Renderman Point Based Effects Lighting Approach

Linghao Li

Submitted in Partial Fulfillment of the Requirements
For the Degree of Master of Fine Arts in Visual Effects
at
The Savannah College of Art and Design

© August, 2012 Linghao Li

The author hereby grants SCAD permission to reproduce and to distribute publicly paper and
electronic thesis copies of document in whole or in part in any medium now known or hereafter
created.

Signature of Author and Date___________________________________________

_____________________________________________________________________/___/________
Malcolm Kesson (Sign here) (Date here)
Committee Chair

_____________________________________________________________________/___/___
Bridget Gaynor (Sign here) (Date here)
Editor

_____________________________________________________________________/___/________
Robert Luo (Sign here) (Date here)
Topic Consultant
Renderman Point Based Effects Lighting Approach

A Thesis Submitted to the Faculty of the Visual Effects Department in Partial Fulfillment of the Requirements for the Degree of Master of Fine Arts in Visual Effects
Savannah College of Art and Design

By
Linghao Li

Savannah, Georgia
August, 2012
ACKNOWLEDGEMENTS

I would like to thank the members of my Committee, chair Malcolm Kesson, committee member Bridget Gaynor and Robert Luo. I could not finish this thesis without their patience and their willingness to help and encourage. I am really appreciating their help not only because they are the committee members, but also I've learned so much from them while I was studying at Savannah College of Art and Design. Though I've been working in the Visual Effects industry, I am still benefiting from some what I've learned from them.

Also I would like to thanks all my friends, who are too many to be listed, for giving me warm supports and endless encouragements. I've learned some much just from working projects right beside of them.

Finally I would like to thank all my family, especially my mother Juanyi Liu, father Dongsheng Li, and my lovely wife Wen Gao. Without their understanding and selfless support I could not even start pursuing my dream.
# TABLE OF CONTENTS

- List of Figures: 1
- Abstract: 2
- Quotes: 3
- Introduction: 3
- Background / History: 5
- Common Problems/Solutions: 8
- Industry Approaches: 13
- A Possible Solution: 17
- Tests and Results: 20
- Conclusion & Future work: 29
- Bibliography: 32
- Appendix: 34
LIST OF FIGURES

Figure 0: Wrong percentage display from original ptc display plug-in..........................13
Figure 1: Snapshot of an exploding trunk in feature film "The A-Team"..........................14
Figure 2: Not accurate point cloud file display.............................................................45
Figure 3: Correct and accurate point cloud file display..................................................45
Figure 4: Point extraction user interface........................................................................46
Figure 5: Extracted points.............................................................................................47
Figure 6: Fire simulation in Houdini..............................................................................47
Figure 7: VOPSOP shader network in Houdini.............................................................48
Figure 8: Houdini Render setup to export point cloud files .........................................49
Figure 9: Points display in Houdini and Maya...............................................................50
Figure 10: PRman light shader setup and render result................................................50
Figure 11: Scene geometries and camera in Maya........................................................51
Figure 12: Scene geometries and camera exported to Houdini......................................51
Figure 13: Colorized particle based fluid in Houdini....................................................52
Figure 14: Exported point cloud file displayed in Maya ..............................................52
Figure 14-2: Custom light shader user interface............................................................53
Figure 15: Interactive lighting pass render based on point cloud file............................53
Figure 16: Interactive lighting pass for another shot.....................................................54
Figure 17: Final compositing of a crash site shot..........................................................54
Renderman Point Based Effects Lighting Approach

Linghao Li

August, 2012

This thesis focuses on providing a possible solution for an efficient and photorealistic effects interactive lighting method. The primary focus is to implement a tool set that bridges elements between effects animation and lighting. The aim of the thesis is to study existing problems in interactive lighting from effects animation, compare the common and industry solutions, and trying to provide a possible approach to solve the problems.
Quotes

"You will likely not see it, but you will feel it"
- Eric Barba

Introduction

Visual Effects has become increasingly and inevitably important to the movie industry, "especially considering that 44 of the top 50 films of all time are visual effects driven" [Visual Effect Society.2011]. Production budgets for visual effects increase every year. However, the consequent requirements of tighter production schedules and better quality are often a burden to the visual effects production companies and artists. As a result, the pursuits and emphasis on efficiency and realism have never stopped.

There are numerous technical directors working exclusively to enhance the efficiency and capabilities of the pipelines for digital lighting and digital effects. However, in many visual effects studios, lighting and effects teams use different software packages. Therefore the integration between effects and lighting packages is still a big challenge, specifically for the interactive lighting from effects to lighting. Complicated and sophisticated custom scripts and tools have been written to bridge effects and lighting, but the ideal solution is still unknown. The technical directors and software engineers are constantly looking for an efficient and applicable method to integrate effects and lighting workflows.

One possible approach is through the implementation of two techniques for better display and use of Renderman point cloud (ptc) as a source of the interactive lighting for effects. Based on the existing Renderman point could Application Programming Interface (API), and
Autodesk Maya API, the tools and methods presented are intended to achieve

- Fast and easy effects element representation,
- User-friendly Renderman point cloud manipulation, and
- Efficient final renders of effects interactive lighting.

The goal of this solution is to craft a balance between overwhelming data complexity, and over-sacrificing the final result of realism within existing software packages, which can be added to an existing production pipeline relatively easier rather than creating a whole standalone system instead. By better supporting lighting and effects pipelines in the visual effects production, the combination of Renderman point based techniques and Maya API provide enhanced efficiency and realism.

In contrast to previous work, which served as a starting point and reference, this approach focuses on correct and accurate Renderman point cloud display, and it provides a point cloud extraction tool set inside of Maya GUI to interactively extract and save new Renderman point cloud file to disk based on current methods of displaying point clouds. A new display algorithm is implemented through point selection algorithms. Furthermore, step by step testing and proof of concept in a well-designed production project, focuses on simplifying the workflow of data generation from Houdini, and achieving efficient and realistic interactive lighting with Renderman point cloud API and Maya API. The proposed workflow has been successfully applied to the short film *En Route*, and the shots using this technique are serving as part of this thesis project.
Background / History

Intro to Digital Lighting and Effects

While Visual Effects contain different roles such as character animation and compositing, here it focuses on topics mainly related to two other equally important elements of visual effects: Lighting and Effects Animation.

Lighting, refers to here as "computer graphics lighting", is the "simulation of light in computer graphics".

Lighting in visual effects is literally creating virtual film lighting inside a simulated world instead of in the real world. There are various types of computer graphic research that focuses on simulation of real world light. Each research implements a simulation method in its own respect. However, the key factors in all the light simulations are mostly the same: Representation of real world lights are usually virtual light sources inside 3D software packages such as Maya.

A light (computer graphics light from now, if not specifically mentioned) commonly defined with characteristic of a real light source such as color, position, and size. In addition, light also has attributes that are defined methodologies of how the light rays are being calculated to emulate real world lighting, which includes diffuse contribution, specular contribution, reflection contribution, and "falloff" types.

Different types of lights have been successfully implemented. A user can find a spot light, point light, area light, ambient light, IES light and HDR dome light in most of the 3D lighting packages. To some extent, each type of light successfully emulates one type of light in real
world. The first types of light emission to be used in computer graphic were comparably simple. A point light mimics the behavior of a light that emits energy in all directions such as a light bulb. A directional light sends out light rays in one arbitrary direction (with parallel rays), and can be used to imitate sunlight. A spotlight emits light rays from a point to a certain direction within a defined cone angle to simulate a spot light in the real world. An ambient light does not emit light rays, but instead it unconditionally illuminates a computer graphics scene without casting any shadows - sometimes used to emulate environmental fill lights. Relatively complicated light types have also been implemented. The light rays directions, and their decay rates, from an IES (Illuminating Engineering Society) light source are read from a pre-defined text file. The "IES standard file format was created for the electronic transfer of photometric data over the web"[cgarena.com.2012]. It is a great way to reproduce an accurate representation of architectural lights in a virtual world. An HDR (High Dynamic Range) Image based dome light samples colors from a texture map, and emits light ray from its dome shape. It provides an efficient way to recreate the illumination of an environment that surrounds an object.

Effects, here referred to as computer graphics generated Effects Animation, could be categorized as one part of computer animation. It is the animation of any non-character elements (fire, smoke, explosions, debris, clouds, water, rain, cloth, etc), achieved by effects artists to meet all technical and aesthetic standards for a project. In technical terms, Effects animation is classified into particle effects, fluid effects, rigid or soft body effects, and any additional applications of research that has been done to simulate natural phenomena.

However, regardless of how the above techniques have been implemented in 3D software
packages such as Maya or Houdini, it does not change the fact that effects animation is "organic" by its nature. For example, in order to create a realistic computer generated fire, which is commonly seen in a large amount of movies, effects artist need to control the color, direction, and movements of a fire by setting up corresponding attributes with either particle or fluid tools. There are incredibly large amount of controls to craft a believable or even super realistic looking fire. Although a fire's primary motion and how vivid a fire can be determined beforehand, the exact shape and tiny details of the simulated result is unpredictable to some extent because of how effects simulations are mathematically generated. As a result, the complicated and unpredictable shape, and yet constantly changing color of the fire raises challenge to reproduce interactive lighting from the fire effect to the surrounding objects.
Common Problems/Solutions

Common problems

Although more sophisticated types of lights could be introduced, it does not change the fact that they are either defined from a point-to-point basis, or designed under a premise that lights are static. In other words, the lights stated above cannot simulate a real world light source that has volume and, even worse, has a constantly changing volume such as burning fire. As a result, interactive lighting cast from effects such as burning fire or explosions has always been a setback for visual effects production.

Lighting and effects are considered different stages of the visual effects pipeline, so there are inevitably different technical disciplines applied to each side. In addition, in most cases, lighting and effects artists are using different software packages such as Maya and Houdini. To create a fire effect that illuminates surrounding objects a great deal of guess work must be made from the lighting side of production. Lighting artists generally need to decide where to create and position a light to match the fire's location, and determine what to be set to the light's color attribute to mimic the fire's colorations. Since these setups are uncertain, several iterations of repeated efforts such as testing and comparisons are required to achieve final satisfactory result. Consequently, more resources including render utilities and manpower will be wasted during the process.
**Common solutions**

Temporary solutions are employed to resolve the issue stated above. Based on the technical approaches, most of them can be put into two categories.

*Image based approximation*

Image based approximation is one possible solution. Effects elements could be rendered out as final images first, and then projected into area lights or HDR lights. Take fire effects as an example. The advantage of image based approximation is that the attributes and characteristic of the desired fire are built-in and finalized to the rendered images. Though lighting artists still have to position the area lights or HDR lights, the actual color and movements of the fire is correct from the camera angle. Effectively, this approach will produce correct interactive lighting on the surrounding objects in terms of color and timing of the fire animation. This solution provides better interaction of effects and lighting mostly through precise key-framing attributes such as color and motion. However, several problems are introduced as well.

Firstly, more disc space is required. Because the fire elements are constantly in motion from one frame to another, each frame of the fire animation will need to be saved out as a different texture, and then projected onto area lights or HDR lights. Despite the extra effort involved for generating a sequence of texture files, the disc space each texture file could end up using is fairly large. Depending on the scale of the visual effects production, one texture file would take up about 50 to 600 mega bytes of disc space. What makes the problem even worse is that texture projection is only legitimate on shot-by-shot basis because the camera
for each shot is usually different. In other words, most of the projection texture files generated during the process of utilizing this solution cannot be applied from one shot to another. As a result, each shot using this approach will ask for about 2 - 30 Giga bytes of disc space, and then the total amount of disc space for all the shots can be huge.

Secondly, more RAM is required. The size of the texture files also affects the rendering process. Every texture in the scene being used, including the fire projection textures, needs to be loaded into memory during the render process. One more extra-large texture means one heavier load of memory and less efficient of the renderer, therefore, resulting in longer render times.

Thirdly, the projected fire texture files may not be sufficiently accurate. Even though acquiring additional disc space and RAM has become increasingly easier and cheaper, this solution hasn't resolved the accuracy issue. The projected textures are a render result of the fire that faces forward toward the shot camera, so the opposite side colors of the fire which the render camera cannot calculate are uncertain.

Furthermore, compared to a vivid and organic fire shape, the fire projection textures are projected onto planar or spherically shaped lights because the common light shapes are limited to planes and spheres.

Moreover, the lights themselves are static, which are not proper representation of how the fire would be formed throughout the animation. In other words, though the color, intensity and motion of the fire emulated by this approach is relatively correct, the exact shape of the fire in 3D space is inaccurate, which will cause render artifacts if the shot camera has large amount of movements.
Overall, this image based approach is not an accurate and universal solution for a large amount of shots in production.

*Light formation approximation*

Light formation approximation could be another solution. In this case a definite number of lights are generated to approximate the actual shape of the burning fire. Sample points are taken on the fire shape to capture color, intensity, and position information. For each position of the sample point, a point or sphere light will be created with exact color and intensity sampled. Accuracy of the approximation depends on the number of the lights a user has decided to create. The larger the number of lights to form the fire shape, the more accurate representation of the fire will be achieved. With this setup, the representation of the fire is relatively accurate in 3D space, as long as the number of lights is high enough.

Unfortunately, this approach has downsides as well. First of all, such setups require technical director supports. A burning fire's shape, color, and intensity are constantly changing. Accordingly, the distribution of point lights need to be adjusted, and each light's attributes such as color and intensity have to be re-assigned. Lighting in 3D software, such as Maya, does not have a built-in capability to create the above complex setup. Though light formation can be accomplished, it calls for considerable amount of a technical director's assistance such as writing and maintaining custom programmable scripts to sample the cached effects file saved in disc, applying gathered information such as position, color, intensity to each created light on each frame, and creating user-friendly interfaces inside of Maya to provide better user control of this setup.
Secondly, depending on how accurate a lighting artist wants to get, there might be as many as 10 to 3000 point/sphere lights being generated for each frame. Unfortunately, more lights, especially more point lights or sphere lights, added to the scene means there will be exponentially increased final render time. The render time is a key factor that drives decision making in a visual effects production, which makes this approach unacceptable for most scenarios. To be fair, finishing and supporting this setup is not even a challenge compared to what Technical Directors are capable of achieving. However, considering the second drawback, the light formation setup would only be applicable for some one-off shots during the production.
Industry Approaches

Visual Effects studios are certainly aware of effects to lighting integration and the interactive lighting problem, and there are several different approaches to try and solve it:

*Double Negative point cloud display tool*

The Maya plug-in for point cloud display served as the starting point of a current possible solution. As it will be shown in the test section, this plug-in meets most of the Photorealistic Renderman Point Cloud file displaying requirements if the user does not request unusual “clipped” displays. However, it does not have a correct percentage display algorithm (as shown in Fig.0), and cannot perform point extraction.

Fig.0. Wrong percentage display from original ptc display plug-in.

In Fig.0, the result of 60% display from the original plug-in is the very first 60%, which is not proportional and thus a less dense version of the original torus. As a result, it is wrong.
Digital Domain ISO surface interactive illumination

The ISO surface interactive illumination approach is applied by Digital Domain to the recent feature film project "The A-Team".

Fig. 1. the screen snap shot of an exploding trunk in feature film "The A-Team"

To illuminate an exploding trunk (as shown in Fig.1), the fire from the explosion needed to cast a considerable amount of illumination onto the truck, the tunnel, and other surrounding objects. The ISO surfaces are exported from Houdini for each frame based on the shape of the fire in order to emulate the changing fire shape. Although the ISO surfaces are small, they basically try to form an organic 3D fire shape by using a large number of such surfaces. The rendered results of fire from a Houdini simulation are composited and projected onto the ISO surfaces as an incandescent texture in order to cast light.

Although such approaches can successfully satisfy a visual effects supervisor's requests, it unfortunately has some drawbacks.

Firstly, the exported ISO surfaces sequence requires large data input and output. An individual ISO surface is geometrically small, so this setup needs as many such surfaces as possible to approximate the desired fire shape to a satisfactory point. Similar to most of the
baked/cached out geometry from 3D software packages, the ISO surfaces files are large when
the exported resolution is set to high in order to mimic the exact shape and volume of a fire.
Consequently, large format files impose large memory requirements on the nodes of a render
farm. It inevitably results in longer render times and increases the possibility of render
crashes from a lack of memory.

Secondly, no matter how geometrically small an ISO surface is, it still has a minimum
size. There is really no way to represent the shapes of the tips of a fire with an arbitrary sized
ISO surface. As a result, the emulated fire shape is still not entirely accurate.

Thirdly, similar to Image based approximation stated before, even though the fire shape
can be mostly mimicked by increasing ISO surface density, the color of the fire is inaccurate
at the reverse side of the fire texture projection because the projected color is camera
dependent.

Although ISO surfaces have the same shape in 3D space, so they would be reusable for
multiple shots, the camera dependent nature of color texture projection limits the universal
application of this technique.

\textit{Bgeo to VRay Mesh Interactive Illumination}

Bgeo (binary geometry file) to VRay Mesh interactive illumination approach can be
considered as an evolved version of the ISO surface interactive illumination. Bgeo file
sequence of effects element can be exported from Houdini with an extra attribute type called
"detailed attribute". The detailed attribute carries fire color information for each surface point,
and it will be stored along with the bgeo files. The bgeos file is converted to a VRay Proxy
Mesh which is subsequently rendered with VRay. A VRay shading node can read the transferred color information inside the converted VRay mesh file as vertex color. Such Houdini to Maya vertex color transfer saves the necessity of projecting rendered images of fire elements onto the surface, and it also provides correct color information in 3D space, especially at the reverse side of camera. Finally, after loading the VRay mesh into Maya, a shading node called "VRayVertexColor" is used in a VRay light material to emit light.

Though this approach still uses geometry to approximate the fire shape, it is now uses the small and efficient VRay mesh format instead of the large and slow bgeo format. This saves both disc space, memory, and thus it saves render time. Also the colors are more accurate in 3D space because the data is transferred with detailed attributes directly from Houdini and can be accessed inside of Maya.

Currently the light material supports only diffuse lighting from GI (Global Illumination) in VRay. As long as specular lighting from the fire is not required, the benefits of using this solution to create effects interactive lighting are considerable.

Compared to the ISO surface interactive illumination, and assuming the effect geometries are shared across the board, this can be a relatively general solution for a number of similar shots. This approach seems promising and is still being evaluated at Digital Domain. This solution will become even more desirable when the VRay lighting materials begins to support both diffuse and specular lighting.
A Possible Solution

Artist Statement

I am a technical director and a visual effects artist. Utilizing my technical skills on my artistic work is what I enjoy the most. The implementation of the proposed solution evolved as a result of working on a Savannah College or Art and Design (SCAD) short film project named “EN ROUTE”. Several (almost full) CG shots were designed in the short film to virtually depict an airplane crash site. Other than the regular modeling, texturing, lighting, and effects tasks, the crash site envisioned by the director (Colin Levy) showed several fires burning around the debris of the crashed airplan. A fair amount of interactive lighting from the effects elements is required. As Mr. Eric Barba (Visual Effects Supervisor at Digital Domain) once mentioned, the subtle detail of the lighting effect, will not easily be seen by the audience, but it will certainly add to the appropriate feel of fire interaction. Using the tool sets and workflows implemented in this possible solution is a cherished learning experience and helped both myself and the EN RROUTE short film grow to a new level.

Approach

Necessary algorithms have been studied and a few technical difficulties have been overcome, which are documented in the Appendix section. The tool sets and methodologies implemented are two parts based on the usage of Photorealistic Renderman (PRman) point cloud (ptc) file.

The first part is a custom Maya plug-in to display and extract ptc files. "A point cloud is
a set of vertices in a 3D coordinate system. These vertices are usually defined by X, Y, and Z coordinates. The ptc is a special type of point cloud can store extra data such as color, area, and normals at each point. It is a powerful but efficient file format that can store a great amount of information in 3D space. Through the use of the PRman renderer, ptc files can be generated in many software packages including Houdini and Maya. Additionally, ptc files can be created not only from a surface being rendered, but also from any render-able object including effects element such as particle and fluid. With this handy feature, ptc files are being used to store effects elements information in many visual effects production companies.

The generated ptc files can be previewed by a standalone command line tool that ships with the RenderMan Pro-server package called "ptviewer". However, the built-in preview and display functionalities of ptviewer are limited. It is a standalone command line tool without much support for user interaction. Additionally, as most lighting setups are completed inside of Maya, the support of displaying ptc in Maya would be ideal. Fortunately, the PRman programmable API called "Point Cloud API" and Maya API provide possibilities of implementing a custom Maya plug-in that can display and even operate on ptc files inside of Maya. This implementation starts with an open source ptc viewing Maya plug-in from Double Negative. Then it improves a percentage display issue of the original plug-in with studies of probabilistic algorithms. Furthermore, current implementation has evolved to add a feature to extract points from either a certain percentage, or defined area of original ptc file into a new ptc file.

The second part is a Houdini to Maya workflow includes generating a ptc file in Houdini and then using custom RenderMan light shader that uses the ptc file as a light source in Maya.
The thesis project, which evolved with the ptc file plug-in mentioned above, requires data inter-change between Houdini and Maya and a fair amount of effects interactive lighting from burning fire elements to the surrounding objects. The workflow as a result is designed to bake out fluid simulation information in Houdini, reuse the information to drive particle simulation, bake out particle simulation as ptc files with a custom Houdini vopsop shader, reuse baked ptc files as light source in a custom light shader, and also add multiple technical workarounds such as Maya to Houdini geometries and camera inter-changing, automated disc space managing, and writing programmable scripts to hack the SCAD render farm limitations.
Tests and Results

Point Cloud Display

Rather than using a complicated ptc file containing points that form a fire shape, a ptc file baked from a torus inside of Maya was chosen for the testing purpose. The torus shaped ptc file has been displayed and compared with both the original plug-in from Double Negative and the implemented version here. In Fig.0, because the original plug-in is using the fixed point selection algorithm, the number of points to be skipped at each step will always be a integer return from the "floor()" function, and this integer will be totally inaccurate when the desired percentage is a value that cannot be divided by 1 (60% for example). As a result, the 60% percentage will get a wrong display result similar to Fig.0.

Initially, in order to fix the percentage display problem, the numerical probabilistic algorithm was used in hopes of producing a better proportional display result. As shown in Fig.2, the display result using this numerical probabilistic algorithm has a much better proportional selection of points in general. However, there are two issues that cannot be ignored:

First, the result to be displayed is unpredictable. Even with a same desired percentage, the display result is constantly changing in between frames. For each frame, the plug-in node's "compute()" function re-computes which points are supposed to be chosen and displays the newly selected points again. The random number generator produces a different number every time, and for each point, it only compares the random number once with the desired percentage. That means the plug-in will take the sample for only once per point. As a
result, the point selection at one frame is normally very different than the new selection at another frame. Thus the display result is inevitably unpredictable.

Second, accuracy has been lost in computation. As shown in Fig.2, the total points are 20,902, so the desired 60% point selection is supposedly 12,594. However, the actual number of points loaded is 11,588. Roughly 7.6% of the total points are lost in computation because the number of samples taken is too small.

Additionally, Monte-Carlo and Las Vegas Algorithms were tested because they require as many samples as possible at each point, and it could theoretically reduce the accuracy problem. Given the number of samples, say 20 means that 20 random numbers are generated per point, the mean value of these random numbers are then compared to the desired percentage. The display result was much more predictable, and the accuracy lost in computation was largely reduced. However, because the number of samples increased, the computation time consequently also exponentially increased. A test shows that using 20 samples per point rather than 1 sample, the overall computation time increased by about 500%.

Finally, varied interval point selection was the algorithm chosen to implement the ptc files display problem. In Fig.3, 18.630% and a 60% has been displayed proportionally. The accuracy lost is controlled at under 0.012% for both percentages. The plug-in node's computation time is fast and the point selections throughout the frames are identical and completely predictable.

The ptc display test results demonstrate that the point cloud display plug-in function sets provide a better solution of bridging effects to lighting. The ptc file display helps lighting
artists and even animators gain a better sense of effect element in the scene.

**Points Extraction**

Other than displaying the input ptc file, the implemented plug-in provides function sets to extract a user desired portion of points from the input and save them as a new ptc file.

The user interface has been updated for supporting the points-extraction feature in the plug-in. As shown on Fig.4, the output point cloud path, bounding box object text field, the extractor bounding box minimum and maximum channels and the write out point cloud button are added on top of the original plug-in user interface. With these updates, the user interface has become more user-friendly when using the ptc extraction feature.

For the actual points-extraction, the fundamental idea is the to-be-extracted points are the points that are currently loaded and displayed. A "loadptc()" function in the source code is designated to display the ptc files correctly. Based on that, given a user defined output path, instead of storing all the required data to the C++ boost arrays, a function called "writeptc()" was implemented to write out any currently loaded/displayed points as a new ptc file. The original plug-in provides two display options - the percentage of the points to display and the colored points to display. Aside from the percentage of points to display, the plug-in can display points that meet certain color criteria. For example, given a red color(0.5,0,0), and conditions such as "larger", "smaller", and "equal to", the plug-in will display only the points having red color larger, smaller, or equals to the value 0.5.

In addition to those two options, a bounding box extraction feature has been added to the plug-in. The points will only be selected and extracted if they are inside the given bounding
box of an object inside of Maya.

Furthermore, extra safety checks were added. The safety checks provide options to enable or disable bounding box mode and also prevent the input file get overwritten when user accidently set the output ptc file path to the same as the input. As shown in Fig.4, the torus shaped point cloud is being extracted with the blue "pCube1". And in Fig.5, the result ptc file is displayed in Maya.

The colored points display option is inherited from the original plug-in, and the percentage display option has been improved. Additionally, the bounding box points-extraction adds further ptc files operation possibilities.

As a result, the points-extraction function has the ability to extract points based on any combination of the above three options:

- Percentage and color extraction
- Percentage and bounding box extraction
- Color and bounding box extraction
- Percentage, color, and bounding box extraction

The point extraction feature gives artists ability to create customized ptc files directly inside of Maya. Inspired by this feature, the ability to extract important or interesting points empowers artistic freedom, and it opens a door to many possible new ptc file generation and re-use approaches.


**Point cloud indirect illumination**

In Houdini, the fluid driven particle approach was chosen to simulate a test burning fire. As shown in Fig.6, a fluid simulation runs first, and then it bakes out the fluid information including velocity, fuel, and temperature as separate fluid cache files. The baked fluid velocity has been used as a driving force of a particle system. The left, middle, and right images in Fig.6 demonstrate the fluid simulation, baked velocity information, and velocity driven particle system respectively.

By default, the particle Cs (color of surface), which in Houdini equivalent as Cd, is a white color. As shown in Fig.7, the network shaped node graph shows that with the baked fluid velocity, fuel, and temperature information, Houdini can reuse the information and set the Cd based on user defined rules in a VOPPOP shader. In this case particularly, the VOPPOP shader uses baked fluid information as input to drive the behavior of the particle. As a result, the particle tends to have a warmer color and bigger acceleration value at the point where the fluid has higher speed, more fuel, and higher temperature.

After the particle has been colorized, it needs to be baked out as ptc files in order to firstly being displayed by the implemented Maya plug-in, and secondly being utilized as the source of interactive lighting. A custom PRman surface shader utilizing a "bake3d()" function is used to generate ptc files with correct information. In addition to the regular render quality settings, where the density of the point cloud could be set to high or low by changing the render qualities, Fig.8 shows the PRman renderer AOV (arbitrary output variable) setup to bake out ptc files with necessary data including "Cs", "_radiocity", and "_area".

The implemented point cloud display Maya plug-in were used to display the baked fire
As the Fig.9 shown, screen capture of effects particle in Houdini is to the left part of the image, and the baked point cloud displayed in Maya is to the right. Given the fact that the ptc file contains fairly low resolution data because of it is baked out mainly for the testing purpose, it still successfully represents the effect element shape in terms of color, position, and volume in 3D space. It surely gives the Maya users especially lighting artists a much better idea of what and how exactly the effect element look like inside of Maya.

The crash site burning fire element is relatively easier to setup with, but the bigger challenge comes from setting up the interactive lighting. There are 4 crash site shots in the short film project, and 3 of them require computer generated burning fire effects and consequently their interactive lighting on surrounding geometries including broken engine and debris.

Common methods such as image based approximation and light formation approximation were tested. However, both of the approaches are time consuming in terms of project setup and final image rendering, which all have been verified having same problems covered on the common solutions section.

Fortunately, the RSL (Renderman shading language) function called "indirectDiffuse()" can take a ptc file as a input and reuse that as a light source.

Similar to using GI, the "indirectDiffuse()" function has been commonly used in the surface shader to calculate indirect lighting casted from one surface onto another surface. When put this function into an illuminate calculation loop of the PRman light shader, it can be treated as the major light source and thus cast out diffuse and specular lighting.

In Fig.10, the top part of the image shows a maya spot light with the custom PRman light
shader attached. The light shader uses the ptc displayed in Fig.9 as the input, and then the button part of the image shows the render result from this light. The interactive lighting usually gets rendered as a separate render pass, and it will then be composited on top of the master beauty render.

After the above test process, all the tested techniques and approaches have been applied to the crash site shots:

Fig.11 shows the crash site geometries setup inside of Maya.

Fig.12 shows the geometries transferred to Houdini with all the geometries in the scene converted as an obj file, and then baked out as bgeo file in Houdini. It also shows the camera that is using the exported ".chan" file. Moreover, it shows the fluid simulation and required node setup to bake out fluid information (including velocity, fuel, and temperature) to disc.

Fig.13 shows the colorized particle based on the fluid's velocity and fuel information. The velocity and fuel data are saved out and read in individually to achieve more flexible control. The customized PRman surface shader is assigned to the colorized particle, and then a sequence of ptc files has been baked out.

Fig.14 shows the generated the ptc file displayed inside of Maya and the Maya light which uses the custom renderman light shader. On top of reading the ptc file as a input, the light shader also applies an user definable overall color tint, providing the ability to override the light color if needed.

Moreover, there are several other attributes added to the light shader to have better control of the indirect lighting as shown in Fig.14-2: The "intensity" attribute works as a
overall light intensity multiplier; the "clamp" and "sortbleeding" make the render result smoother; and the higher the "samples" number is, the less artifact the final render result will get.

Fig.15 shows the fire interactive lighting pass rendered based on the ptc file while using the above light shader. The total size of the ptc files for a shot is relatively smaller than using other approaches, but it still exceeded the 1 Giga bytes render farm limit. Thus a pre-render and a post-render script have been used to copy and clean up needed ptc files when accessing the render farm.

Fig.16 shows the fire interactive lighting pass has been rendered for another crash site shot with same setup. Since the effects fire simulation is the same, the resulting ptc files are the same. One big advantage of this point based interactive lighting work flow, is that the baked ptc files are multi-functional. In other words, what can be previewed in Maya viewport can also be used as the source of interactive lighting. Additionally, because the ptc information are stored in 3D, the render camera could be positioned freely without requiring a new effects element representation. No extra ptc files needed to be baked, and also the ptc files baked from a previous shot could be shared and reused for the interactive lighting on another shot.

Fig.17 shows the final composited image of a crash site shot that uses the effects interactive lighting from Fig.16. The final render resolution is 2048X1024, and the average render time per frame for interactive lighting is less than 5 minutes. The render time is relatively nothing compared to the traditional light formation approximation approaches which take over 2 hours per frame to produce a fairly inaccurate result. Given the fact that all
the 4 crash site shots are over 500 frames in total, the point based interactive lighting approach saved at least 1000 hours of render time. In addition, the rendered frames have no noisy render artifacts which could normally be found in the renders from the common approaches.

Overall, the point based indirect illumination approach provides accurate and efficient effect interactive lighting by achieving correct shape in 3D space, matched color variation at each point position, reduced render artifacts, remarkably faster render time, and completely shared and reusable resource files.
Conclusion & Future work

The primary goal of future work will be to increase the display tool's displaying flexibilities, to add complex ptc file operation abilities, and to improve point based interactive lighting capabilities.

The ability to display Renderman brick map file could be added to the display function sets of the Maya plug-in for further development. As another built-in type of file format from PRman, the brick map files can either be converted from ptc, or baked out as render AOV directly. Brick maps not only stores 3D data similar to the ptc files, but also optimizes its data structure with "octree" in order to represent stored 3D data information based on level of detail. That said, by giving a certain level of detail, user can decide how precisely a brick map can be displayed or reused. And with such premise, the Maya plug-in will supposedly read only the necessary brick map information given by a predefined detail level, and then display that information inside of Maya viewport.

Additional development could be expected to support operations on more than one ptc files. To provide further capability of complex operation rather than only extract point data from one file, Boolean operations such as union, difference, and intersection on two or more ptc files can be added. Given two ptc files for example, user would ask for the creation of a new ptc file which has all the points from having red color and positioned in the common area of original two ptc files have taken in 3D space. Theoretically, along with the existing percentage and color based extraction, this further development would provide even more exciting ptc file operation possibilities to artists.
In order to maximize usability, future implementation of point based lighting workflow might involve supporting volume render and better shadow calculation. Currently in this possible solution, the point based lighting only illuminates surface geometries. By adding the ability to also support rendering with the volume object such as smoke and fog, this technique will be even more preferable to be applied on large scale Visual Effects projects involving lots of fire and smoke interaction. Moreover, currently the sample points of shadow calculation from the points inside of a ptc file are fairly arbitrary. A better shadow computation algorithm which can proportionally sample the shadow points would be expected to cast softer and more realistic shadow.

To better support lighting artists, future work would positively involve a point cloud data transformation method to provide ability to post transform the point cloud data information inside of Maya's viewport. The ptc files store data information in 3D space statically. That said, position of a point inside one ptc file cannot be simply moved from one 3D location to another. In order to move these points in 3D space carrying all the original data, normally a new ptc file has to be created to represent the changes. The Prman light shader uses a ptc file as light source, so the actual lighting are computed from where the points are in 3D space from inside of the ptc file, ignoring the Maya light's own transformation. In other words, currently no matter where and how this Maya light is positioned, the render result from a same ptc file will be the same. Therefore, before rendering process, the transformation of the light should be taken into account, and calculated on top of where the ptc points originally at. Then the "indirectdiffuse()" function computes the post-transformed points as the new light sources, which theoretically would give freedom to the lighting artists to "move" a Maya
point based light inside of Maya without creating an extra ptc file.

The PRman point based effects lighting approach, as presented, represents a solution for visual effects artists who need to create effective and realistic effects interactive lighting. Combining the PRman point based technique and Maya API provides abilities to quickly and easily generate an accurate effects interactive lighting strategy that would normally be difficult or resource consuming to achieve using traditional or common techniques. The PRman point based effects lighting approach is controlled by a few key features while giving user customizability and access to advanced tools to fetch artistic freedom. The improvements focus on better display flexibility, ptc file operation capability, volume/shadow render ability, and 3D data transformation based on light position will be achieved in the future work, without preventing usability of current completed features. And when applicable, the final resulting workflow will hopefully be competitively appealing and thus gives visual effects artists and technical directors a preferable solution to choose from.
Bibliography


< http://www.cse.msu.edu/~cse872/tutorial4.html>

< http://www.randomnumbers.info/content/Random.htm>

< http://www.sidefx.com/docs/houdini10.0/nodes/pop/voppop>


< http://www.stanford.edu/~ctj/pc.html>

Sun, Chen. "Simulation of The Mixing of Ink and Water in Houdini". August. 2010

< http://www.visualeffectssociety.com/node/2425>

< http://www.wisegeek.com/what-is-3d-modeling.htm>

Appendix

Algorithm Studies

Probabilistic Algorithms

Fixed interval point selection

This algorithm is simple enough, and it is the one being used in the Double Negative open source ptc file display tool. Given a pool of points to select from, a fixed interval is applied to selected points from the pool. That said, every n-th point will be selected from the original pool. For example, if 50% percent of original points are requested, every 2nd point will be selected. Similarly, if 25% of original points need to be selected, every 4th point will be chosen. However, if the requested percentage is to 60%, which means the desired interval (1.66667 in this case) is not an integer, it will cause a fatal accuracy issue. A point cannot be split into a half or into a certain percentage such as 66.777%. Consequently, the desired interval has to be either "floored" (process by a math function "floor()") or "rounded" (processed by a math function "round()") into an integer.

In the Double Negative ptc file display tool, the "floor()" function is chosen to convert the interval value from a float number into an integer. When the percentage is set to 60%, the number of 1 will be returned as the interval. As a result, instead of a correctly and proportionally distributed 60% point selection, the very first 60% of all the points will be selected because the interval is 1.
Numerical probabilistic algorithm

Numerical probabilistic algorithm is based on number of samples. For each sample, one or multiple random generated numbers are generated. The larger the sample numbers, the more accurate the result will be.

A famous application is coins flipping experiment: the possibility of getting a coin's heads and tails are 0.5 if the number of coins flipped is large enough.

In the tests and results process, a pool of points and a desired percentage are given. For each point in the pool of points, a random number between 0 and 1 will be generated to compare with this percentage. If the random number is smaller than the percentage, the current point is selected. For example, if the given percentage is 60%, and if a random number generated is smaller than 0.6, the current point is chosen.

The applications of this algorithm are seemly easy and applicable, but they are all suffering from lose of accuracy. In the implementation process of this ptc file display tool, an accuracy test showed that a desired number of points is 12541, and the actual points number got selected is 11588. In terms of accuracy, the results are having an intolerable 7.6% mismatch.

Monte-Carlo and Las Vegas Algorithm/Methods

It is "A randomized algorithm that may produce incorrect results, but with bounded error probability." [Atallah.1998] The Monte Carlo algorithm requires large number and fairly distributed samples, and may return an answer that is not correct. Similar to Monte Carlo algorithm, Las Vegas algorithm needs large number of samples. However, it may not return
an answer at all, but if it does, the answer is guaranteed to be correct. The simple applications of these two algorithms have been tested on the ptc display plug-in. Given a target percentage and a pool of points, the random numbers between 0 to 1 have been generated as many times as possible for each point. Then compare the mean value of the sum of all generated random numbers to this desired percentage. If the mean value is smaller, choose the current point.

For each point, a random number generator has to be run many times. As a result, the more the total points are, the slower the overall calculation will be. Based on the test results, it can be concluded that the algorithm is not suitable for a Maya plug-in node which supposed to react/compute to a user input at almost real time.

Sherwood Algorithm

Sherwood Algorithm basically works as a quicker version of Las Vegas algorithm. However, it still involves fair amount of random number generations which will still be a time consuming process if the total point number is larger than 1 million. Given the fact that the result is inaccurate and still can't be completely expectable, this algorithm is not the ideal choice.

Varied Interval point selection

Varied Interval point selection is similar to the fixed interval point selection. To select a certain percentage of points from a pool of points, an interval is applied to selected points. However, instead of one constant interval, the interval being used is varied at each step when needed. In other words, every N-th point will be selected from the original pool, but the
number "N" is constantly changing when needed at each iteration step.

For example, if 60% of the points need to be selected, the desired interval will be 1 divided by 0.6, which is 1.6667. For each step, the interval is calculated differently:

Firstly, at the first step, a number 1 calculated from 1.6667 will be taken as the first interval, and a remainder of 0.6667 will be stored temporarily.

Secondly, at the second step, the desired interval plus the temporary reminder, 1.6667 plus 0.6667 in this case, is 2.3334. Then a number 2 calculated from 2.3334 is taken as the second interval, and the remainder is 0.3334 and stored temporarily.

Thirdly, at the third step, the sum is 2.0001 from 1.6667 adding 0.3334. Then the third interval is 2, and the remainder is 0.0001 correspondingly. Following the similar steps and the interval is constantly changing between 1 and 2.

This method requires no random number generation, so consequently it produces no efficiency issue. Also it provides accurate and predictable result for each desired percentage of point selection. A fast and correct approach of point selection could be achieved. As a result, this varied interval point selection algorithm is chosen to implement the ptc file display tool functions sets.
Technical Difficulties

Work Environment Setup.

The implementation of the ptc display/extract tool has been majorly worked on with a lab workstation at Montgomery Hall, Savannah College of Art and Design (SCAD), using Linux as the operating system. Initially, all the required software are either missing or at wrong versions. Additionally, all these software installation, update or change, and the C++ Maya plug-in compilation need super user permissions. Fortunately SCAD provided an admin user name and password to install extra software on lab workstations. To properly setup the working environment, there are several steps involved:

Step I. Install Cmake

Cmake 2.8 or up is required. Nothing special to mention, the software works as expected after installation.

Step II. Install Boost

Boost is a C++ library for efficiently utilization of some C++ data types. Technical director at many visual effects studios such as Digital Domain are using boost extensively to achieve possible software efficiency. Only fairly small amount of array function sets from boost will be used for the plug-in, so it doesn't need to build the library separately. The library could be copied to any allocation on the computer disc, and link the allocation to the plug-in when compiling. However, after some trials and errors, the easier way to utilize this library is to copy the source scripts into Maya's installation directory.

Step III. G++ version down grade
A compiling error similar to:

"undefined

symbol :ZSt16__ostream_insertIcSt11char_traits_ZSt16__ostream_insertIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_E
S6_PKS3_lcEERSI13basic_ostreamIT_T0_ES6_PKS3_l"

occurred at the first several times when trying to compile the Maya plug-in. It turns out the G++ version on the lab workstation was 4.2.4, but the G++ version the Maya usually compatible is 4.1.2 or older. After down grade the G++ version to 4.1.2, the original Maya plug-in from Double Negative finally compiles.

**C++ Programming Language**

Several programming languages such as C, Assembly Language, and Python have been experienced and applied to previous work before this possible solution. Based on how much fluently above programming languages have been practiced, C++ is not a primary choice if possible. However, the C++ programming language is used in the original point cloud Maya plug-in, and it’s also the required programming language for creating general Maya API tools including custom nodes, commands, and deformers. Though Python is the only other language Maya API supported, continuing working on the plug-in with C++ is chosen in order to take a quick stab on the actual function sets implementation. Fortunately, the Maya API and Point Cloud API have their own libraries of Functions to program a plug-in, which is more than enough to be utilized to add extra functions to the original source scripts.

Consequently, not too many C++ language specific programming rules and usages were required during the process of implementing the plug-in. As a result, the ptc display/extract Maya plug-in has been successfully implemented without taking too much time on learning
and mastering the C++ language itself.

**Maya to Houdini Pipeline Setup**

Inter-changing geometries and cameras from Maya to Houdini is needed because the lighting setups are generally operated inside of Maya. Houdini needs the same geometries to render with effects element, and it also needs to render with the exact same camera movements as what it is inside of Maya.

An "obj" is a Maya support file format that has description of how geometry should be constructed in 3D space, and Houdini can read obj files exported from Maya. After the obj exported from Maya and read in Houdini, the result geometry will be exported out from Houdini as bgeo file, which stands for binary geometry file. It's a more user-friendly format for Houdini and will be processed relatively faster inside of Houdini's graphical user interface.

The geometries transferring stated above is pretty straightforward and easy to operate, thus no additional customization is required. However, camera export from Maya to Houdini is not natively supported by off-the-shelf tools inside of both Maya and Houdini. Common camera attributes along with camera transform information over time, are stored with animation curve in Maya. Unfortunately Houdini cannot read the animation curve from Maya directly, and there is no built-in animation exporter to convert the animation curves to a Houdini supported format such as a "chan" file.

A customized Python script has been written to gather all the camera information in Maya, and then it writes them out as "chan" file. Correspondingly in Houdini, a customized
OTL (Operator type Library) reads the camera information from the exported "chan" file, and then apply the information to a native Houdini camera. Test renders from both original camera inside of Maya and new camera with "chan" file loaded inside of Houdini shows that this setup is correct and ready to be used in the thesis project.

**PRman Point Cloud API Functions Debugging**

In addition to implementing the display function sets in the ptc display Maya plug-in, a ptc file extraction function set is added, providing the freedom of writing out a new ptc with desired color, percentage, and even certain chunk of the original ptc file.

The Renderman point cloud API provides an example source code for a command line tool to merge two point cloud file into one. The example code is written in C language, but it was not hard to use it as a reference and update the Maya plug-in in C++.

The actual implementation of the ptc-writing out function sets is fairly easy:

A "writeptc()" function was programmed based on the "loadptc()" function in the original plug-in. Instead of storing all the required data to the C++ boost arrays and then displaying them in Maya interface, it writes out the ptc file points data. Initially this "writeptc()" function closes the write out ptc file with:

```
PtcClosePointCloudFile(ptc).
```

The function above was designed to close a read-in ptc, and when it is being used to close a Write-out ptc, the result will lost the last several thousand points supposed to be written out. For example, if a user wants to write out a ptc file contains 42,345 points, only 40,000 points
will be actually written with the "$PtcClosePointCloudFile(ptc)$".

Fortunately, it did not take too long to find out the correct function to close a Write-out ptc is:

\[ PtcFinishPointCloudFile(ptc) \]

After that, the written out ptc files are always contain desired amount of points. A thread was submitted to the PRman support forum asking why the two functions are causing different results, but no clear answer was given.

**Disc Space Management**

There is a disc space limitation (10 G bytes) applied for each user on the SCAD network. However, a large amount of data such as effects simulation cache files, ptc files, and test render images have been generated during the thesis project which are largely exceeded the limitation. Additionally, the lab workstation where the thesis project primarily being worked on is only available when there is no regular SCAD classes taking place, so the workstation is rather public and providing access to any SCAD member. To prevent unnecessary loss, a portable external hard drive and a custom Python script have been prepared to back up the data. Given a working directory in the lab workstation and a backup directory in the external hard drive, the script will automatically synchronize the data from the working directory to the backup hard drive in every 10 minutes. This setup saved a lot time from manually managing tedious data back up operations.
**Render farm Setup/Hack.**

Similar to the disc space limitation, the render farm at SCAD is limited at 1 G bytes per user. Each user has a pre-allocated directory on the network, and this directory is where the render project files including Maya and Houdini scene files, texture files, cached simulation files supposed to be copied to before submitting the render job to the render farm. However, the cached simulation files have easily exceeded the limitation when working on the crash site shots. A simple fire simulation could result in cache files around 100 Mega bytes per frame, and consequently around 10 Giga (assuming a 100 frame shot) bytes per shot.

As a result, to meet the limitation, SCAD render farm users with excessive amount of data normally have to copy every several cached files to the render farm directory, and submit a separate render job for only those copied frames. It can be imagined that rendering such jobs is not efficient, and it is hard to keep track of what is being rendered when the render farm gets busier during time.

Fortunately, before a render farm node (a render farm controlled computer) actually starts rendering the project, it will copy the entire project folder into its own local temporary directory. In addition, both Maya and Houdini supports pre-render and post-render script, which means a Python script can be run before or after a frame gets rendered. This feature provides accessibility to "hack" the limitation:

Firstly, prepare all the cached simulation files or any other large disc-consuming files on a computer in the same network as SCAD render farm, and set the copied directory with sharing status turned on.

Secondly, a pre-render Python script is attached to the actual to-be-rendered Maya or
Houdini scene file. Before a frame starts render, the pre-render script will find what the currently render farm node is, and copy all the necessary cache files from the shared network directory into the copied project directory. In this case it will be copied into the render farm node's temporary directory. As a result, even though the cache files are not stored in the user's render farm directory, the actual rendering temporary directory will have them accessible.

Thirdly, when the render farm job finishes, a post-render script will remove all the copied cache files, in order to keep the render farm node's temp directory clean and ready to use for the next round of render.

The pre and post render scripts approach works around the render farm limitation problem with a rather worry free solution. This method has been applied to the thesis project and saved a tremendous amount of time. The implemented SCAD render farm "hack" could be utilized to work on any other projects require large data interchange.

Finally, an example pre-render script will be similar to:

```plaintext
//beginning of pre-render script

string $currentFrame[] = `renderSettings -firstImageName`

python("splitString = \\
" + $currentFrame[0] + "\n")
$currentFrame[0] = python("%04d' %int(splitString.split('.')[-2])")

system("smbclient /mont-110-06-rh/temp/-N -Tc /tmp/render/oneD02_inter/transfer1.tar crashsite_ptc/csite_emiterA." + $currentFrame[0] + ".ptc")

system("cd /tmp/render/oneD02_inter;tar -xvf transfer1.tar")

string $ptcname = "/tmp/render/oneD02_inter/crashsite_ptc/csite_emiterA." + $currentFrame[0] + ".ptc"

setAttr -type "string" "RenderManLight1.bakename" $ptcname ; //end of the script
```
Figures

Fig. 2. Not accurate percentage display from first implemented ptc display plug-in, using numerical probabilistic algorithm.

Fig. 3. Correct percentage display in implemented ptc display plug-in.
Fig. 4. User interface to extract points from a ptc file.
Fig. 5. Extracted ptc points from a torus shaped ptc file.

Fig. 6. Fluid simulation in Houdini. Baked out the fluid velocity. Velocity driven particle system.
Fig. 7. VOPPOP shader uses baked fluid information as input to drive the behavior of the particle.
Fig. 8. Houdini render setting to export ptc files.
Fig. 9. Colorized particle in Houdini and baked ptc file result displayed in Maya.

Fig. 10. A Maya spot light with the custom PRman light shader which uses the ptc displayed in Fig. 9 as input. The button part of the image shows the render result.
Fig.11. Scene geometries and camera in Maya to be exported.

Fig.12. Scene geometries and camera exported from Maya and displayed in Houdini. DOP Network generated to create fluid simulation.
Fig.13. Colorized particle based on fluid's velocity and fuel information in Houdini. Houdini’s VOPPOP node tree also displayed.

Fig.14. Effects element ptc files displayed inside of Maya and the Maya light which uses the custom renderman light shader.
Fig. 14-2. User Interface of the custom Prman light shader.

Fig. 15. The fire interactive lighting pass rendered based on the ptc file while using the custom light shader in Fig. 14-2
Fig. 16. the fire interactive lighting pass for another crash site shot with same setup.

Fig. 17. final compositing of a crash site shot
**Definitions**

AE template - It is the attribute editor template inside of Maya. Customize the editor interface of a specific Maya node.

Animation curve - One type of Maya node stores Maya attribute values at different frames through the "timeline" inside of Maya.

Attribute - A certain value contains information of one type of data inside 3D software application.

API - Application programming interface, providing pre-defined and as-is programming language functions library for user to customize or extend the tool sets of a software application.

Array - C++ array, "An array is a series of elements of the same type placed in contiguous memory locations that can be individually referenced by adding an index to a unique identifier."[Cplusplus.2012]

Autodesk - "Autodesk, Inc., is a leader in 3D design, engineering and entertainment software. Customers across the manufacturing, architecture, building, construction, and media and entertainment industries"[Autodesk.2012]

Boolean - A data type in computer science, the value is either "True" or "False". Or it is a type of math operation such as union, difference, and intersection.

Bounding box - A smallest virtual box that can contain the exact shape of a model.

C++ - A programming language evolved from C language. It is mostly being used to create sophisticated software packages.

Cache file - File formats stores certain types of information that can be reused later per request.

Chan file - Channel file in Houdini, stores attribute values for frames through the time line.

Cmake - short term for Cross Platform Make, it is an open-source software to manage the build process of software.

Compositing - Digital Compositing," is the process of digitally assembling multiple images to make a final image, typically for print, motion pictures or screen display. It is the evolution into the digital realm of optical film compositing" [Brinkmann.2008]
Compute() - A function in Maya API to compute based on input parameters.

Digital Domain - A visual effects production studio with head quarter in Venice, California, USA. Visual Effects academy awards winner, famous work including Titanic, The Curious Case of Benjamin Button, Tron Legacy, and Real Steel.

Diffuse - Short term for diffuse reflection. It is "Reflection from a rough surface in which a collimated beam emerges in all directions." [wolfram.2012]


Eric Barba - Visual Effects supervisor, visual effects academy award winner, currently works at Digital Domain.

Effects - Computer graphic effects animation

Falloff - An attribute usually used in a computer graphic light. It determines how fast a light's energy decrease through distance.

Frame - One second in feature film is constructed with 24 frames. One frame thus means 1/24 step of a second.

Fuel - A fluid attribute. It defines how much energy can be used to burst into fire at a certain area of a fluid container.

Fluid - Short term for fluid simulation. "Fluid simulation is an increasingly popular tool in computer graphics for generating realistic animations of water, smoke, explosions, and related phenomena." [Sun, Chen. 2010]

Floor() - A math function to convert a float (number) into an integer. Given an input x, this function outputs the largest integer less than or equal to x.

G++ - An open source C++ equivalent in Linux operating system

Geometry - A shared name of Maya surface objects including polygon and nurbs

Gigabytes - A computer hardware storage measurement unit. 1 gigabytes equals to 1048576 bytes. A mainstream personal computer currently (as of January 2012) would have 500-1000 gigabytes hard drive capacity.

Global Illumination - "any rendering algorithm that simulates the inter-reflection of light between two surfaces." [Birm.2006]
Houdini - It is 3D design software application from Side Effects Inc, usually but not limited to be utilized to create effects animation in the visual effects industry.

Incandescence - A shading color attribute in a shader, it defines how much self illuminating a geometry using the shader is. The more this color, the less the shader will be affected by surrounding lights.

Illuminate calculation loop - A programming function loop inside Renderman Shading Language to calculate light rays of a light source.

Linux - An open source computer operating system. Commonly used in the visual effect industry.

Maya - Short term for Autodesk Maya. It is a 3D computer graphics software largely used in the Visual Effects industry.

Maya node - A node is a data block in Maya. One type of node does one specific job that other nodes cannot normally do.

Modeling - Short term for 3d modeling. "3D modeling (also known as meshing) is the process of developing a mathematical representation of any three-dimensional surface of object (either inanimate or living)" [Wisegeek. 2012]

Network - short term for computer networks.

Octree - "Octree is a tree where each internal (non-leaf) node has eight children. Each node spans a particular space area, expressed as an axis-aligned bounding box (available as Box property of TOctreeNode). Each node also has a chosen middle point inside this box (available as MiddlePoint property of TOctreeNode class). This point defines three planes parallel to the base X, Y and Z planes and crossing this point. Each child of given octree node represents one of the eight space parts that are created by dividing parent space using these three planes." [Sourceforge. 2012]

Open source - Previously proprietary (or private) software or source code released to public for free use and modification.

Particle - Short term for particle system. a computer graphics technique to simulate certain fuzzy phenomena such as flocking birds and fire.

Plug-in - A programmed add-on to extend functionalities and features to existing software does not have. Plug-ins inside of Maya are created with Maya API.
Projection - Project a sequence of images from the render camera back to objects in the scene.

Pseudo-random - a numbers that computer graphically generated to approximates the properties of random number. It's not truly random because given a same generating seed, the generated pseudo-random numbers are always determined as the same.

Random number - "A random number is a number generated by a process, whose outcome is unpredictable, and which cannot be subsequentially reliably reproduced."

[Randomnumbers.2012]

RAM - Random access memory (RAM), is a relatively quicker form of computer data storage than computer hard drive. The data stored on RAM are usually dynamic and will be erased when the computer is powered off.

Render - The process of rendering.

Render crash - Renderer failure generally caused by lack of memory or bugs inside the rendering software application.

Renderer - Alias for render engine. Common render engines used in visual effect industry are PRman, VRay, Arnold, and Mental Rey.

Render farm - A grid of computers to perform rendering tasks,

Prman - Short term for Photorealistic Renderman, a powerful computer graphics render engine from Pixar.

Round() - A math function to convert a float (number) into an integer. Given a input x, this function outputs result from the "floor()" function with the input x+0.5.

Scene file - An interactive description contains necessary nodes to fulfill a meaningful artistic or technical purpose in 3D software applications.

Sequence - A sequence is constructed with several (similar) shots which normally can be cut one after another.

Set light - Real world lighting setup at the movie set when shooting the live action footage.

Shader - "Task of describing the surface characteristics of objects" in 3D computer graphic program. [Calahan, Sharn. 1996]

Shading node - A Maya node can be used as component of a complex shader.
Shadow - an area where light cannot directly reach.

Shot - A shot is one small chunk of a film/movie/TV program with normally one single camera move.

Specular - Short term for specular reflection. Specular reflection is the mirror-like reflection of light from a smooth surface.

Temperature - A fluid attribute. It defines how much heat and thus how likely the fuel can be lit on fire at a certain area of a fluid container.

Texturing - Short term for texture mapping. Texture mapping applies an image to a surface. [MSU. 2012]

Torus - A torus shaped geometry in Maya.