, which then allowed the Windows build of Photoshop to run on any Linux system in the pipeline.

GIMP

GIMP’s use in this pipeline has been purely for texture maps, this is where it excels and just as Cinepaint is invaluable in it’s area of expertise. Cinepaint just does not have the amount of tools desired by texture artists, something that GIMP does have despite it’s shortcomings with color bit depth as well as it’s inability to open HDR images like EXR and DPX.

Cinepaint

Cinepaint is not as developed as GIMP in terms of other features, however in the visual effects field it does in fact have powerful uses, in this pipeline specifically it has been invaluable and still encourage the developers of Cinepaint to keep up the work because people like us do use it and rely on it for certain imaging work that is just not possible with GIMP or due to lack of funding to purchase Photoshop.

Both applications are used in our pipeline so there is no reason to not use both in any other pipeline, it really depends on the studio or group to decide if purchasing commercial software that can do it all, or to save money and use two that have their own strengths and weaknesses. In the end imaging work is not a concern for anyone in terms of functionality.
In the final stages of the pipeline imaging is just as important as the rest of the software, this is because certain file formats are suited more for their intended uses, such as using EXR or DPX during compositing, even if the final frame format will be TIFF and eventually compressed video formats like MPEG or DIVX. HDR imaging has become a huge improvement to the open source pipeline, once all the kinks got worked out over the years. The Aqsis developers even improved their OpenEXR output by the inclusion of MultiLayer OpenEXR image output. This format is fully compatible with the same EXR file output that Blender produces, this way one can use a frame sequence of EXR files created by Aqsis that has AOV renders all compiled into a single EXR file, which can be loaded into Blender and have all the correct output points ready for compositing, just as if the image was written by Blender itself. Aside from the actual legwork to set up the composite nodes, the actual image produced requires no additional setup, it is truly plug n play. As far as we know Aqsis is the only RiSpec renderer that actually produces MultiLayer OpenEXR files.

**Version Control**

This is where things get tricky as there are many ways to share production files, however just standard file sharing can become a sticky mess of random files here and there with no order. Certain version control systems like SVN require a central server, others like Git are distributed and do not require a central server thus reducing the chances that ALL data is lost and if one system goes down it does not halt the rest of the team. FTP can be a good way to store files but there is always the chance that someone works on a file and overwrites a crucial update and well we don't want to think about that. It really depends on the project manager, or the person directly in control of how version control will be handled.

| **SVN** | This version control system is very common, having replaced CVS as the “standard” development version control. It is a server / client system and all revision information is stored on the server. This can pose a problem if the server is shut down for any reason, be it maintenance or system crash, the entire team connected to the SVN repository will not have access to updated data. |
| **Git** | This version control system is a distributed system, all clients have a local copy of not only the data but the revision list as well, so there is no central server that controls anything. This eliminates the risk of a total data loss in the event of a worst case scenario, or if the host forgot to pay the internet bill, each user that can connect to the internet can have the same amount of access to the repository as everyone else. |
| **Mercurial** | This is another distributed version control system, similar to Git, however it is implemented mainly in Python. |
| **FTP** | This is the most basic form of storage and has no version control at all, requiring high discipline from the team to make new files for each update to a file, numbered in a sequence to reduce time searching and have absolute clear communication over who is working on what file, to prevent accidental overwrites. This is also server based, so if the file server goes down then nobody has access to the files. |
| **Other options** | There are other ways to share files, depending on the size of the team and production. Some are suitable for small teams, like free file sharing hosts with a limit on available space. Some are better suited for large teams, like version control systems tied in with database software. Depending on the software a lot of these tools can be tied together via Python, something that Industrial Light and Magic did with their own pipeline. Having the ability to link together tasks, software and protocols can have a profound effect on the efficiency of a pipeline, no matter what 3D, imaging and rendering software is used. |

All file control methods have their own pros and cons, it really depends on what you are planning for. With internet based productions it is almost essential to use VCS, however if you are part of a studio with more than 20 employees, it might be just easier to use a network storage server. The needs of the storage and control should be one of the first things
done otherwise migration to other systems can be problematic.

**Production Example:**

When Project Widow started we really did not know how to approach the file control, at first we started with DropBox and it did work out fairly well at first but as time went on and the file numbers increased we lost track of what was edited, as well as why it was edited. Then Paul had started to develop a VCS that was based off of Unison, we called it Arachnid. While this did work it was not very intuitive and most of our team was not used to command line operations. During the summer of 2009 I had helped in the renderings of the Prince Harming to Prince Charming preview, as well as helping monitor the renders for the Hand Turkey Studios 48hr film project, both of which used SVN. So I brought this up to the team and while we had reservations that binary files could get corrupted, ultimately the we began the migration into SVN.

We had set up an email notification whenever a commit was made so we all knew who was doing what just by that alone so already production was a bit more smooth. Eventually this paid off when we were offered project hosting on Shotgun, their API included ways to retrieve information from SVN or other version control systems. By the fall of 2010 our system was finely tuned and now it is only a matter of bug fixes or troubleshooting on very few occasions.

**Project Management**

This is a crucial part of any production beyond a college assignment and even then with the level of complexity even then some kind of management is needed. Project management is not artistic by any means, it is barely technical even but it is an important part when more than one or two people are involved, simply because it just makes planning easier, all members know who is working on what and what needs to be done, that is if you do it right. On Project Widow this only became an issue later on as production got more and more complex and as more tasks got tacked on and this is only a small team of artists and developers, things would be a mess with larger teams and a lack of management. This also helps in the demise of collaboration films when management is planned poorly, people loose interest or get confused about things, files can be modified when they should not be, or there is just no clear direction of where to go and what steps are needed to get there first. Project Widow narrowly avoided these pitfalls and Shotgun Software was a saving grace for the production and planning.
Shotgun Software

Shotgun Software is a project management web based software created by some talented programmers that have worked in the visual effects and animation industries, mainly technical directors and pipeline engineers that understand the needs of 3D artists and wanted to create a powerful system for productions. This companies software has been used by over 150 different studios around the world, including Pixar, Laika and Tippet Studios to name only a few, so this is not just a bunch of college students with a dream, this has been used in many real world productions and is only improving on their success so they know deeply about what is needed by the people that build the pipelines since a lot of the work involved means setting up software and tools, server configuration and then linking everything together in a cohesive manner that makes sense and is streamlined enough to lessen bottlenecks at certain stages.

Shotgun is similar to other project management systems that are used to develop software, however it is designed for the CG industry and that is where the similarity ends. It is also highly customizable and has it's own API so that you can link your own servers to Shotgun, even whole new protocols to call up other software locally, such as calling the previewing software RV for evaluating dailies.

With Project Widow we did a lot of work to customize it to suit our needs. Since we had gotten to use the system mid way through production, we did not have the time to link every single asset to a Shotgun asset, just a database entry really, though this was not really a problem. What we did want to do is to have our SVN server relay commits to Shotgun, so the first order of business before the rest of the team joined was to work out the details with our SVN admin. For a week we tested out the Python script, designed the site and have planned shots.
Tying it all together.

The entire purpose of this of course is to demonstrate the power of all tools combined and encouraging the idea that no one tool can do it all and it should not. There is no need to have a studio grade recording software like Ardour inside Blender, it will only bloat the source code and ultimately be more of a problem than anything. Blender is a 3D tool with some pretty neat tools inside it that can reach into other areas but the core of it is 3D modeling, animation and rendering. Aqsis is a renderer, just as Cinepaint is an HDR image editor and Shrimp is a shader tool. There are also other tools used
like text editors, programming IDE's, server dameans for email, svn, ftp and so on. There is no single tool that is used for everything and in Project Widow's case, much of it was remote and we all had different operating systems and hardware but we all were able to work on the same content.

One of the key needs is some sort of communication system, be it emails, a forum, chat rooms, or all the way to Shotgun with a quite elaborate note system. Logs and manual entry should not be forgotten, there is only so much one can communicate with a single SVN commit log that only has a bare minimum of what was changed. Though in some cases making something a bit more efficient in theory does not always work out in practice and in turn makes things worse. So having some sort of bug tracking system for issues relating to the entire pipeline is a must, this way the person directly in control of the pipeline can make the steps to fixing it, or making the call to get it fixed. For example the Project Widow SVN host was responsible for the upkeep and modifications done to the SVN post commit hooks for Shotgun but the requests and bug reports came from me primarily.

The Widow R+D Shotgun site was developed as a test bed for any customization we did to the Project Widow site, we also had designed a new Feedback and Support page so anyone could let us know when an issue arises, or wants something new.

**Modeling for Renderman**

This is one of the most overlooked process of the pipeline, mainly because a lot of people do not know, or remember that good topology is essential no matter what application is used to render it. In the REYES world things are not so simple at times as models with high poly counts and massive displacement shaders can bring the system to it's knees and most likely extremely high memory consumption and possibly even too much to handle and not one single bucket rendered hours later. It could be the shader, it could be there model, it can be hard to tell later on when production goes into the shading and lighting stages and by that time it is far too late to fix if in fact the model was not optimized at the geometrical level. Modeling requires discipline and finesse that sometimes gets overlooked.

**Quad Polygons**

The reason for using Quad polygons rather than triangle polygons is more related to the rendering end. When Aqsis (or any RenderMan renderer) begins to render a bucket, each polygon is diced up into micropolygons, which each are about a pixel size, then these micropolygons are displaced and shaded using the shaders assigned to it.

It is actually far more efficient to use Quad polygons than Triangles because of how the REYES works, since it does slice up the geometry there is less work the renderer needs to do with a quad as it is an even distribution of micropolygons. With triangles, there is more work involved as the dicing process needs to figure out the best way to do this, as triangles are an odd number of vertices there is no even distribution of micropolygons.
There will be instances where triangles are unavoidable, Aqsis CAN handle triangles, it just would take a little more time to render rather than with quad geometry. Regardless of what type of polygons used, high poly count models will increase render times. Sometimes this is not possible to counter, as a modeler it is important to fully understand these concepts and practices.

**Subdivision Modifiers**

When modeling objects in Blender keep in mind that objects do not need to be highly detailed geometry in order for it to render as such. In other words, depending on the object it is far easier for both the artist and Aqsis to keep it as simple as possible. Walls, floors and ceilings do not need to be more than a few polygons if they do not have to, though it may require a SubD modifier applied to them in order to prevent what is known as "grid cracking". The objects that require a fine edge while still remaining smooth will need some editing in Blender to crease right. Using best judgment on poly count is essential when applying SubD modifiers to objects, if one were to visualize what Aqsis "sees" when it renders a SubD sphere you could turn up the Level to 4 and that would look similar for instance. While in the literal sense this is NOT how Aqsis dices up geometry, that is something for modelers to remember: more polygons does not mean better. In many cases more polygons means that Aqsis has to work harder at something it doesn't need to.
Figure 6: Blender 2.48 mesh with SubD modifier attached

Figure 7: Aqsis 1.4.2 SDS rendering error that was later fixed for the 1.6.0 release
Quad geometry is ideal for this as it is much easier for Aqsis to dice up, rather than triangle geometry. Sometimes it is not possible to do so, in which case make the best of it. In those cases using Aqsis to preview the mesh should be important, as opposed to the Blender Internal renderer which is designed to handle the geometry generated in Blender (obviously).

Aqsis can handle any geometry thrown at it, even things not FULLY exportable from Blender (such as NURBS surfaces which in that case it exports unpredictably and not accurate at all). For instance the following is a list Poly / SDS data and Curve data that RIBMosaic can handle.

**Poly / SDS Data**

- Multiple materials per object.
- Per face smoothing.
- Optimized for all smooth faces on SDS geometry (RenderMan default).
- Optimized for all flat faces on mesh geometry (RenderMan default).
- Per face UV mapping.
- Whole object uses materials diffuse color for RiColor.
- Whole object uses materials alpha for RiOpacity.
- Per vertex colors.
- Per edge creasing for SDS geometry.
- Support for animated shape keys and armatures.
- Can export most modifiers if used properly.
- Support for datablock instancing/sharing.
- Meshes can be turned into point clouds by using the "Halo" material with the "HaloSize" controlling point size.
- The objects bounding box can be exported if object is using the "Bounds" drawtype (this is helpful for seeing how far to adjust the displacement bound).

**Curve Data**

- Support for ribbons with curves "extrude" property.
- Support for "Tilt" on ribbon control points.
- Support for per control point widths (percentage of "extrude") using radius.
- Support for base and tip width control for non ribbons with materials "HaloSize" and "Add" controls.
- Support for both free and aligned control points.
- Support for bezier handles.
- Support for cyclic curves (periodic, some renderers don't support this).
- Support for both Blenders 2D and 3D control points.
- Support for multiple strands per object (they will be combined into a single RiCurve call).
- Support for multiple materials per objects.
- The objects bounding box can be exported if object is using the "Bounds" drawtype (this is helpful for seeing how far to adjust the displacement bound).

**Preview using Renderman**

It might be a good idea to use Renderman to make periodical preview renders in case there is something wrong with the geometry, sometimes it is possible that faces are not built right for Aqsis to dice up (or most RenderMan software). Sometimes this can result in cracking, which looks very bad in the final renders and can be a huge pain to fix later on. Sometimes it won't show up until later, or at a camera angle different than what you were using for modeling. Try to do preview renders at ALL angles to avoid such complications later in the process.
Materials

When modeling an object make sure that Blender materials are assigned to geometry that has a different look than other parts of the geometry, such as if there is half one type of Material that is to be a metal look to it and another Material that has a dirt look to it. This will save a LOT of time later on when doing the shader and texture process, especially with complex geometry. It is not required to make separate objects in Blender for each type of Material, though if you do decide to do so make sure it is Parented so when moving them around these objects remain in place. The reason for a Material assignment is so that later on there is not a complication when changing custom shaders or textures, it just makes the rest of the process smoother to complete.

One of the advancements made in recent years with Blender to Renderman exporters is the individual Material per Poly in Mosaic. With previous plugins it was simply not possible to assign Blender Materials to individual polygons and render them. The shader was bound to the Object, not the Material in Blender. With Mosaic this has changed and it is possible to have multiple materials per object, thus it is possible to have multiple Renderman shaders per object. This dramatically helps the shading process plus it is far easier to reassign these polygons to new Materials. In the end this means that a smoother and faster shading stage, not to mention beautiful renders.

Texturing

Texture maps usually is one of the more important steps in any 3D project, one can easily make a sub par model look spectacular with well made texture maps, or can make a finely detailed high quality model look less than stellar if these maps are poorly designed. In most cases it is preferred to create maps over procedural texturing, in cases such as characters, environments and even effects. Blender has quite a large selection of formats that it can read and write, however not all of them are suitable for production purposes, in many cases Blender users often choose formats such as PNG to save on disk space or JPEG which is one of the worst formats to use for texturing in any situation. One of the aspects of a Blender to Renderman pipeline is texture formats, the image has to be consistent across the entire pipeline, otherwise problem arises later on and having to fix these details can be time consuming.

| TIFF | The TIFF format is the most common file format used in 3D production. Renderman renderer's support this format, from Pixar's Renderman to Aqsis, it is a format that can work for any aspect of production. One of it's faults however is that file size can be quite large, it is not uncommon for a single texture map to occupy 10 – 50 MB of space and if a single model has multiple maps then you are looking at quite a large collection of maps per object. While hard drive space usually is not a problem these days, these large texture maps can create bottlenecks over networks. One solution to these problems is file cache's through networks, something that only a few Renderman renderer's support, such as 3Delight and Pixar's Renderman. The power of open source is present here too with the Jupiter Jazz Group releasing code for this very issue, it is up to developers to add this into the source code, otherwise it has to be programmed from scratch. |
| OpenEXR | This format was developed by Industrial Light and Magic in the late 1990's and released as open source in 2001, a surprising move that started a revolution. Since then most 3D applications have added support for this format and can be used in many areas of production, from texture maps to compositing, in many ways this has become the new HDR standard, replacing the Radiance format. |
| Ptex | This image format created by Disney is quite unique as it is not just an image format but how it works with 3D geometry. Instead of using UV maps, the library converts each polygon into a UV map of it's own, then the image it writes will be a mass of small tiles that attach themselves to these polygons. This allows for detailed texture maps that cannot be achieved using normal texture mapping, thus it is possible to have ultra detailed maps while zoomed in really close to a model, yet still keep memory consumption down, without sacrificing power or speed. It was also designed with Renderman in mind. Though keep in mind that while Blender 2.5 now has limited Ptex support, Aqsis does not. Nor does RIBMosaic or any other RiSpec plugin for Blender export to this format at this time. |
RIBMosaic and Texture Maps

The Mosaic surface shader as of version 0.5.1 has been a finely tuned shader that not only has custom shading models that nearly replicate Blender's, it also has a very tight integration with Blender's texture mapping, so tight that by itself the shader is very hard to work with. The token system is quite complex, yet once you get the hang of it, quite easy to build shaders around Blender's Material system, however with the default Mosaic surface shader nearly every single part of Blender's material is linked to the RSL equivalent, so when you make a change in Blender, this is reflected in the render. This only covers shading model parameters and texture maps, this does not apply to procedural texturing which is not possible to directly translate. In order to do something like that, one would need to create RSL code that can nearly replicate Blender as close as possible, something of a quite large task considering the current Mosaic shader code is 821 lines long, without comments.

So one should really think of the Mosaic surface shader as a texture map shader, or a basic surface shader until the production shaders are added, other wise the true power and beauty of Renderman is lost. Which brings up the next topic, Shading.

Shading Pipeline

Brief Introduction

Five years ago there was no such thing as a shading pipeline in the idea I had for what eventually became a reality, it simply did not exist like it does today. At that time the only really powerful shader tool available was Shaderman 0.7, which was enough to force me to learn about Renderman enough to make some really nice shaders. This continued on for several years and even into the first half of Project Widow, there was just no clear idea of what was needed to make shader development as easy as possible, yet powerful and expandable as possible so that professional TD's would appreciate it and maybe even use it.

It is a combination of a shader builder, a massive RSL function library and linking the important parts into Blender.

Shrimp

Shrimp is a visual shader editor that has undergone a rebirth of sorts. This software has actually been in existence for around 10 years but for many of those years had ceased to update at all. A couple years ago some people took the source, changed nearly everything about it and actually ended up creating one of the most powerful and versatile shader building tools out there. It is in the function library, which has a collection of the majority of the standard functions seen over the past
20 years, such as noise.h, fractal.h and patterns.h. The thing that makes this library so powerful is that it also includes a lot more than that, the code itself is structured in a way that is optimized and makes sense, it is something that with some tweaking and changes to the node source could be the key to making Project Widow a reality.

Until the spring of 2010 we had been using other shader tools, pretty much all the free or open source shader tools to create the shader source. The problem was not that the shader tools were not powerful enough to do what we wanted, it was for the lack of AOV support, as we made the decision to take advantage of the new EXR display driver this was a massive problem. When you try to render an image with shaders that do not have AOV output parameters and have these outputs declared in the RIB, the renderer gets confused and renders an image that is beyond recognizable, such as seen in Figure(xx). RIBMosaic has these parameters included in the shader source but the custom shaders we were building did not so the frames either needed to be rendered in one pass or we needed to develop a better solution that allowed for optimal performance but total freedom to create without limits.

During this time I thought it might be a better idea to take existing code and just improve it, so I went through and checked out which software would be the easiest and quickest to work on. The last shader tool I checked out was ShaderLink, which was a brand new shader tool compared to the rest. However it did not have much for nodes and was buggy, this was the perfect playground however when checking under the hood and figuring out the code. ShaderLink was developed with Python and uses PyQT for it’s GUI. First task was to fix the GUI bugs, then once that was fixed the next task was to port over shader node code. The one thing that is common in all these shader tools is that they all use some sort of XML file format for the nodes, all that
was different were the parameters and structure of the XML code. Converting the RSL code was easy actually and a lot of the nodes found in Shaderman, Sler and Shrimp were successfully ported over.

The result of the modifications is WidowShade (Figure 7), a heavily modified fork of ShaderLink that may never see the light of day beyond the directory in the Widow SVN it is placed in. The problem was that there was no clear way to add AOV functioning into the program without a massive overhaul to the entire program, something nobody wanted to undertake since this was only meant to be a way to work on a short film, we did not want to take away the time and energy on Widow for a shader tool when there was another tool that already could.

Shrimp became the tool for the pipeline at that point, however it too needed modification in order to really harness the power of Blender, Shrimp, RIBMosaic and Aqsis. It was not that any of the tools could not perform as expected, the problem was linking all the small detail together to form a shading pipeline that was versatile, powerful, expandable and usable by any RiSpec renderer period. This is part of the Blender to Renderman goal, to make as much of the software usable with each other, creating limits between commercial, in house and open source software only causes problems and not useful for anyone, not to mention counter productive.
The only problem was the header files of Shrimp, some of the file names were simply too long for Blender to process in the Text editor, for instance `Shrimp_rsl_shadingmodels.h` was a huge culprit and the key include file since it contained every single shading model that Shrimp had nodes for. So a small amount of time went into editing all the original Shrimp header files into a more generalized RSL library. Thus was born the RSL Library v 0.1.

The shader tool works like this, the program itself only makes calls to the renderer for anything beyond the GUI and basic functioning. The node library is where the RSL code is located, each input and output of the nodes have an internal parameter that the shader tool recognizes, so when you connect the nodes together and compile the shader through the shader tool, the RSL code is pieced together by the shader tool to form one shader, thus allowing any Renderman software to use it. How elaborate the shader code is depends on how elaborate the nodes are and what you are planning to do.

Another key aspect of the RSL library is that it has uses beyond the pipeline, for instance any commercial RiSpec renderer could include this library, since it does contain the very same functions already needed but is structured far better. Same could be said for any software that interfaces with Renderman, including Blender, K3D, Ayam, Maya, 3DSMax, Lightwave, even to the far reaches of Pixar itself with it’s in house tool Marionette.

Making the RSL library connection was easy once the nodes of Shrimp were edited to include the renamed include files, these functions were then added into every scene file that Project Widow had, as well as included into the default startup blend files so in the future any files created won’t need to be configured, just load up Blender and go. This library is also part of an open source project, hosted on Google code but may become it’s own project soon.
### RSL Library

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rsl_aov.h</td>
<td>Contains the AOV output declarations, includes both variables for Shrimp and RIBMosaic</td>
</tr>
<tr>
<td>rsl_color.h</td>
<td>Color conversion functions</td>
</tr>
<tr>
<td>rsl_conversion.h</td>
<td>Coordinate conversion functions</td>
</tr>
<tr>
<td>rsl_data.h</td>
<td>Contains functions and table data for glass IOR and Lafortune model</td>
</tr>
<tr>
<td>rsl_extrapatterns.h</td>
<td>Pattern functions</td>
</tr>
<tr>
<td>rsl_extras.h</td>
<td>Contains various functions for shading such as Ward and Anisotropic distribution</td>
</tr>
<tr>
<td>rsl_fractal.h</td>
<td>Noise and fractal pattern functions, as well as filtering functions often used by them</td>
</tr>
<tr>
<td>rsl_helpers.h</td>
<td>Contains math constants like PI, EPS, SQR and Cosine</td>
</tr>
<tr>
<td>rsl_odwikicomplex.h</td>
<td>RSL version of odForce Wikicomplex.h, contains complex math functions</td>
</tr>
<tr>
<td>rsl_sampling.h</td>
<td>Sampling functions, such as Blinn PDF and Ward isotropic PDF</td>
</tr>
<tr>
<td>rsl_shadingmodels.h</td>
<td>Shrimp specific shading models (at this time), with diffuse, specular, fur and glass types</td>
</tr>
<tr>
<td>rsl_shapes.h</td>
<td>Functions to draw various shapes</td>
</tr>
<tr>
<td>rsl_texture.h</td>
<td>Texture mapping functions, from projection types to environment</td>
</tr>
</tbody>
</table>

https://code.google.com/p/projectwidowopensource/source/browse/#svn/trunk/RSL

The library is still buggy as the shading model functions are still heavily tied to Shrimp's source code, though much effort will be done to make the functions as universal as possible, the rest of the library is usable. The AOV functions too are also tied to the shading models, however they can be used by itself if you are able to program RSL well enough.

Why was the RSL library created in the first place? Well if anyone who has used commercial or open source Renderman software will notice that there are a set of include files that are pretty standard, files such as fractal.h, noises.h and so on. While it is not a problem it seems that for such a powerful language it seems silly to not treat it with the respect and structure it deserves. Shader writing should be treated just like any other programing language, using the same tools if editing the source itself, plus with the shader building tools in the pipeline, makes for a very sophisticated and powerful rendering pipeline. Much of these functions are just taken from the public domain and structured so there is little in theses files that we have not used before it is just easier to use a standard library rather than including a bunch of little functions scattered around, like the functions are pieced together with duct tape and RSL is worth more than that. Consider it an RSL framework to build your shaders upon, using whatever RiSpec renderer you want, as long as the include files are where the renderer can see it then there is no reason not to use them.

The better one is at understanding Renderman Shader Language the easier it is to make powerful but efficient shaders. The scope of RSL is beyond this paper though, requires some reading to understand the full power and there are several books available that are excellent sources of information.
Compositing

Compositing Methods

There are two ways to composite images, one of which uses image layers with various elements, masks, effects and filters applied with the final composite rendered to the final image, or by node based software which is actually more common and preferred in the visual effects industry. Blender has a node based compositor, so people experienced with commercial software will be able to quickly adapt to the way it works, as well as people who want to learn more about node compositing can with Blender before moving on to the professional grade software such as Nuke.

The reason a pipeline needs to have a composite program is simple, to reduce the amount of rendering elements that do not need to be repeated. Post processing filters also are applied, as well as any corrections that can be done without rendering hours worth of frames when a simple change can be completed in a few minutes. Why spend hours rendering a specular color change? It does not make sense and wastes time and in the professional field, money. No professional pipeline is without a compositing application and neither does this one.

In visual effects work compositing can take up the majority of time, effort and money because it often involves combining live actors and sets, greenscreens, CG elements in several layers, filtering, color correction, masking out unwanted areas and tying it all together to form the final frame that is seen on the big screen. These elements can reach in the numbers of hundreds and can often take weeks or months to complete per shot. There is a reason Industrial Light and Magic receives the amount of funding it gets, the skill and perfection they put into each and every shot.

In CG animated films however things are not so complicated, as everything is a CG element of some sort and it is often a common mistake to think that everything is rendered on screen at once. Sometimes this is the case, if the shot is not
as complicated or does not require it, or in some cases would be impossible to composite well (which honestly is very rare). Other times compositing will be required to reduce the amount of rendering needed. For instance it would be far more efficient and faster if you have a static camera to only render the background and environment first (in layers if environmental objects are both in front and behind the subject model) then anything else that is animated next. Or for instance rendering shadow map passes for a sequence of frames, freeing up time to render out the rest, if the shadow maps are too dark, it can be changed later on instead of rendering that series of maps again. Compositing is a crucial process and even at the basic level will save hours of time.

**RIBMosaic AOV and MultiLayer OpenEXR**

One of the things that the Project Widow team had requested from the developers of both Aqsis and RIBMosaic was AOV output and the ability to import the EXR file into Blender and have the least amount of setup work to do to complete a shot. So the idea began and between the two, a direct link between Aqsis and Blender was created via the MultiLayer OpenEXR display driver, which is a great achievement considering that all one needs to do is load the single file into the compositing nodes and the AOV output layers rendered by Aqsis and automatically linked to the correct type of layers that Blender can directly read. There is no need to load multiple image sequences and then connect them to the correct nodes, it is all in one single image that Blender can read and treat as it’s own.

For Project Widow we decided early on that we would be using Blender as our compositing application, mainly because at the time this was pretty much the only option we had considering the format of final frames we wanted to use. There is a distinct lack of quality open source compositing and video editing applications, so our choices were limited to what we had at the time.

Then a new open source project came onto the scene and changed this. That would be Ramen. Not only is it a node based system as well but it can open the very kind of files we were writing. However it is very early in development and there are some things that still need working out.

However the program is a great tool to add to the pipeline, should one not choose to use Blender.
Developmental Goals

As it stands right now it is possible to use Renderman with Blender 2.5x, however the couple of scripts that have popped up over the past year either have only been experimental or not as in depth as desired by professional studios, though the authors of scripts are very talented programmers and they have done an awesome job. The Blender to Renderman pipeline is evolving towards a tighter connection to each other, not only with Blender and Aqsis but with all tools involved plus with a little rethinking of pipeline design and Blender's role in it has changed as well. In the 5 years since the idea of making a Blender to Renderman site and community a lot has changed and it continues to do so even to this day. Sometimes things break between versions, other times software dependencies of a totally different piece of software can change something that breaks another thing in the pipeline, other times the software is buggy. That is the beauty of open source, for example during the writing of this paper, the framebuffer application for Aqsis, which is called Piqsl, would not compile after one of the developers had updated the code in it, so I posted a bug report and two days later the latest source of Aqsis compiled with no problems.

Of course this was because of the active development by the Aqsis team, so fixes and features are only done by those who invest the time and effort into their software. Same with the rest of the tools, much of this pipeline is built with software that is at least active in development or powerful enough to overlook slow development cycles. Some software

![Figure 12: Aqsis 1.7.0 MultiLayer EXR render linked to Blender 2.50 Alpha 2](image)
suffer from this, such as with Cinepaint, which has not seen a lot of updates to the code in the past couple of years. Recently even Pixie has showed signs of slowing down, even with the occasional update once a year the renderer is not improving. This is in stark contrast to Aqsis which is undergoing a lot of changes and improvements, so it is in the world of open source. Where one project runs out of steam, another gathers it and speeds up.

A lot of thought has gone into what needs to really be accomplished in order to really drive this pipeline, what can be done to make it appealing to the technical directors who really are the ones that piece things together, the ones that interface with the renderer, the ones that streamline the pipelines so workflow can get from one stage to another without a major headache, the ones that put the final touches to the finely animated characters seen on screen. Since this is one of the goals of Project Widow it only makes sense to fully invest the time and effort into building the pipeline as it would be in any other place, regardless of the size of the team or project.

Blender has changed so much over the past year alone that it is senseless to really consider any script as complete and RIBMosaic is one of those. As are the other smaller RIB exporters that have popped up, the underlying Python API changes so often that already one of these scripts is broken, the exportman.py script built by Nathan Vegdahl while working on the Sintel short film. During the early part of 2010 he was experimenting with this script, sort of testing the capabilities of the RenderAPI and how it would interface with Aqsis. Some of these renders and tests I have had the pleasure of viewing (though not saving, of course) and even though the process was a bit rougher than what Widow is used to, the point was made, it was officially possible to export to Renderman and the first person to do it was part of the Durian team. Matt Ebb also programmed an exporter as well, again this script has basic functionality but not meant for production use [3].

Aqsis has latched on to this movement and as the official renderer of Widow, they are very intimate with the entire pipeline as they have helped in the design, as well as developing towards linking with Blender, as in the case if the MultiLayer EXR display driver.

RIBMosaic has undergone a huge amount of work to change the underlying framework of how it will work in the Blender 2.5x series and beyond. Eric Beck detailed the problems of RIBMosaic of the Blender 2.4x series.

- Too much direct integration to Blender’s GUI and render pipeline (too rigid and ultimately unsustainable)
- Too much emphasis on adding ALL features from ALL renderers in one system causing marginal performance in all (not to mention over complication)
- Too much hard coded data such as binary calls, display codes, perspectives types, pass types and RIB export
- No access to RIBMOSAIC’s GUI data from tokens
- Not enough access to Blender’s scene and object data through tokens
- Not enough flexibility in the GUI for user modification
- Too many features that are specific to too many different renderers (causing interface clutter)
- Render passes as scenes complicate complex projects, become broken too easily and complicate the exporter
- Too much complication in both the exporter code and GUI for exporter optimizations
- Although custom RenderMan scenes and passes are possible they are not portable between projects
- Most of the more technical aspects of RenderMan cannot be hidden from the user

Planned changes :

- **RIBSets** – Like now RIBSets contain all controls under each Blender tab and will provide a method of logically grouping controls and RIB code together on the same datablock, this allows one datablock to server multiple purposes. They can be manually selected or globally selected by render passes with the RIBSet filter. One small difference however are RIBSet’s will work as XML structures allowing you to add and remove GUI controls in them or assign certain RIBSet’s as default for each Blender tab for any given pipeline.
- **Tokens** – Tokens will go through a major transformation from what they are today and will truly be the “glue” that will make the entire pipeline concept work! One of the first big differences will be Token syntax will be automatically generated according to the naming of Blender’s underlying data, the hope is if Blender’s new API really does expose ALL internal data then tokens should automatically be able to hook to that data. For this reason tokens will be generated by RIBMOSAIC in a menu system similar to the Token Wizard today, however it will contain far more categories.
- **PassSets** – Render passes will also go through a major transformation. First off render passes are going to switch from using Blender scene’s as passes to data structures in the render tab (similar to what Blender uses for render
layers). The concept will be fairly simple, instead of cycling through Blender scenes to export each pass just re-use the same scene cycling through its render pass entries changing the export setup as the data specifies. To make it possible to show and hide objects in each pass, render layers will be controllable per pass (just like Blender’s). The setup options available for render pass will be a combination of most of the controls in the current Scene Setup tab with some of the generic options in the auto render pass dialogs. The thinking here is to provide every generic option necessary to setup any kind of render pass possible in RenderMan.

- **DisplaySets** – These will be viewed as a menu option in PassSets and allow the creation and logical grouping of display code for render passes. The thinking here is you would add all the displays necessary to achieve basic output formats, such as layered exr’s that require a large number of display codes.

- **BinarySets** – RIBMOSAIC no longer requires presets as all necessary setup is saved within the pipelines. Since presets no longer exist this structure makes it possible to save and exchange the binary strings between render passes and pipelines. Just like DisplaySets these will be available within PassSets and like everywhere else tokens will be accessible within these making very complex setups possible with user created controls. These also provide the handy ability to quickly change between custom command line options so for example one could have a BinarySet for calling options for a distributed bucket render, or special console output options or even calling scripts for a render farm manager[1]

Right now a working RIBMosaic for Blender 2.5 is still not available at the time of this writing, however before the end of 2010 or early 2011 the Blender world will see it’s release. Shader development has reached a point where the only thing left to do is to add even more connections between Blender and Renderman, either directly translated which will most likely not happen, or in the case of RIBMosaic and Shrimp this can be accomplished via XML. Both Shrimp and K3D share the shader parameter XML file, this file contains the syntax that creates the GUI elements of a shader control. RIBMosaic will also have this same functionality so when creating a shader project it is simply a matter of using that same XML file in Blender and have direct control in Blender over a custom shader, rather than the current method which involves using tokens and shader fragments to link a Blender Material function to an RSL shader parameter. While it does work in some ways this too also has limits since there are only so much paramters in a Blender Material, RSL shader can contain as many as one can possibly fit or have a function for. The current Mosaic shader was designed to replicate the Blender Material system, complete with all shading models, the texture mapping functions and all the miscellaneous little differences in each renderer, like certain render passes of global illumination are done differently in Pixie and 3Delight. Aqsis cannot currently ray trace so there is another series of commands to make environment mapping easier for technical directors, including automatic creation of Ambient Occlusion, something that even in RIBMosaic’s past was not so easy and required a massive amount of manual work to achieve.

So where to go from here? Well one thing is to keep the Blender RenderAPI in place, stabilize it and keep working on it to make sure that there is future compatibility between Blender and the rendering application, this applies to all rendering software, not just Renderman. In the stable pipeline there are limits to what can be accessed and what cannot, not to mention if there is limited access then only certain procedures and processes can be done, while the one thing you really want to do is not possible. New areas like Smoke in Blender should be possible to access and translated into a format that Renderman can use, either raw RIB as inefficent as it would be, to baked cache files or particle clouds. Focus in other areas as well like Ptex, which could improve texturing like OpenEXR did with HDR imaging.
Volumetrics

Rendering of clouds in the past usually involved modeling a mesh and applying a surface shader that contains a turbulent opacity function, it is a form of cheating and combined with special light shaders that simulate volumetric shadowing, one can create a fairly realistic cloud, the general idea of it anyway. The problem is that this method works well for static meshes but would be impossible to manipulate them like real fluids, so having a real fluid simulation has been an invaluable asset for Blender, it just doesn't work real well for clouds or smoke. The smoke modifier in Blender can be exported to a different format that Renderman could read, in theory the professional industry could offer the solution.

Field3D

Field3D (http://opensource.imageworks.com/?p=field3d) is an open source library for storing voxel data. It provides C++ classes that handle storage in memory, as well as a file format based on HDF5 that allows the C++ objects to easily be written to and read from disk. While the supplied C++ classes map directly to the underlying file format, it is still possible to extend the class hierarchy with new classes if needed. The library was initially developed at Sony Pictures Imageworks as a replacement for the three different in-house file formats already used to store voxel data. It is the foundation for Imageworks' in-house simulation framework and volume rendering software and is actively being used in production.[8]
In early 2010 Walt Disney Animation Studios released the Ptex file format into open source. What makes this image format so special is not due to its source of creation but in what it does for texturing. Ptex stores a separate texture per quad face of the subdivision control mesh, along with a novel per-face adjacency map, in a single texture file per surface. Ptex uses the adjacency data to perform seamless anisotropic filtering of multi-resolution textures across surfaces of arbitrary topology. Just as importantly, Ptex requires no manual setup and scales to models of arbitrary mesh complexity and texture detail. Ptex has been successfully used to texture all of the models in an animated theatrical short and is currently being applied to an entire animated feature.\[9\]

Figure 14: 3D-Coat forum posting days after Ptex release to open source

Figure 15: The Ptex image map as it is written by the modeler
Open Image Input Output (OIIIO)

This is an imaging library that has been developed by Larry Gritz, yes the same one that created BMRT. OIIIO is a library for reading and writing images, and a bunch of related classes, utilities, and applications. Main features include:

- Extremely simple but powerful ImageInput and ImageOutput APIs for reading and writing 2D images that is format agnostic -- that is, a "client app" doesn't need to know the details about any particular image file formats. Specific formats are implemented by DLL/DSO plugins.

- Format plugins for TIFF, JPEG/JFIF, OpenEXR, PNG, HDR/RGBE, Targa, JPEG-2000, DPX, Cineon, FITS, BMP, ICO, RMan Zfile, Softimage PIC, DDS, SG1, PNMP/PPM/PGM/PBM. More coming! The plugins are really good at understanding all the strange corners of the image formats, and are very careful about preserving image metadata (including Exif, GPS, and IPTC data).

- An ImageCache class that transparently manages a cache so that it can access truly vast amounts of image data (thousands of image files totaling hundreds of GB) very efficiently using only a tiny amount (tens of megabytes at most) of runtime memory. Additionally, a TextureSystem class provides filtered MIP-map texture lookups, atop the nice caching behavior of ImageCache.

- Several image tools based on these classes, including info (print detailed info about images), iconvert (convert among formats, data types, or modify metadata), idiff (compare images), igrep (search images for matching metadata). Because these tools are based on ImageInput/ImageOutput, they work with any image formats for which ImageIO plugins are available.

- A really nice image viewer, iv, also based on OpenImageIO classes (and so will work with any formats for which plugins are available).

- Supported on Linux, OS X, and Windows.

- All available under the BSD license, so you may modify it and use it in both open source or proprietary apps. []

This is a project that I am particularly interested in because this could open the door for more image formats available to use with Blender and Aqsis, the later in particular is already discussing adding this sometime in the future though it is on the backburner until the new core of the renderer is completed. OIIIO is a project that can be useful during each and every single step of a pipeline and that is a hard feat to accomplish, one of which this could potentially do very soon. Bridging the gaps between file formats is often a headache and reducing these steps only leads to increased efficiency.

---

Live Action and CG Visual Effects.

Strangely the open source world is working kind of backwards in the timeline of achievements compared to the professional visual effects industry, in that we already know that animated films are possible with these tools in this pipeline but it is still early for the visual effects realm and generally not tested yet. This is in stark contrast to the professional industry where since the early 1980's any CG in films was strictly for visual effects and was still years away from a full length CG animated film. Blender has recently been accredited with that feat with Plumferos.

This is a frontier that sadly is lacking in key pieces of technology, which is motion tracking. While there is the freeware Voodoo, it is kind of buggy and has not been updated in several years. There are other cheap software too such as PFHoe and PFHoe Pro which are not as feature rich as their flagship product PFTrack. However the one thing that Blender has going for it in this field is the Node Compositor, this has allowed people to really understand the reasons for it, not to mention truly take advantage of the tool and use it for visual effects. The only thing that has not been tested in this area is Blender with Aqsis, the very thing that RiSpec was designed for so it remains to be seen when and if this will happen, be it a small short film or a small studio with a vision, talent and funding.
Conclusion

Blender to Renderman development has changed and improved so much over the past few years it is hard to believe we have gone this far this fast. Years ago it was a chore to render a preview of something made in Blender and usually required a lot of hand coding, not very intuitive. Now it feels like we are working with technology and methods that the professionals have for many years and in some cases right there level with them, this is truly an exciting time as many of us have been waiting for this kind of workflow for years, it is here right now and will only improve. The only thing that really needs to change is more adaption into professional environments, even if the Hollywood studios won't do so, smaller places can easily work with this, people can learn, more development and collaboration between developers of all areas. We all want the same thing, to produce spectacular imagery and most of us have been inspired by the big screen effects since Renderman made dinosaurs come to life.

Another thing to do is to try to keep open communication between developers, the Blender to Renderman site was designed for that very purpose. The intent was to bring together the key people involved in these areas and this is not just limited to Blender and Aqsis, it also encompasses other software that have been used in the Widow pipeline. There is no competition to compete in, the idea is collaborate and build a better connection between these software types, from models, to shaders, image types and other data that is used in making films and visual effect shots. The shocking part is that studios are contributing as well with their own code and libraries, so there is support on both ends, there is a way to do this and it just takes some really devoted and talented programmers to make this happen. Having the support of the Blender developers is great step in that direction, as is having a deep connection to Aqsis, there is more room for diplomacy in other words and that is the aim of this paper. To encourage collaboration that will make people in the visual effects world go “Wow”.

As mentioned before the open source community is not the only one using open source tools. Industrial Light and Magic for instance uses Python for the majority of it's entire pipeline. At the time when Python entered the ILM pipeline it was intended to replace UNIX shell scripts, however as they worked with Python they realized that it could do a lot more than just processing of files. Over the past 15 years Python has grown to unify the ILM pipeline, it has grown far beyond it's original purpose. Python is now used not only to manage CG assets and elements, but also in daily shot review, as a network-based whiteboard, as an instant messenger, and even allows an occasional game of chess [4]. Tools are tools, it does not matter what realm of development they arrive from and as the name implies it is meant to get a job done.

This is not a battle over want software is better, or why this does that, or even pride. Blender is an awesome piece of code, we just want the choice to use another renderer for our work and we like Renderman. Thus the idea was born 5 years ago and we have only just begun. Hopefully more and more people will look at this with a new light and understand that we all want the same result we just choose an alternate path to get there.
References


   Magnus Wrenninge1