Approximating the Appearance of Human Skin in Computer Graphics

Ben Jones
March 2006

Abstract

The simulation of realistic lighting and shading for the reproduction of human skin in computer graphics is a subject of great depth. This paper deals with a method for implementing an approximation of the appearance of human skin in a RenderMan shading network, and using Pixar's 'prman' scan-line native renderer.

Keywords

BRDF, BSSRDF, Subsurface scattering, reflection models, realistic image synthesis, Fresnel effect.

1. Introduction

The rendering of complex surfaces in computer generated imagery is a visually driven field, and most of the time the final image is the only thing which is important. Physical phenomena and physically based light transport solutions in computer graphics can look stunning; however, it is extremely computationally intensive. Therefore the solutions used to shade complex surfaces are only very rarely based on real-world physics. The majority of computer generated shading algorithms are based on approximations of these complex real-world observations, and yet it can be more than adequate to trick the viewer into accepting the visuals as 'real'. The rendering of a material such as skin is complex as it has such a varied and detailed surface, and light is reflected, refracted, scattered, and absorbed in many complicated ways.

1.1 Previous work

There has been a significant amount of work that relates to the shading of computer generated surfaces since it is of critical importance to the rendering of photorealistic images. The bidirectional reflectance distribution function (BRDF) was first introduced by Nicodemus [3] as a simplification of the complex light transport that occurs in real-life as light interacts with surfaces. This approximation is a basic solution for the appearance of computer generated materials, however, many of the subtleties of materials such as marble, skin, and fruit are lost in such a crude approximation. Such materials, as with most materials in real life, do not consist of a finite shell, and light is able to pass into and interact with the inside of a surface. The BRDF approximation in computer graphics makes the assumption that light is reflected from its exact point of incidence, and therefore does not account for the fact that light may exit a surface at a different point and with different attributes as it is scattered and absorbed within the material. The images resulting from a simple BRDF approximation are usually harsh representations of such materials. There has been much research in this area and much study and development of light reflection models to incorporate this subsurface scattering. There are many ways of computing the bidirectional surface scattering reflection distribution function (BSSRDF) of a material. One method is to compute the full radiative transfer equations for sub surface transport [4], which can be extremely slow, and on current computer hardware may be considered unusable for animated sequences. Dorsey et al. [5] implemented this method using photon mapping for the realistic rendering of weathered stone. An approximation of this subsurface scattering was presented by Jensen [1], where an adequate solution is computed in a fraction of the time, by using a rapid hierarchical integration technique to evaluate a diffusion approximation based on the irradiance samples at the surface of the object. The method described by Jensen [1] for computing sub surface scattering in translucent materials was incorporated into a stand-alone program that accompanies version 12.5 of Pixar's "prman" renderer. Jensen's method, implemented in this case, calculates the scattering from a three dimensional map consisting of points on a surface which contain values of the colour and intensity of direct illumination (or irradiance) at that point. It computes new values for these points according to parameters of scattering and absorption that define that material.

These values for scattering and absorption can only be represented approximately because many different types of material within the surface exhibit different properties (skin, for example, would be influenced by blood, tissue, ligament and bone etc.) therefore the values used are a recorded average for the material as a whole. The recording of these reflectance values of various materials has been studied by many including Debevec [6], but a more robust collection of coefficients for scattering and absorption have been recorded by Jensen [2] for various test materials including marble, skin, chicken, apple and milk among others.

2. Method

The methods used to create the images in this project follow an ideal that good approximations of complex real-world phenomena can produce visuals of adequate quality, and are flexible and fast to render without the use of complex rendering algorithms. The speed of render is of importance, but not at the sacrifice of image quality, therefore a balance must be found (depending on the artist's patience, or deadline). The shaders used in this project are written using the RenderMan shading language and the images are rendered in Pixar's 'prman 12.5' renderer. The renderer is used as a scan-line renderer without ray-tracing or any global illumination, which leaves adequate time in the render for other important processes such as real depth of field and motion blur.

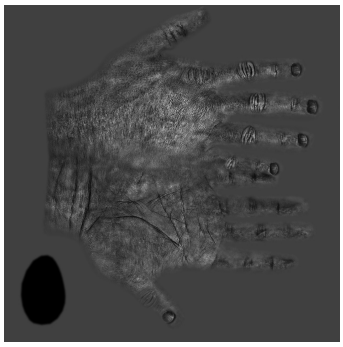
2.1 Descriptions of Skin

Skin is a particularly complex material as it has many attributes that contribute to its appearance. Skin is made up of many layers, all of which exhibit different scattering and absorption attributes, the epidermis, which is the top most layer of tissue (save for the thin film of oil which rest upon it) and is very translucent, allowing a lot of light through to the relatively thick dermis layer below. This layer contributes to the colour of the light partially as it passes through small capillaries where it is tinted by the blood. The dermis is still very translucent and light is allowed down into the hypodermis. Light is scattered greatly in the hypodermis layer and collects much of the colour of the blood vessels that it passes through. All the layers of the skin are translucent and light may travel a great way through the skin and into the tissue below, all the while being scattered greatly. There is an overall red tint to the light that scatters in the lower layers of skin and tissue (as can be seen if you hold your hand to a strong light), as most of the colouration of light is attributed to the blood it passes through. The light transport in skin is also affected by ligaments and bones that are

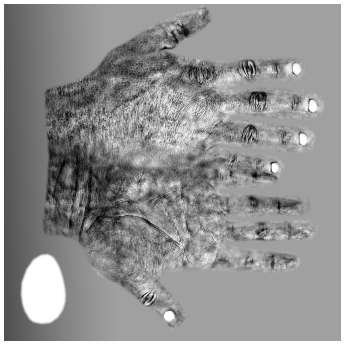
roughly opaque. Therefore, in computer graphics the synthetic rendering of any method that is physically correct is extremely computationally intensive. The use of approximations for this interaction of light is currently essential, however for such a material it may not be adequate to use a simple BRDF solution as the sub-surface scattering element is such an integral attribute of the surface, and a BSSRDF solution should be used.

To capture the faceted surface appearance of skin on a human hand, a high resolution three dimensional scan of a hand would be ideal; however due to the limitation of equipment available during this project a different approach was needed. A low detailed model of the geometry was modeled by hand and a displacement map created from macro photographs of a hand. The photos were taken at 12 million pixels each, using a relatively inexpensive Fujifilm S7000 digital camera, and under such lighting conditions as to accentuate the grooves and patterns on the surface. Then a map of the surface facets was stitched together by hand in Adobe Photoshop. The drawbacks of this approach compared to a high resolution scan are only apparent where the texture seams run along the geometry and a visible seam is obvious. A package such as Zbrush or any 3D painting package would allow the seamless integration of the texture and displacement maps along the seams of the model, and allow for more detail around that area, but, again due to the limited availability of such packages, and the time constraints of this project this avenue was not explored, and remains an option for future work. Colour, and specular maps for the skin were also painted by hand in Photoshop.

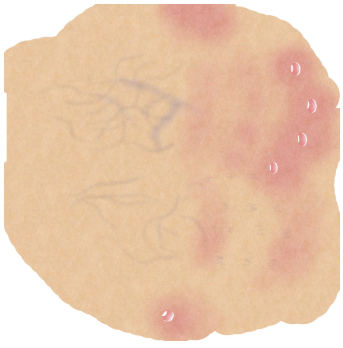
Macro Photograph of a section of skin on the back of the hand, many images like this were stitched together to create the displacement and specular maps shown.



Displacement map for the hand.



Specular map for the hand



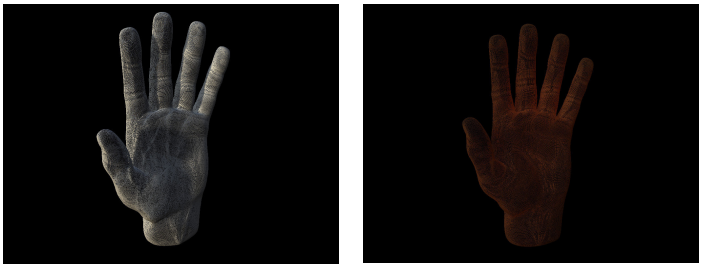
Colour map for the hand

2.2 Diffuse Illumination

The illumination of an object in computer graphics is approximated by many methods, and usually either by pseudo-light sources or by other more complex rendering techniques such as Image-based lighting and global illumination. As speed was a consideration in this project, the approach chosen to illuminate the scene was a simulation of Image-based illumination approximated by a dome of virtual lights coupled with ambient occlusion in the object shaders. The 'LightGen' plugin for HDRShop by Jonathan Cohen was used to obtain the dome of lights with colours and intensities taken directly from a high dynamic range light probe image. This approach lowers render time significantly and is an adequate solution for this project. It also allows the same HDR image to be put to other uses, such as environment and reflection maps and still be considered 'correct' with respect to the lighting. The diffuse component of the skin is a simple lambertian approximation, as used in most computer graphics shading.

2.3 Subsurface Scattering

The 'ptfilter' program which accompanies version 12.5 of Pixar's 'prman' renderer, and is based on the methods outlined in Jenson's papers [1] and [2], operates on a RenderMan 'pointcloud' file. As Jenson's sub surface scattering approximation can operate on values of intensity and color of incident light on the surface of a material, this information is baked out into a pointcloud file from a separate baking shader. The ptfilter program is then run on the pointcloud file and filters it according to Jensen's subsurface approximation. The parameters of the ptfilter program are the three coefficients for red, green and blue scattering: the three coefficients for red, green and blue absorption, the index of refraction of the surface, and an overall scale parameter. Jensen [2] recorded the value of the index of refraction for human skin to be 1.3, and also the skin scattering and absorption coefficients to be RGB(0.74, 0.88, 1.01) and RGB(0.032, 0.17, 0.48) respectively. Jensen's values for skin scattering and absorption coefficients were recorded from the arm of a Caucasian male, and therefore will serve as a good approximation for rendering of the hand in this project. Once the pointcloud information is filtered, it is stored in another pointcloud file, and then this new file can be read back into the shader as a three dimensional texture and added to the diffuse component of the shader. The separation of the sub surface scattering calculation and the final image rendering means that the final render is much quicker, and the artist may tweak other parameters of the shader without recalculating the sub surface scattering component. This method makes the process a distinct three-stage operation, and therefore a bit more organization is required to allow the method be handled in a production pipeline. Using Maya as the base package for producing the RenderMan 'rib' files in this project it was useful to write a series of MEL scripts to try and manage this 3 stage process. [Appendix C].



Irradiance data (left) in the form of a pointcloud file, and the filtered pointcloud (right) using the 'ptfiler' program and parameters from Jensen [2] for Caucasian skin. Notice the overall red tint to the colour of the filtered data, which comes from Jensen's recorded skin parameters.

2.4 Specular

Skin is covered with a thin film of an oily substance produced by the Sebaceous gland to lubricate the skin and hairs; this layer and the very top epidermis are the main contributors to the reflection of the surface, which is approximated using a slightly modified specular function. The main addition to the specular component is an approximation of the Fresnel effect, whereby two specular functions are calculated with different parameters and mixed together according to the angle between the vector to the viewer and normal vector of the geometry. This is a common approximation in computer graphics as it allows specific alteration of parameters of either the facing specular function (where faces of the object are roughly perpendicular to the viewer) and the glancing specular function (for faces that are more orientated to ninety degrees to the viewer). Therefore the Fresnel effect can be approximated crudely but with great flexibility. A specular map for the shader was constructed as an alteration of the displacement map to accentuate the faceted surface of the skin, and also to distinguish the parts of the skin with a shinier appearance.

Sub Surface Scattering

Facing Specular

Glancing Specular

Ambient Occlusion

2.5 Hair

Whilst hair is not strictly an attribute of skin, it may be argued that it is of great importance in the realistic image synthesis of skin, as there are very few places on the human body that have no hair on the surface (only the palms of the hands and soles of the feet). Therefore a very simple hair approximation was included to aid the overall realism of the image. Indeed the rendering of synthetic hair in computer graphics demands its own multitude of research papers, and therefore in this project the hairs are rendered simply with a lambertian diffuse, phong-like specular, and transparency and translucency coefficient. The geometry for the hair consists of strips of polygons.

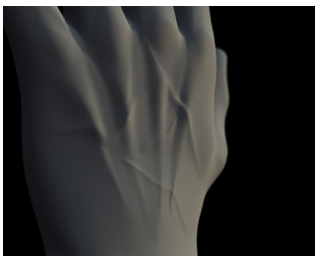
3. Results

Here are a selection of final renders showing various components of the final image, and comparisons between traditional BRDF and a BSSRDF solution. The scene consists of a dome of 25 lights (the brightest of which casts RenderMan 'deep shadows'), the hand object, a simple plane base, and an environment sphere textured with the High Dynamic Range image that was used to generate the lights. The images are rendered in scan-line mode, at 1280x1024 resolution, with 8x8 pixel samples, a shading rate of 0.25, and real 3D Depth of Field in Pixar's 'prman' renderer. Images were rendered on an Alienware MJ-12 workstation with Dual Opteron 244 processors, and would take roughly 3 to 4 minutes per frame (including all passes).

3.1 Layer Breakdown

Below is a breakdown of the individual layers that make up the shader on a close-up section of the hand.

Colour pass



Diffuse Illumination

The final composite of the layers above:
Occlusion * (Color * Diffuse + Glancing Specular + Facing Specular) + SS Scattering
It also includes the hairs on the hand, and an environment which is the HDR image used to generate the data for the lights.

3.2 BRDF and BSSRDF

Below is a comparison of a traditional BRDF solution (the same components of the shader without the subsurface scattering layer), and the BSSRDF approximation in the final shader.

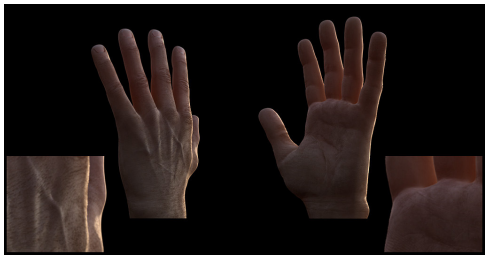
A BRDF solution.

A BSSRDF solution.

Note how in this implementation a lot of the color for the skin is attributed to the sub surface scattering and without it the skin lacks its distinctive warm tones and looks quite dead.

3.3 Light Interaction

See attached video file, showing a how each layer interacts with a rotating environment of lights, and then the build up of the layers into a final composite, and also an example of the shader without the sub surface scattering contribution.



Still frame from the video sequence showing the hand in a moving environment.

3.4 Extreme Close-up

Close up of the back of the hand, this is about the closest it is possible to get to the geometry without any deterioration of the image due to the resolution of the maps used for specular, displacement and color, unfortunately the texture maps used were only at 4k resolution as the 8k version that was originally used had to be scaled down to improve interactivity in Photoshop. Note that if a 3D scan were used, the limit would depend on the resolution of that scan, and it may be possible to get good results far closer.
It is also worth noting that the method of using displacement maps means that the closer you come to the geometry the slower the render times will be, as this render was roughly 3 times as long as the ones above.

4. Further Work

Areas of interest and expansion in this area could include the animation of the subsurface scattering coefficients which are used to filter the point cloud files. Whilst these coefficients are approximately constant in real-life, in the field of computer graphics we are not bound by these constraints and animation of the scattering and absorption coefficients could result in some intriguing effects.

Another obvious expansion of this project is the animation of the hand, and realistic simulation of the skin, tissue and ligaments.

One drawback of Jenson's method [1] is its inability to deal with opaque materials inside the surface that affect the transport of light (such as bones etc.). Jensen himself noted the possibility for expansion to deal with this as a future feature of the algorithm [1].

The three stage approach to the method could be considered cumbersome for use in a production pipeline, and whilst scripts for handling this method can automate the process to a certain extent, the ability to adapt this method of scattering into a single stage operation would be very useful. However, the flexibility and speed associated with this method of pre-computing the scattering may be lost in a single stage solution.

This solution for the shading of human skin (while it could be considered adequate in some cases) is relatively naive, and due to the layered nature of skin, a layered shading model would allow for greater realism and flexibility by simulating the different attributes of each skin layer separately.

An interesting subject relating to the shading of skin would be the procedural creation of the facets and grooves in human skin. This is very complex problem as there is a multitude of different patterns in the skin, no pattern is the same, and they also merge and flow into one another seamlessly on the surface. There are general rules as to the shape, position and pattern of these grooves that depend on the part of the human body in question, however the patterns vary subtly and almost infinitely from human to human. If such a procedural solution was developed it would remove the problems associated with resolution and seams of three dimensional scans, or hand painted maps, and allow for rendering at any distance from the surface.

The images in this project are for Caucasian male skin specifically, however human skin is so varied (even within this group), that factors of age, sex, and race affect skin properties greatly. Environmental factors such as heat and humidity will affect colour and texture of skin, as well as the natural functions of the body such as sweat, movement of the hairs on the skin's surface and the effect of pimpling of the skin in cold conditions. Skin also interacts with other materials in complex ways, and materials such as dirt, paint, water, oil, change the appearance and interaction of skin severely. Most of these interactions could be simulated in the skin shader, and leave this area open to a seemingly unlimited amount of future development.

5. Conclusion

This method for the fast rendering of realistic human skin is a crude approximation and fast to render, but may well be adequate in visual appearance. Using approximations for attributes of the skin such as the faceted appearance of the surface, scattering of light inside the tissue, and diffuse and specular reflectance, the resulting image can be rendered quickly and without advanced rendering algorithms such as ray-tracing or global illumination.

6. Critique

Having no real previous experience in shader writing, this project was an interesting challenge, and I feel it has been a valuable learning experience in being able to understand how surfaces interact with light, and the ways that we can simulate it in computer graphics. I found the RenderMan standard for shader writing more useful and intuitive than a GUI-based system, and found that the explicit operations on values of the shader made it easier to visualize exactly what the shader was doing with each pixel, and therefore understand better how variations in the surface would affect the light.

I feel the results of the project are of a standard that I am pleased with; although I had originally intended to include another variable element into the shader, such as water resting on the surface or another similar attribute. Addition of such an element would be a natural progression of the current state of the project; however it is only time constraints that have prevented it.

One section of the project that I am slightly disappointed with is the approach to recording the maps describing the surface of the hand. The photography and stitching of photos went well; however, due to

the hardware constraints of today's machines I could not manipulate the photographs in their original resolution, and had to scale down the maps by fifty percent in order to regain the interactivity. This meant that some detail from the original photos was lost in the final maps.

The greatest problem with using this method of displacing the geometry came with the seaming of these textures together along the edges of the virtual model. A hand is a complex piece of geometry for standard texture projection techniques and I found that in order to reduce the visibility of these seams it was necessary to reduce the amount of detail around these areas in the skin texture maps, which in turn created areas of low detail around these seams, and partially undermined the aims of the project. Had I had the time and equipment available, a three dimensional scan, would have been ideal, as it would have resulted in anatomical accuracy as well as accuracy in the surface detail. I would have liked to spend more time exploring ways to merge the seams whilst retaining the necessary detail.

I had ruled out movement of the hand early on in the project, perceiving the multitude of difficulties in creating realistic movement of the skin. The fact that anything less than good simulation of its movement could undermine the realistic appearance of the skin was apparent to me. However, I should have liked to animate the geometry to show how the skin shader would adapt as the geometry is manipulated.

I experimented with a few methods for simulating the effect that the bones within the hand would have on the transportation of light through it, but unfortunately none of these experiments provided results of sufficient standard. I feel it is an important attribute of a hand and would have liked to explore it more.

Throughout the project I have explored and experimented with the RenderMan shading standard, and Pixar's prman renderer, and feel that I have a good understanding of the systems and their processes. I would now count it as my rendering and shading method of choice. I am pleased with the outcome of my shaders, and feel that it would be easy and flexible to adapt it to simulate many types of skin by simply creating new textures for surface variation, reflectance and colour.

A significant amount of effort on this project went into the creation of management scripts for the integration of this method and the base 3D package of Maya. I am pleased with the results of these automation scripts, and I will continue to develop them for use in future projects that incorporate this shading method.

7. References

[1]
Henrik Wann Jensen and Juan Buhler.
A Rapid Hierarchical Rendering Technique for Translucent Materials.
Proceedings of SIGGRAPH 2002.

[2]
JENSEN, H. W., MARSCHNER, S. R., LEVOY, M., AND HANRAHAN, P. 2001.
A practical model for subsurface light transport.
In Proceedings of ACM SIGGRAPH 2001, ACM Press/Addison-Wesley Publishing Co., New York, Computer Graphics Proceedings.

[3]
F. E. Nicodemus, J. C. Richmond, J. J. Hsia, I. W. Ginsberg, and T. Limperis.
Geometric considerations and nomenclature for reflectance.
Monograph 161, National Bureau of Standards (US), October 1977.

[4]
S. Chandrasekhar.
Radiative Transfer.
Oxford Univ. Press, 1960.

[5]
J. Dorsey, A. Edelman, H. W. Jensen, J. Legakis, and H. K. Pedersen.
Modeling and rendering of weathered stone.
In Computer Graphics Proceedings, Annual Conference Series, 1999, August 1999.

[6]
P. Debevec, T. Hawkins, C. Tchou, H. Duiker,W. Sarokin, and M. Sagar.
Acquiring the reflectance field of a human face.
In Computer Graphics Proceedings, Annual Conference Series, 2000, July 2000.

Appendix A: Further Images

Appendix B: RenderMan Shaders

```
////////////////////////////////////////////////////////////////////////////////////////////////////
//
// SHADER FOR BAKING OUT AN IRRADIENCE POINTCLOUD
//
////////////////////////////////////////////////////////////////////////////////////////////////////
    surface
    bjbakeptc
        (
        //PARAMETERS
        float Ka = 1;
        float Kd = 1;
        string opactex = "";
        string disptex = "";
        string dispspace = "";
        float truedisp = 1.0;
        float scaledisp = 1.0;
        string pointcloudfilename = "";
        string displaychannels = "_area,_radiance_t";
        )
    {
        //LOCAL VARIABLES
        color Omap= (1.0, 1.0, 1.0);
        float Dmap= 0.0;

        //READ IN A TEXTURE FOR DISPLACEMENT
        if(disptex !="")
        {
                Dmap= float texture(disptex);
                N = Displace(normalize(N), dispspace, scaledisp*Dmap, truedisp) + (N-Ng)*10;

        }
        //READ IN A TEXTURE FOR TRANSPARENCY
        if(opactex !=""){ Omap= color texture(opactex);}

        normal Nn = normalize(N);

         color irrad, rad_t;

        float a = area(P, "dicing");

         irrad = ambient() + diffuse(Nn);

         rad_t = irrad;

        //BAKE OUT THE IRRADIENCE POINTCLOUD FILE
         bake3d(pointcloudfilename, displaychannels, P, Nn, "interpolate", 1,"_area", a,"_radiance_t", rad_t);

        Ci = rad_t;
         Ci *= Omap;
         Oi = Omap;
    }

////////////////////////////////////////////////////////////////////////////////////////////////////
//
// SHADER FOR RENDERING SKIN
//
////////////////////////////////////////////////////////////////////////////////////////////////////
    surface
    bjskin
        (
        //SHADER PARAMETERS
        float Ka = 0;
        float Kd = 0.45;
        float KsGlance = 1.2;
        float KsFace = 0.6;
        float Groughness = 0.0;
        float Froughness = 0.0;
        string coltex = "";
        string spectex = "";
        string opactex = "";
        string disptex = "";
        string dispspace = "";
        float truedisp = 1.0;
        float scaledisp = 0.06;
        color specularcolor = (1.0, 1.0, 1.0);
        float subsurface = 1.0;
```

```
    color subsurfacecolor = (1.0, 1.0, 1.0);
     string pointcloudfilename = "";
    float occlusion = 1.0;
    float occsamples = 64;
    float occmaxdist = 30;
    float occconeangle = PI/2;
    output varying color C=(0.0, 0.0, 0.0);
    output varying color Cdif=(0.0, 0.0, 0.0);
    output varying color Cspec=(1.0, 1.0, 1.0);
    output varying color Cgspec=1.0;
    output varying color Cfspec=1.0;
    output varying color Cocc=1.0;
    output varying color Csss=1.0;
    )
{
    //LOCAL VARIABLES

    color Omap= (1.0, 1.0, 1.0);
    float Dmap= 0.0;
    float facingratio=1.0;

    //READ IN A COLOR MAP
    if(coltex !=""){ C= color texture(coltex);}

    //READ IN A SPECULAR MAP
    if(spectex !=""){ Cspec= color texture(spectex);}

    //READ IN A DISPLACEMENT MAP
    if(disptex !="")
    {
            Dmap= float texture(disptex);
            N = Displace(normalize(N), dispspace, scaledisp*Dmap, truedisp) + (N-Ng)*10;

    }

    //READ IN A TRANSPARENCY MAP
    if(opactex !=""){ Omap= color texture(opactex);}

    normal Nn = normalize(N);

    vector V= -normalize(I);

    //CALCULATE FACING RATIO
    facingratio=Nn.V;
    Cgspec=Cspec*(KsGlance*(1-facingratio));
    Cfspec=Cspec*(KsFace*facingratio);


    //IF SUBSURFACE SCATTERING IS ON
    if(subsurface!=0)
    {
            //READ IN A FILTERED POINTCLOUD FILE
            texture3d(pointcloudfilename, P, N, "_ssdiffusion", Csss);
            Csss*=subsurfacecolor;
    }


    //IF OCCLUSION IS ON
    if(occlusion!=0)
    {
            Cocc = occlusion(P, Nn, occsamples, "maxdist", occmaxdist, "coneangle", occconeangle);
    }

    //CALCULATE SHADING COMPONENTS
    Cocc=(1 - Cocc*occlusion);
    Cdif=(Ka*ambient() + Kd*diffuse(Nn));
    Csss*=subsurface;
    Cgspec=(specularcolor*Cgspec*specular(Nn,V,Groughness));
    Cfspec=(specularcolor*Cfspec*specular(Nn,V,Froughness));


    //COMPUTE COLOR
    Ci = (Cocc*((C*Cdif)+Cfspec+Cgspec))+Csss;
     Ci *= Omap;
     Oi = Omap;
}
```

```
////////////////////////////////////////////////////////////////////////////////////////////////
//
// FUNCTION FOR DISPLACEMENT
//
////////////////////////////////////////////////////////////////////////////////////////////////
    normal Displace
        (
        vector dir;
        string space;
        float amp;
        float truedisp;
        )
{
        extern point P;
        float spacescale=length(vtransform(space, dir));
        vector Ndisp = dir *(amp / spacescale);
        P+=truedisp * Ndisp;
        return normalize(calculatenormal(P+(1-truedisp)*Ndisp));
}
```

Appendix C: MEL Scripts

```
////////////////////////////////////////////////////////////////////////////////////////////////
//
//
//      RENDERSCRIPT
//      BEN JONES 2006
//
//      SCRIPTS FOR THE MANAGEMENT OF RIBS, POINTCLOUDS, AND SHADERS,
//      WHEN USING POINTCLOUDS TO SIMULATE SSS
//
//      NOTE: THIS SCRIPT WAS WRITTEN TO AID A SPECIFIC PROJECT AND WHILE IT MAY WORK IN CERTAIN CASES
//      IT WAS NOT DESIGNED WITH OTHER USERS / USES IN MIND. I HOLD NO RESOPNSIBILITY FOR ANY DAMAGE CAUSED.
//      USE AT YOUR OWN RISK.
//
//
////////////////////////////////////////////////////////////////////////////////////////////////


////////////////////////////////////////////////////////////////////////////////////////////////
//
// GUI
//
////////////////////////////////////////////////////////////////////////////////////////////////


proc renderControl()
{
        //GUI

        if (`window -exists renderControl`)
        deleteUI renderControl;

        window -widthHeight 500 300 -title "BenderGlobals" renderControl;
        columnLayout -columnAttach "both" 5 -rowSpacing 7 -columnWidth 250;

             text -label "Name";
             textField name;
            text -label "Start Frame";
             intField startframe;
             text -label "End Frame";
             intField endframe;
            text -label "Bake point clouds for selected objects";
             intField doPointBake;
            text -label "Cleanup bake ribs and Filter PTC";
             intField doFilter;
            text -label "Do Final RibGen";
             intField doFinalRibGen;
            text -label "Render Final Ribs";
             intField render;
            text -label "Farm ON or OFF";
             intField farm;
```

```
        button -command "goRenderControl" "GO!";

        showWindow renderControl;
}

////////////////////////////////////////////////////////////////////////////////////////////////
//
// MAIN
//
////////////////////////////////////////////////////////////////////////////////////////////////

proc goRenderControl()
{
        // GET VALUES FROM GUI

        string $name = `textField -query -text name`;

        int $startFrame = `intField -query -v startframe`;

        int $endFrame = `intField -query -v endframe`;

        int $doPointBake = `intField -query -v doPointBake`;

        int $doFilter = `intField -query -v doFilter`;

        int $doFinalRibGen = `intField -query -v doFinalRibGen`;

        int $render = `intField -query -v render`;

        int $farm = `intField -query -v farm`;

        //CHECK FOR RIBGEN IF RENDER

        if ($doFinalRibGen == 0)
        {
                if ($render==1)
                {
                        error "Cannot render if ribs are not generated.\n";
                }
        }

        //GET MTOR VARIABLES

        string $path = `mtor project get`;
        string $origDspyName = `mtor control getvalue -rg dspyName`;


        // SET PROJECT NAME
        mtor control setvalue -rg dspyName -value $name;

        // IF POINT BAKING IS TURNED ON...
        if ($doPointBake==1)
        {
                // GET CURRENTLY SELECTED OBJECTS
                string $objects[] = `ls -sl -o`;

                // SET APPROPRIATE RENDERMAN GLOBALS
                mtor control setvalue -rg shadingRate -value 5;
                mtor control setvalue -rg pixelSamplesX -value 4;
                mtor control setvalue -rg pixelSamplesY -value 4;
                mtor control setvalue -rg dspyServerMode -value rgb;
                mtor control setvalue -rg dspyServer -value tiff;

                //FIND OUT ORIGINAL VALUE OF DOF AND MOTUION BLUR
                int $dof= `mtor control getvalue -rg doDOF`;
                int $mblur= `mtor control getvalue -rg doMotionBlur`;

                //TURN THEM OFF
                mtor control setvalue -rg doDOF -value 0;
                mtor control setvalue -rg doMotionBlur -value 0;

                //ASSIGN BAKING SHADERS
                assignBakeShader($objects, $path, $name);

                //GENERATE RIB FILES
                generateRibFile($startFrame, $endFrame);

                //RENDER RIB FILES
                renderrib($path, $name, $startFrame, $endFrame, $farm);
```

```
                //ASSIGN RENDER SHADERS
                assignRenderShader($objects, $path, $name);

                //RESET VALUES OF DOF ANS MOTION BLUR
                mtor control setvalue -rg doDOF -value $dof;
                mtor control setvalue -rg doMotionBlur -value $mblur;
        }
        // IF POINTCLOUD FILTERING IS TURNED ON...
        if ($doFilter==1)
        {
                //FILTER POINTCLOUDS AND DELETE BAKING IMAGES AND RIBS
                doptcfilter($path, $name, $startFrame, $endFrame, $farm);
        }

        // SET RENDERGLOBALS FOR FINAL RENDERING
        mtor control setvalue -rg shadingRate -value 0.25;
        mtor control setvalue -rg pixelSamplesX -value 8;
        mtor control setvalue -rg pixelSamplesY -value 8;
        mtor control setvalue -rg dspyServer -value tiff;
        mtor control setvalue -rg dspyServerMode -value rgba;

        // IF RIBGEN IS ON...
        if ($doFinalRibGen==1)
        {
                //GENERATE RIBS
                generateRibFile($startFrame, $endFrame);
        }
        // IF RENDER IS ON...
        if ($render==1)
        {
                //RENDER RIBS
                renderrib($path, $name, $startFrame, $endFrame, $farm);
        }

        //RESET PROJECT NAME AND DISPLAY SERVER
        mtor control setvalue -rg dspyName -value $origDspyName;
        mtor control setvalue -rg dspyServer -value it;


        print "COMPLETE!\n";
//      system ("mail -s \"RENDERS DONE!\" rhubarb72@hotmail.com");

}

////////////////////////////////////////////////////////////////////////////////////////////////
//
// ASSIGN BAKING SHADER
//
////////////////////////////////////////////////////////////////////////////////////////////////


proc assignBakeShader(string $objects[], string $path, string $name)
{
        print ("ASSIGNING BAKE SHADERS...\n");

        //for selected objects
        int $i;
        for($i=0;$i<size($objects);$i++)
        {
                string $shaderName = ($objects[$i]+"_BAKE");
                string $paramname = "pointcloudfilename";

                //list all slim appearances
                string $apparray[];
                string $apps = `slimcmd slim GetAppearances`;
                tokenize($apps,$apparray);

                //for all appearances
                for($j=0;$j<size($apparray);$j++)
                {
                        //is the appearance the correct baking shader for the object?
                        if ( $shaderName == `slimcmd $apparray[$j] GetLabel`)
                        {
                                //get the slim appearance key
                                string $key=`slimcmd $apparray[$j] GetID`;

                                //find the parameter that coincides with the point cloud filename
                                string $param = `slimcmd $apparray[$j] GetProperties -name pointcloudfilename`;
```

```
                                //set the parameter
                                slimcmd $param SetValue ("{" + $path + "ptc/" +$name+ ".$F4.ptc}");


                                //select the object and attach the shader
                                select -r $objects[$i];
                                mtor control attach surface $key;
                                print ("SHADER "+$shaderName+" ASSIGNED TO "+$objects[$i]+"\n");
                        }
                }
        }
        print ("FINISHED\n");
}


////////////////////////////////////////////////////////////////////////////////////////////////
//
// ASSIGN RENDER SHADER
//
////////////////////////////////////////////////////////////////////////////////////////////////


proc assignRenderShader(string $objects[], string $path, string $name)
{
        print ("ASSIGNING RENDER SHADERS...\n");

        //FOR SELECTED OBJECTS
        int $i;
        for($i=0;$i<size($objects);$i++)
        {
                string $shaderName = ($objects[$i]+"_RENDER");
                string $paramname = "pointcloudfilename";

                //list all slim appearances
                string $apparray[];
                string $apps = `slimcmd slim GetAppearances`;
                tokenize($apps,$apparray);

                //for all appearances
                for($j=0;$j<size($apparray);$j++)
                {
                        //is the appearance the correct baking shader for the object?
                        if ( $shaderName == `slimcmd $apparray[$j] GetLabel`)
                        {
                                //get the slim appearance key
                                string $key=`slimcmd $apparray[$j] GetID`;

                                //find the parameter that coincides with the point cloud filename
                                string $param = `slimcmd $apparray[$j] GetProperties -name pointcloudfilename`;

                                //set the parameter
                                slimcmd $param SetValue ("{" + $path + "ptc/" +$name+ ".$F4.ptc}");

                                //select the object and attach the shader
                                select -r $objects[$i];
                                mtor control attach surface $key;
                                print ("SHADER "+$shaderName+" ASSIGNED TO "+$objects[$i]+"\n");
                        }
                }
        }
        print ("FINISHED\n");
}

////////////////////////////////////////////////////////////////////////////////////////////////
//
//GENERATE RIB FILE
//
////////////////////////////////////////////////////////////////////////////////////////////////


proc generateRibFile(int $startFrame, int $endFrame)
{

        int $i;
        int $numFrames=$endFrame-$startFrame;

        print ("\nGENERATING RIBS...\n");
```

```
	//FOR EACH FRAME SPECIFIED GENERATE A RIB FILE
	for($i=0;$i<=$numFrames;$i++)
	{
		string $date= `system ("date")`;
		print ("Frame: "+($i+$startFrame)+"\t"+$date);
		mtor control genrib -frame ($i+$startFrame);
	}

	print ("FINISHED\n");
}


////////////////////////////////////////////////////////////////////////////////////////////////
//
// RENDER RIB
//
////////////////////////////////////////////////////////////////////////////////////////////////


proc renderrib(string $path, string $name, int $startFrame, int $endFrame, int $farm)
{
	int $i;
	int $numFrames=$endFrame-$startFrame;

	print ("\nRENDERING RIBS...\n");

	if ($farm==0)
	{
		//LOCAL RENDER
		for($i=0;$i<=$numFrames;$i++)
		{
			string $date= `system ("date")`;
			print ("Frame: "+($i+$startFrame)+"\t"+$date);
			string $padded = pad4($i+$startFrame);
			system ("render -p:2 "+$path+"rib/"+$name+"."+$padded+".rib");
		}
		print ("FINISHED\n");
	}
	if ($farm==1)
	{
		//FARM RENDER
		int $j;
		int $numFrames=$endFrame-$startFrame;

		print ("\nSORTING RIBS FOR RENDERFARM...\n");
		string $farmFolder =($path+"rib/"+$name+"_FARM"+$startFrame+"to"+$endFrame);
		system ("mkdir "+$farmFolder);

		for($j=0;$j<=$numFrames;$j++)
		{
			string $padded = pad4($j+$startFrame);
			system ("mv "+$path+"rib/"+$name+"."+$padded+".rib
"+$farmFolder+"/"+$name+"."+$padded+".rib");
		}
		print ("FINISHED\n");
		string $date= `system ("date")`;

//		system ("cd;cd "+$farmFolder+";farm;");

		print ("FOLDER... "+$farmFolder+"\tREADY TO SEND TO RENDERFARM\t"+$date);
	}
}

////////////////////////////////////////////////////////////////////////////////////////////////
//
// POINT CLOUD FILTERING, RIB DELETION, IMAGE DELETION
//
////////////////////////////////////////////////////////////////////////////////////////////////

proc doptcfilter(string $path, string $name, int $startFrame, int $endFrame, int $farm)
{
	int $i;
	int $numFrames=$endFrame-$startFrame;


	if ($farm==1)
	{
		//DELETE BAKING RIB FILES
		string $farmFolder =($path+"rib/"+$name+"_FARM"+$startFrame+"to"+$endFrame);
```

```
                print ("\nDELETING RIBS...\n");
                for($i=0;$i<=$numFrames;$i++)
                {
                        string $date= `system ("date")`;
                        print ("Frame: "+($i+$startFrame)+"\t"+$date);
                        string $padded = pad4($i+$startFrame);
                        system ("rm "+$farmFolder+"/"+$name+"."+$padded+".rib");

                }
                print ("FINISHED\n");
        }
        else
        {
                //DELETE BAKING RIB FILES
                print ("\nDELETING RIBS...\n");
                for($i=0;$i<=$numFrames;$i++)
                {
                        string $date= `system ("date")`;
                        print ("Frame: "+($i+$startFrame)+"\t"+$date);
                        string $padded = pad4($i+$startFrame);
                        system ("rm "+$path+"rib/"+$name+"."+$padded+".rib");
                }
                print ("FINISHED\n");
        }

        //DELETE BAKING IMAGES
        print ("\nDELETING IMAGES...\n");
        for($i=0;$i<=$numFrames;$i++)
        {
                string $date= `system ("date")`;
                print ("Frame: "+($i+$startFrame)+"\t"+$date);
                string $padded = pad4($i+$startFrame);
                system ("rm "+$path+"rmanpix/"+$name+"."+$padded+".*");
        }
        print ("FINISHED\n");

        //FILTER POINTCLOUD FILES WITH PTFILTER
        print ("\nFILTERING POINTCLOUD FILES...\n");
        for($i=0;$i<=$numFrames;$i++)
        {
                string $date= `system ("date")`;
                print ("Frame: "+($i+$startFrame)+"\t"+$date);
                string $padded = pad4($i+$startFrame);
                system ("ptfilter -ssdiffusion -maxsolidangle 1 -unitlength 0.23 -scattering 0.74 0.88 1.01 -
absorption 0.032 0.17 0.48 -ior 1.3 "+$path+"ptc/"+$name+"."+$padded+".ptc
"+$path+"ptc/"+$name+"."+$padded+".ptc");
        }
        print ("FINISHED\n");
}

////////////////////////////////////////////////////////////////////////////////////////////////
//
// PADDING
//
////////////////////////////////////////////////////////////////////////////////////////////////

proc string pad4(int $i)
{

        if ($i >= 1000)
        {
                return (string($i));
        }

        if ($i >= 100)
        {
                return ("0" + string($i));
        }

        if ($i >= 10)
        {
                return ("00" + string($i));
        }

        return ("000" + string($i));
}
```