12-2014

Underwater God Rays from a Custom Volume Renderer

Gowthaman Ilango
Clemson University, gilango@g.clemson.edu

Underwater God Rays from a Custom Volume Renderer

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Fine Arts
Digital Production Arts

by
Gowthaman Ilango
December 2014

Accepted by:
Dr. Jerry Tessendorf, Committee Chair
Dr. Donald House
Dr. Timothy Davis

Abstract

Peanut Butter Jelly, directed by Alexander Beaty, is a 51 second computer–animated short film produced by Digital Production Arts. The plot focuses on a fight sequence between a pirate jelly fish and a flyboy jelly fish over a peanut butter jar. The production demanded a photo–realistic computer generated underwater environment, which lead to the need for a custom built volume renderer to render high quality god rays. This thesis illustrates the requirement for a customized volume renderer for the production, the algorithm, and the implementation of the renderer. It also describes a tool created for Maya 2012 which gives the artist, artistic control to change the render settings.

Dedication

I like to dedicate my work in memory of my friend, teacher and mentor The Late Mr. Thilak Daniel.

Acknowledgments

I would like to express my sincere gratitude to my advisor, Dr. Jerry Tessendorf, for providing support and inspiration throughout my research. This thesis would not be possible without the background knowledge and guidance from Dr. Jerry Tessendorf and his Production Volume Rendering course. I thank Dr. House and Dr. Davis for not only serving as members on my committee, but also giving me the background knowledge I would need to implement my thesis. I would like to thank Alexander Beaty, the director of *Peanut Butter Jelly*, who carried out his vision and gave me an opportunity to be a part of this remarkable creation. I thank DPA alumnus Kacey Coley, for giving his deep image class which helped a lot in rendering godrays as deep images. I also like to thank the entire *Peanut Butter Jelly* team and DPA 891 group who gave me lot of constructive feedback. Finally, I want to thank my parents, my sister and her family who has been a great moral support.

Table of Contents

List of Figures

Chapter 1

Introduction

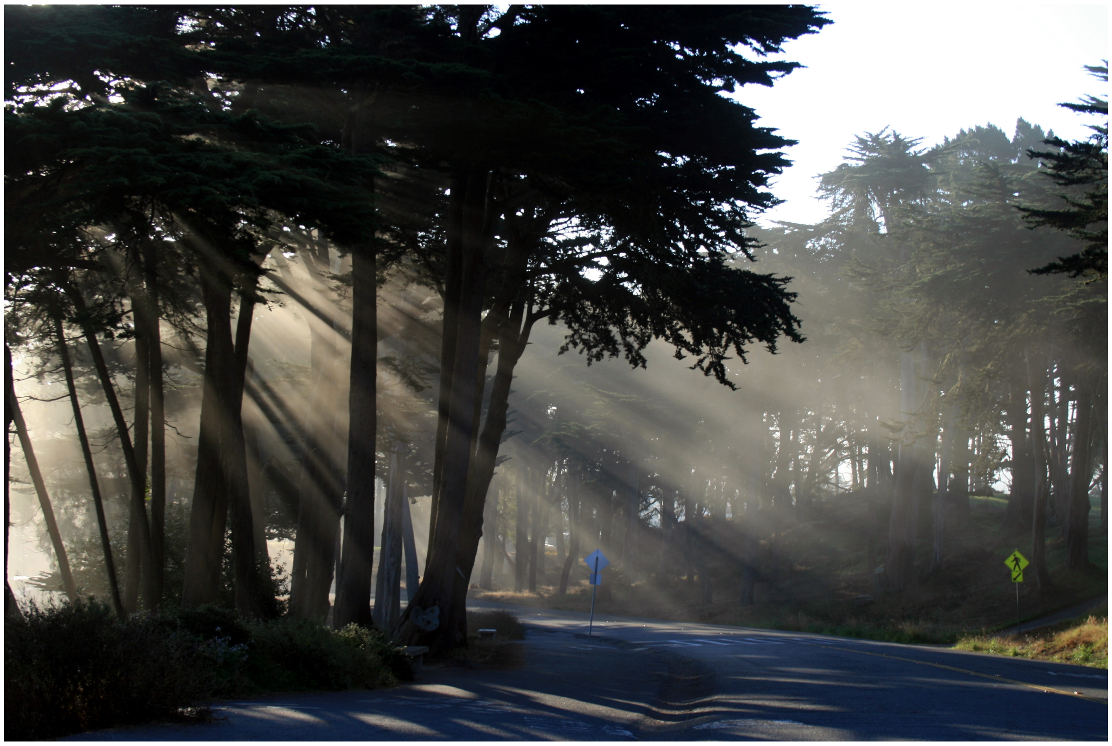


Figure 1.1: An example of godrays occuring in a natural environment
[8]

God rays, or Crepuscular rays, are shafts of sunlight that appear to originate from the point in the sky where the sun is located. These light rays are usually created when the sun is partly covered by a shadow inducing object like clouds, trees and buildings, and seem to converge at a point. In fact they are parallel shafts of sunlight and their convergence is a perspective effect as shown in Figure 1.1. Light shafts are also frequently observed in underwater scenes when looking up toward the water surface and the sun above. As water is a much denser medium than air, light interacts differently with water. The water surface has a great effect on light below the surface. Light rays refracted from the water surface can converge and diverge, scattering from water molecules and suspended atmospheric particles, generating the god ray appearance. An example of underwater light rays is shown in Figure 1.2

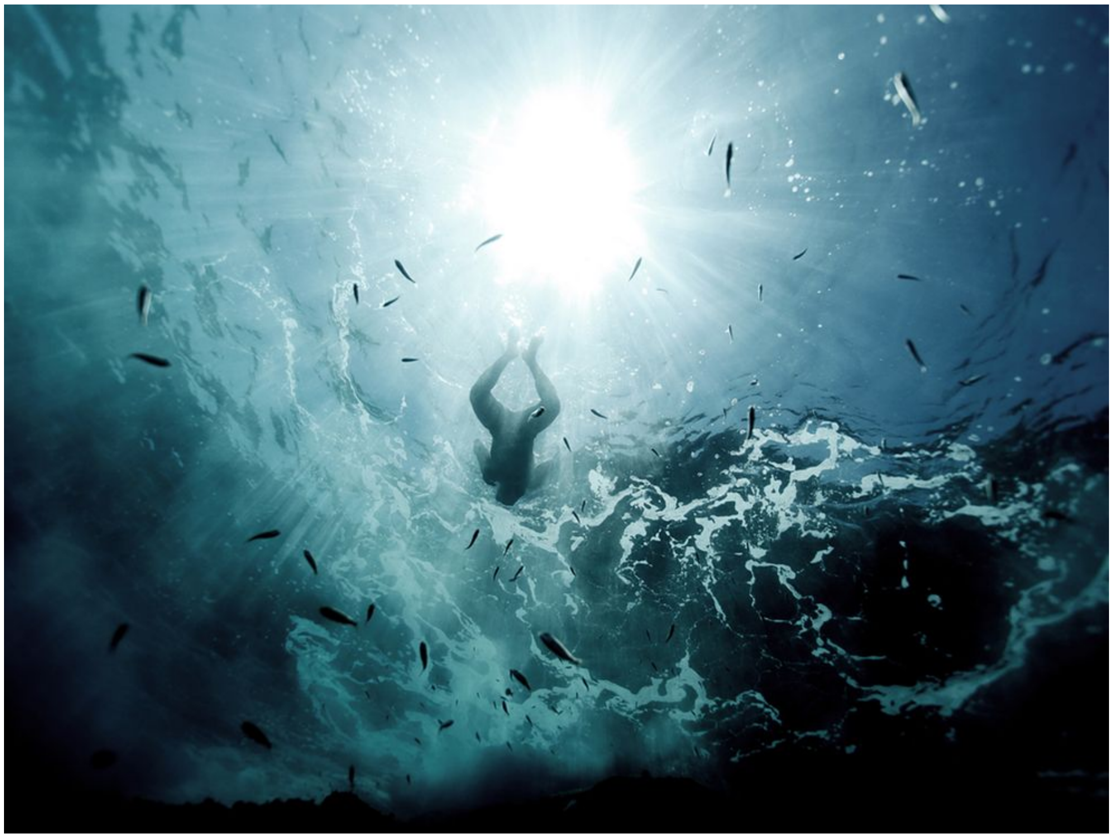


Figure 1.2: Underwater light rays
[10]

In computer graphics, lighting plays a key role in adding realism to the scene. It enhances the mood and emotion of a scene. Many optical effects are involved with underwater lighting including caustics, shaft of rays, reflections and refractions, and help to achieve highest degree of realism. The light shafts are not only and interesting optical illusion, but also add a dramatic effect to the scene.

There are several different approaches to render underwater god rays. One technique is to model the rays as geometric shafts [12]. Another method to get this effect is by sampling lights within a volume shader [7]. Matte painting techniques are also widely used to create god rays which can be added as a layer in compositing.

God rays have been used in many feature and animation films, for example, *Titanic* (1997), *Double Jeopardy* (1999), *Finding Nemo* (2003)(Figure1.3), *Pirates of the Caribbean: The Curse of the Black Pearl* (2003) (Figure1.4), *Shark Tale* (2004)(Figure1.5), *Superman Returns* (2006).

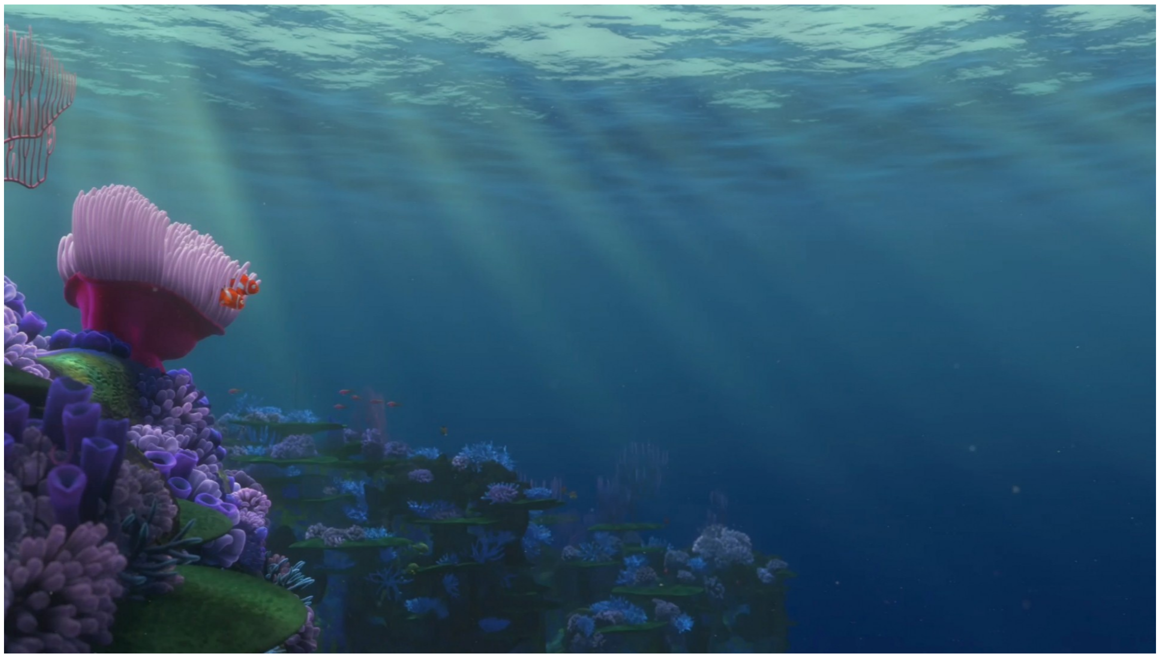


Figure 1.3: God rays from the motion picture *Finding Nemo*, Pixar Animation studios, 2003

In the above examples from feature and animated films, the light shafts are one of the important components providing authenticity to the underwater scene. *Peanut Butter Jelly* is a student production in which the entire story takes place undersea. The film required high quality visuals and realistic underwater looks which necessitated the design of a custom volume renderer.

This thesis presents the design and implementation of the custom volume renderer to render god rays. Chapter 2 describes the student production *Peanut Butter Jelly*, it's workflow and the need for creating a custom volume renderer. Background information in Chapter 3 defines Fresnel refraction and other important concepts in this paper which are essential for the renderer. Chapter 4 explains the implementation of the rendering algorithm and the tool created for Maya to access the renderer in DPA pipeline. Results of using the renderer for the production *Peanut Butter Jelly* are discussed in Chapter 5.

Figure 1.4: *Pirates of the Caribbean: The Curse of the Black Pearl*, ILM studios, 2003

Figure 1.5: *Shark Tale*, DreamWorks Animation, 2004

Chapter 2

Visual Development

Peanut Butter Jelly, directed by Alexander Beaty, is a computer–animated short film produced by Digital Production Arts. The plot focuses on a fight sequence between a pirate jelly fish and a flyboy jelly fish who are fighting over a peanut butter jar. The production was scheduled to be a one year project beginning in the month of January, 2014 with a team of 13 volunteer members. As the entire story occurs underwater, it demanded high quality visuals to generate photo realistic ocean environment. The story happens to take place in a Philippines coral reef under 40 feet depth and with clear visibility of water. The length of the short was framed to be 51 seconds with the expectation of professional quality output rather than a student level production.

The production used the DPA pipeline following a specific workflow to get the final output. The first stage of production involved visual development including research, storyboarding, character design and building the story arc. An adequate amount of research was done for the visual development of the film to create an authentic underwater ambiance 2.1. After substantial research and gathering sufficient reference images, the look of the film was finalized. A visual illustration of a shot which has all the key elements of the short film is shown in Figure 2.2. The next step of the workflow was storyboarding the entire story arc using sketches. Some of the storyboards by Alexander Beaty are shown in Figure 2.3

The sequence of steps following storyboarding were character design, environment design, modeling, surfacing, layout, animation, fx and lighting. The key components for a realistic computer generated (CG) ocean environment scene include lighting, particulate matter, haziness, caustics, reflections and refractions.

(a) Underwater scene

(b) Coral reef

Figure 2.1: Coral reef in Philippines
[2]

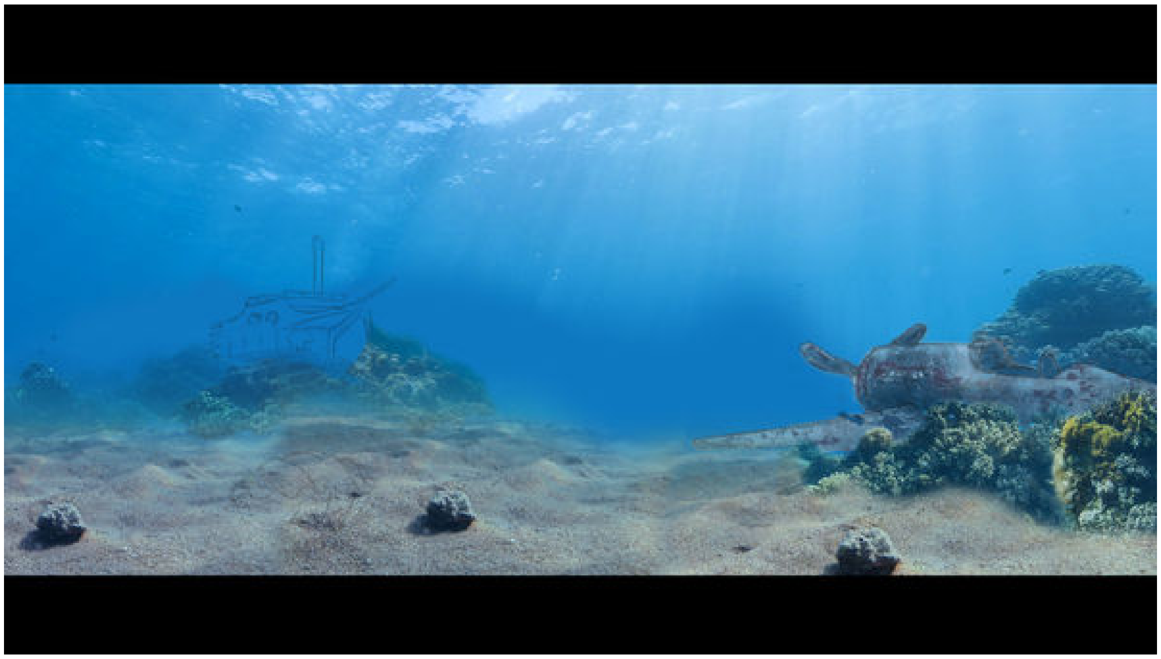
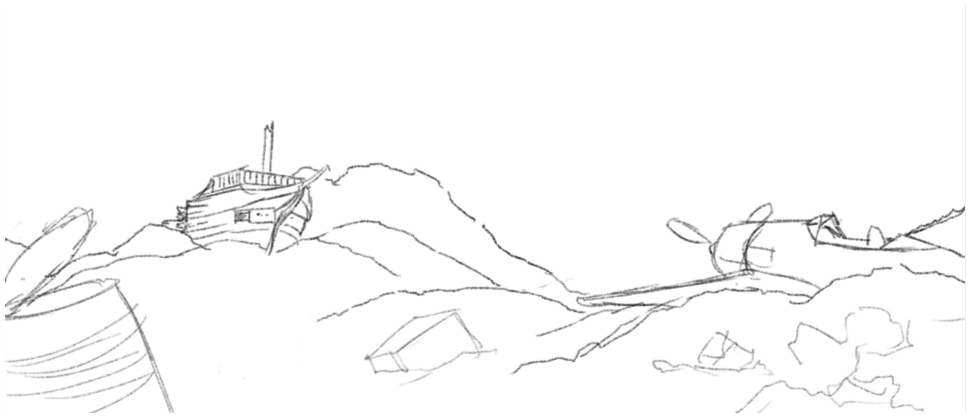
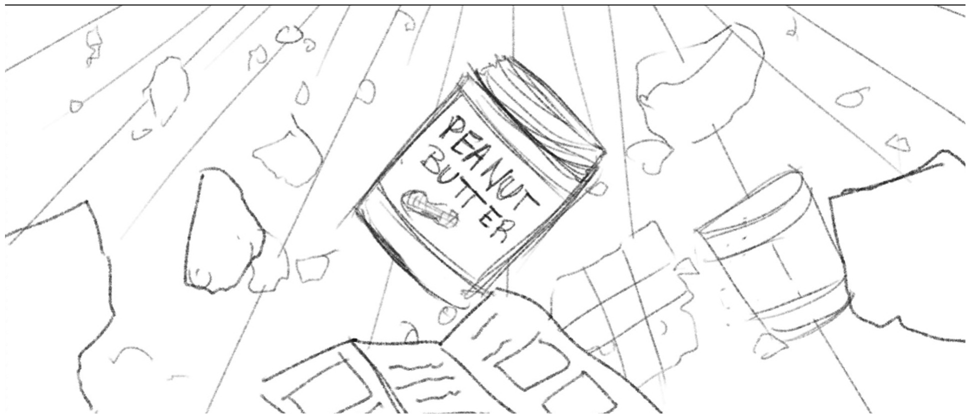


Figure 2.2: Visual illustration of *Peanut Butter Jelly* which includes the key components of the short

One kind of lighting effect which makes a CG undersea scene believable is godrays. As discussed in Chapter 1, there are different methods to render underwater godrays. In modeling geometric shafts [12], line primitives are generated as a grid of rays from the water surface and intersected with the scene. The light shafts produced from this method are perpendicular lines and do not look like underwater god rays because light shafts in real world can converge at a point. The movement of god rays generated by sampling lights inside a volume [7] is not dynamic and does not have varying structures for the light shafts. Matte painting is a tedious process, as each frame has to be hand drawn, and it is not physically based. The results of those methods are not convincing enough to produce a high quality aquatic scene. This instigated the need for an alternate method for generating high quality underwater godrays which has fine detail, proper structure, and good motion to enhance the look of the film. Volume rendering techniques can be used to achieve high quality output images to generate effects elements such as fog, smoke, fire, water splashes and other natural phenomena [16], hence a custom volume renderer was designed to render fine quality light shafts for *Peanut Butter Jelly*.


(a) Shot 01



(b) Shot 03

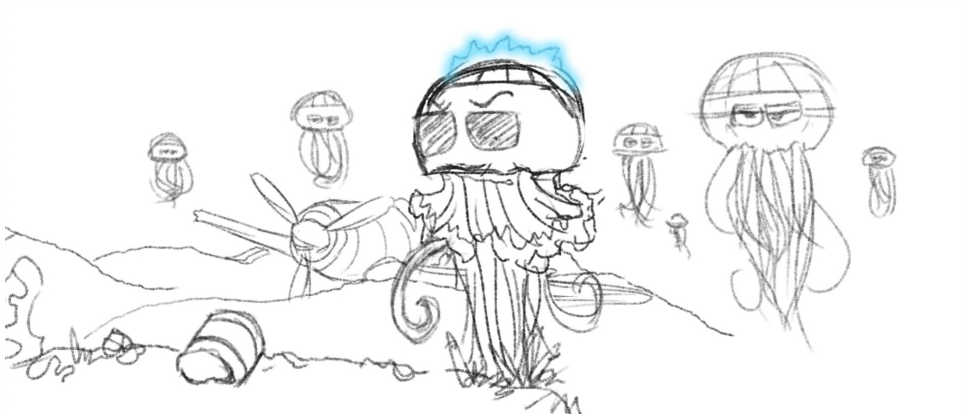

(c) shot 06

Figure 2.3: Storyboard of different shots from *Peanut Butter Jelly*

Chapter 3

Background

A volume is a 3D scalar field which has values at any point in 3D space. Volume rendering is a technique which displays the 3D information as a 2D image. The 3D data can be density, velocity, color or light intensity. This data is stored in 3D grids. The rendering technique commonly used in any volume renderer is ray marching [13]. In order to render the volume, there has to be a camera in the scene through which the 3D data can be viewed. For each pixel, a ray is generated which traverses through the volume. When it marches through the 3D data, it collects the color information at regular intervals. When the color values at regular intervals are accumulated, the final value is the color of that particular pixel. This is called ray marching, as shown in Figure 3.1.

To produce realistic underwater godrays, the renderer should take in to account the physical properties of the actual natural phenomena, which include characteristics of light and properties of

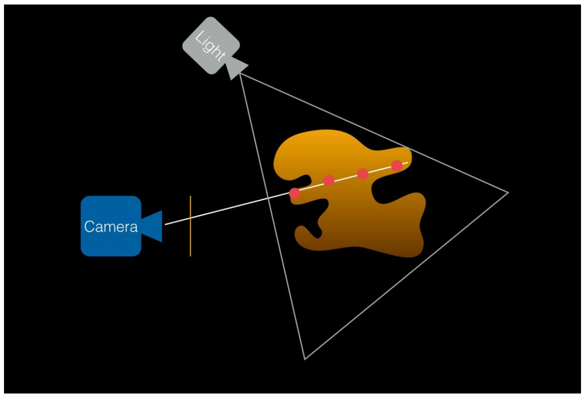


Figure 3.1: Ray Marching Technique

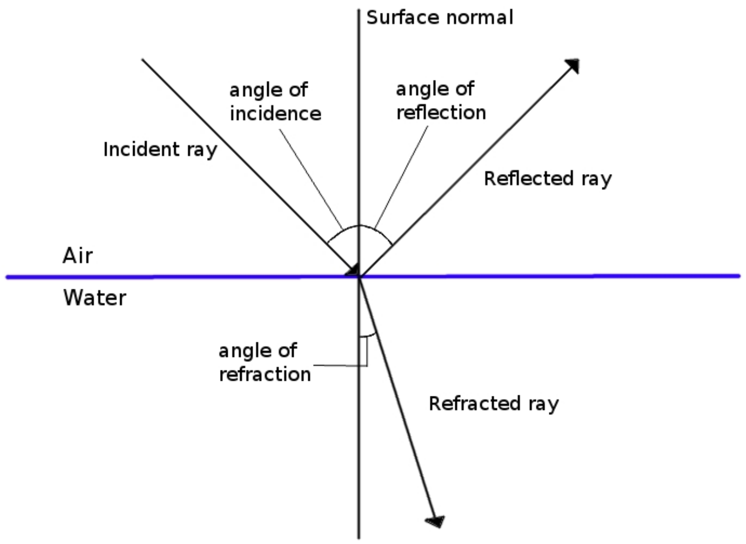


Figure 3.2: Reflection and Refraction of a light ray

water. A light ray can travel from one medium to another medium. The basic characteristic of a light ray is that it is reflected and refracted when it hits an interface. When a light ray travels from one medium to another medium with different refractive indices, a certain fraction of light is reflected and the remaining light is refracted into the next medium. Occurrences of this phenomenon can easily be observed when light enters and leaves some sort of transparent medium. The angle of incidence is the angle between the incident ray and the surface normal, while the angle of refraction is measured between the surface normal and the refracted ray. The different parameters when a light reflects and refracts on a surface are illustrated in Figure 3.2. For this project, the light source is a directional light representing the sun, the first medium is air and the second medium is water. The index of refraction for air is one and for water is 1.33. The wave roughened water surface has the effect on light below the surface of creating visible rays or shafts of light called godrays.

The amount of light reflected back to the atmosphere and the amount of light refracted into the ocean can be determined using Fresnel's equation. The relationship between the incident angle and the transmitted angle is given by Snell's law, which states that the ratio of the sines of incident angle and transmitted angle is equal to the ratio of the opposite refractive indices of the medium. Snell's law is represented as:

$$\sin\theta_t = \left(\frac{n_{air}}{n_{water}}\right) \times \sin\theta_i \tag{3.1}$$

where, θ_i is angle of incidence, θ_t is angle of refraction, n_air is refractive index of air, n_water is refractive index of water.

To render underwater light shafts, a triangulated ocean surface mesh with waves, a directional light source, and a camera are given as inputs to the volume renderer (Figure 3.3). The ocean surface mesh was generated in maya using a tool created by Samuel Bryce, which uses FFT waves based on Dr. Tessendorf's technique [15]. The tool conveniently allowed parameter variations such as the resolution of the surface, the shortest and the longest wave, and the cusp factor as inputs to FFT waves. In all elements that can be rendered as volumes, the density of the volume affects the transitivity of the light, contributing to the apparent depth of the 2D image. Hence, the density of volumes are generally stored in a 3D grid, and a ray marching technique is used to render the images. For rendering god rays, the camera lies under the ocean. As water is the volume, the density factor is constant and there is no need for 3D grids to store the data. Light rays originating from the light source are parallel rays above the ocean surface. The vertices of the triangles are considered to be the points of intersection on the ocean surface. When incident on the triangles, a certain amount of light is refracted through the triangle vertices and enters the ocean volume, as shown in Figure 3.4. The refracted rays from each vertex can be calculated using Snell's law. The refracted ray for a vertex on a triangle is given by the equation

$$\hat{d}_t = \hat{d}_i \; n_r + \hat{n} \; B \tag{3.2}$$

where, $\hat{d}_\mathrm{t}$ is refracted ray from the vertex, $\hat{d}_\mathrm{i}$ is incident light ray from the parallel light source and $\hat{n}$ is normal of the triangle vertex.

$$\begin{aligned} B &= -n_r \times \cos\theta_i \pm \left(\frac{\sqrt{1 - n_r^2 + n_r^2 \times \cos^2\theta_i}}{\sqrt{1 - n_r^2 \times \sin^2\theta_i}}\right) \\ n_r &= \left(\frac{n_{air}}{n_{water}}\right) \end{aligned}$$

where, n_air is refractive index of air and n_water is refractive index of water.

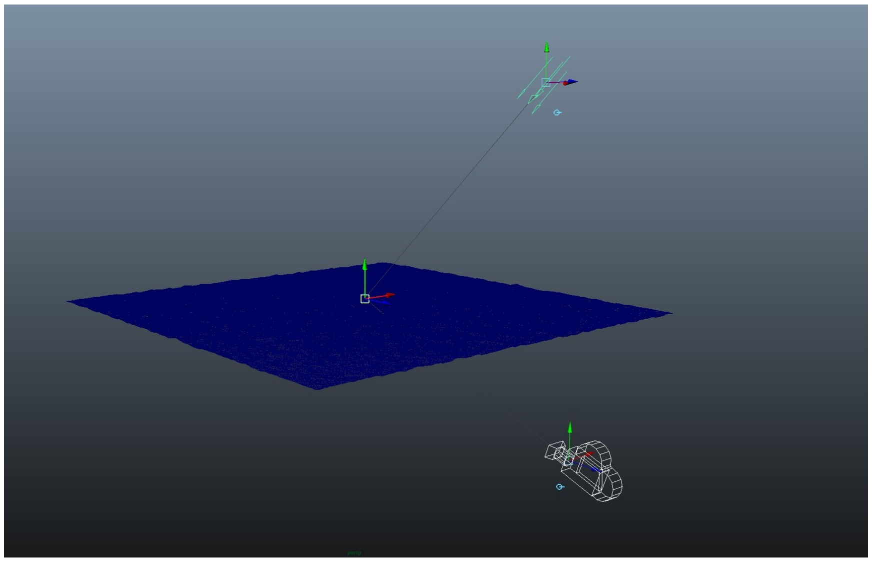


Figure 3.3: Maya scene with the ocean surface, light and the camera which are the inputs for the renderer

For each triangle in the ocean surface mesh, three refracted rays are calculated. Similar to a technique used by Watt [17], a beam of light is projected from each triangle, with the refracted rays as edges of the three sides shaft. Each side surface of the shaft is a bilinear patch, becuase the points on the refracted rays used to form the shaft sides are non-coplanar points. While it is simple to calculate an intersection between a ray and a planar surface, finding an intersection between a ray and a bilinear patch involves much more computational complexity. In order to render the image, a ray is generated from the camera for each pixel. The equation of the ray is given as:

$$\vec{X}\,(r) = \vec{X}_{cam} + \hat{n}_p\ r \tag{3.3}$$

where, $\vec{X}_{\mathrm{cam}}$ is the position of the camera and $\hat{n}_{\mathrm{p}}$ is the direction of the pixel.

If the ray from the camera hits the projection of the light shaft, the intersection points are calculated and a triangle is projected which contains the intersection point. Now, there will be two triangles and the bottom projected triangle is usually called a caustic polygon while the actual ocean surface triangle is called a specular polygon. The caustic volume is the volume bounded by the specular and caustic polygons [11]. Each side of the caustic volume is defined by two points on the specular triangle and two points on the caustic triangle which forms a quadrilateral surface in three dimensional space as shown in Figure 3.5. If a pixel ray intersects the light shaft, there will be

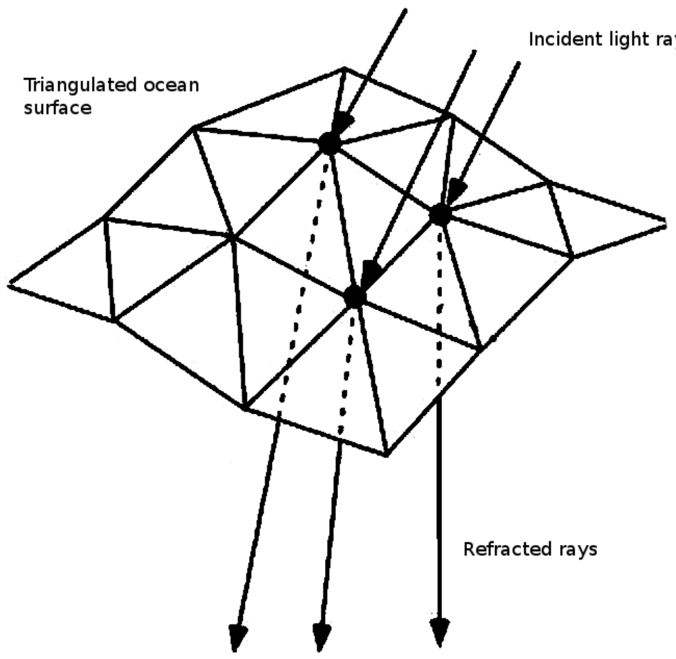


Figure 3.4: Refraction of light rays from trianlge vertices

two intersection points. There are four different cases for finding the intersection points. They are,

1. Two points on the caustic volume.
2. One point on the caustic volume and one point on the specular polygon.
3. One point on the caustic volume and one point on the camera.
4. One point on the specular polygon and one point on the camera.

The methods for calculating intersection points are different for each case, and will be discussed in detail in Chapter 4. After calculating the intersection points, the amount of light accumulated at those two intersection points are evaluated. As stated in Watt's method [17], the intensity of the caustic polygon is proportional to the area of the specular polygon divided by the area of the caustic polygon. It is calculated using the formula,

$$L_v = L_s \times \left(\frac{A_s}{A_c} \right) \tag{3.4}$$

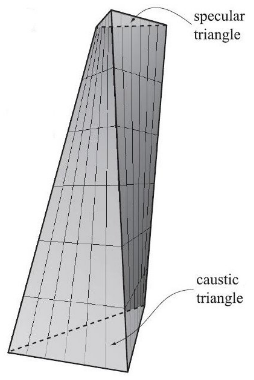


Figure 3.5: Representation of caustic volume with specular and caustic triangles

where L_v is the calculated light intensity, L_s is the sun intensity which is the intensity of light source, A_s is the area of the specular triangle and A_c is the area of the caustic triangle.

Similarly another caustic polygon is projected for the second intersection point and the light intensity is calculated. Once the light values at those two points are calculated, the color contribution from the triangle for that one pixel ray is calculated using the formula

$$Color = e^{-\rho \kappa r_1} \int_{r_1}^{r_2} dr \ L(r) \ \rho \ \kappa \ e^{-\int_{r_1}^{r_2} \ dr' \ \rho \ \kappa} \tag{3.5}$$

In order to get the volume render, the light intensity at the two intersection points is integrated between r_1 and r_2, producing the total light accumulated between those two points. Because the density is constant between these two points, ray marching is unnecessary and the integral can be evaluated analytically as:

$$Color = \left(\frac{r_2 - r_1}{M}\right) e^{-b(r_1+s_1)} \times \left[L_1\left(1 - e^{-bM}\right) - \left(L_2 - L_1\right) e^{-bM} + \frac{L_2 - L_1}{bM}\left(1 - e^{-bM}\right)\right] \tag{3.6}$$

where,

$$\begin{aligned} b &= \rho\kappa \\ M &= (r_2 - r_1 + s_2 - s_1) \end{aligned}$$

ρ is the density of water, κ is the transmisivity, r_1 is the distance between the camera and the first intersection point, r_2 is the distance between the camera and the second intersection point, s_1 is the distance between the first intersection point and the closest edge in the specular triangle, s_2 is the distance between the second intersection point and the closest edge in the specular triangle.

Scattering and absorption of light in the water reduces visibility exponentially as the distance from the viewer increases. This attenuation is very color-dependent, and produces the bias toward a blue-green color. In the rendering equation, the attenuation factor b is responsible for reduction of visibility. The factor b is a color component and has three different values for the RGB channels. The value for this parameter was provided by the *Peanut Butter Jelly* project [9], as 0.0758 for the red channel, 0.02197 for the green channel and 0.0025 for the blue channel. As the value of the red channel is higher, the contribution of red color will be less in the final color calculation.

Once the color is calculated for that triangle, the next triangle from the ocean surface is selected and the process repeated. When all the ocean surface triangles are checked for intersection for that one pixel ray, the accumulated color values are the color of that pixel. After calculating color values for all of the pixels, they are stored as an image.

This renderer also allows storing the image as a deep image. Deep images have a format that stores RGBA values at the actual distance from the camera. The advantage of using deep images is that two images rendered seperately without holdout mattes can be composited with proper holdouts into a single image with the help of the depth information. OpenEXR 2.0, [6] includes the deep image format used in this renderer. The values r_1 and r_2 calculated for each triangle after the intersection test are the depth values provided to the deep image.

To apply this renderer for the *Peanut Butter Jelly* production, a tool was needed to provide a user interface. A tool for Maya 2012 was created, with purpose of allowing any artist working in the production to setup a scene and render god rays. The tool allowed the user to change and specify the input parameters needed by the command line renderer. It also managed the rendered images so that the assets are stored in a directory conforming to the DPA pipeline. The description of the Maya tool is given in Chapter 4.

Chapter 4

Implementation

The implementation steps involved in rendering god rays are as follows:

1. Read the ocean surface obj file.
2. Calculate refracted rays for each vertex of a triangle based on parallel rays from source.
3. For a pixel ray, perform the intersection test with the triangle.
4. Calculate light intensity at the intersection points.
5. Compute color value contributed by that triangle and accumulate.
6. Repeat the process with the next triangle.
7. Move to the next pixel and repeat step 3–6.

The triangulated ocean surface mesh is one of the inputs for the renderer. An obj file parser, written in C++, reads the ocean surface obj file and stores the vertices and normals. For each triangle, the edges are also calculated and stored. Using equation 3.2, refracted rays are computed for each vertex. A caustic volume can be projected whose sides are formed with the refracted rays from three vertices of a triangle.

After calculating the refracted rays, a ray is emitted from the camera along the direction of each pixel. An intersection test is made in order to find whether the pixel ray hits the caustic voulme or the ocean surface triangle. Any point on the caustic volume which intersects with the pixel ray will satisfy the equation,

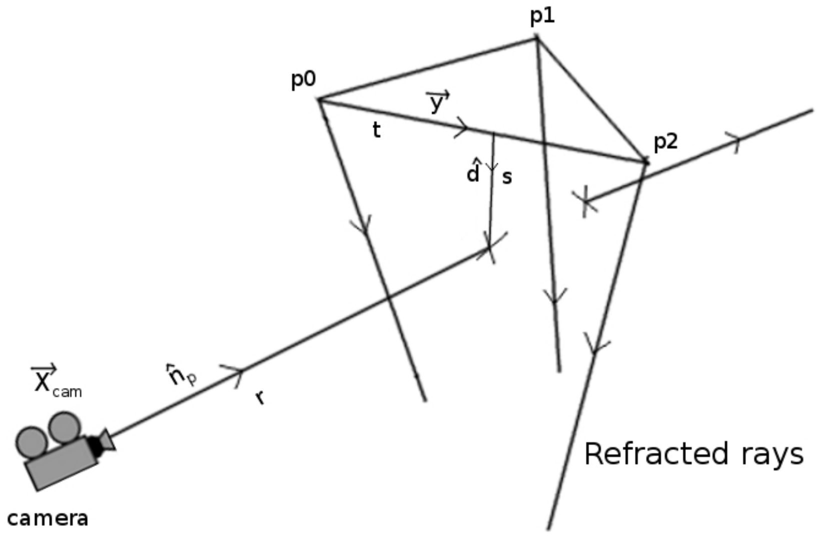


Figure 4.1: r,s and t values on a caustic volume

$$\vec{X}_{cam} + \hat{n}_p r - \vec{y}(t) + \hat{d}(t)\ s = 0 \tag{4.1}$$

where, $\vec{X}_{cam}$ is camera position, $\hat{n}_p$ is direction of pixel ray, $\vec{y}(t)$ is a point on the edge of the specular triangle, r is the distance from camera to the intersection point, $r > 0$, s is the closest distance from the intersection point to the triangle edge, $s > 0$, $\hat{d}(t)$ is the direction of $\vec{y}(t)$, and t is a barycentric coordinate of a point on the edge.

To solve the above equation, the values of r, s and t must be calculated. The values of r and s can be obtained in terms of the variable t as below.

$$\begin{aligned} r &= \frac{\hat{n}_p \cdot \left(\vec{y}\,(t) - \vec{X}_{cam}\right) - \left(\hat{n}_p \cdot \hat{d}\,(t)\right) \hat{d}\,(t) \cdot \left(\vec{y}\,(t) - \vec{X}_{cam}\right)}{1 - \left(\hat{n}_p \cdot \hat{d}\,(t)\right)^2} \\ s &= \frac{\left(\hat{n}_p \cdot \hat{d}\,(t)\right) \hat{n}_p \cdot \left(\vec{y}\,(t) - \vec{X}_{cam}\right) - \hat{d}\,(t) \cdot \left(\vec{y}\,(t) - \vec{X}_{cam}\right)}{1 - \left(\hat{n}_p \cdot \hat{d}\,(t)\right)^2} \end{aligned} \tag{4.2}$$

(4.3)

In order to find the values of r and s, the value of t has to be calculated simultaneously. The solution for the parameter t is obtained using root finding. The equation to be solved in order to find the value of t is,

$$f(t) = \left(\vec{y}(t) - \vec{X}_{cam}\right) \cdot \left(\hat{n}_p \times \hat{d}\,(t)\right) = 0 \tag{4.4}$$

where, $\vec{y}(t)$ is a point on the edge of the specular triangle, $\vec{X_{cam}}$ is the position of the camera, $\hat{n_p}$ is direction of pixel ray and $\hat{d_t}$ is the direction of y(t).

The secant method [5] was used in order to find the solution for t. It is based on approximating the function by secant lines [4]. For this approach, two initial approximations are taken in to account assuming they are close to the solution. As the value of t determines a point on the edge of the triangle, the two inital values are taken as the two vertices on either side of the edge. After the first iteration, the result from the previous iteration is used as the approximation for the next iteration. The process is repeated until the value of t converges on an accurate answer. If there is no solution which satisfies the equation, then iteration will not converge and there is no intersection point, which means the light shaft from that triangle is not in the path of the pixel ray. If there is a solution, the values of r and s are obtained by substituting the value of t in equations 4.2 and 4.3 respectively. There has to be two intersection points in order to calculate the color of the pixel.

As discussed in chapter 3, there are four different cases to be considered when doing the intersection test. In the first case, both the intersection points can be on the caustic volume (Figure 4.1). The second intersection point can be calculated using the same approach discussed above on one of the other two sides of the caustic volume formed from the other edges of the triangle. The second case to be considered is that the second intersection point can be on the specular triangle itself (Figure 4.2). The intersection point is calculated using a ray triangle intersection test. For this renderer, a fast, minimum storage ray/triangle intersection method was used [14]. If an intersection point is on the specular triangle, the value of t is zero and the light intensity at that point is equal to the intensity of the light source. This is because the specular triangle is the ocean surface and there will be no light decay on the surface.

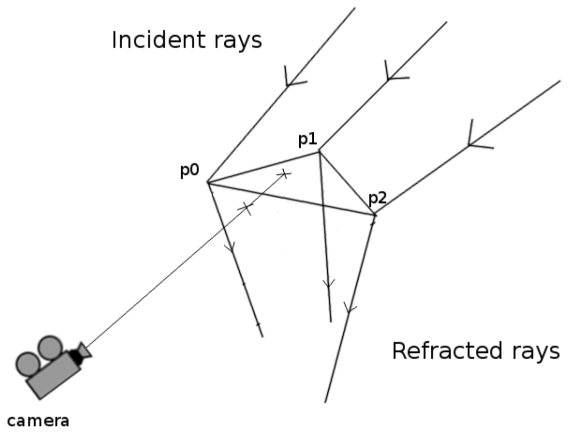


Figure 4.2: One intersection point on the caustic volume, another on the specular triangle

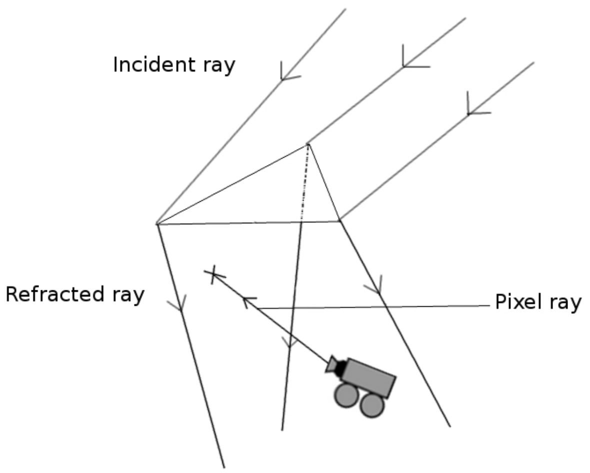


Figure 4.3: Only one intersection point on the caustic volume

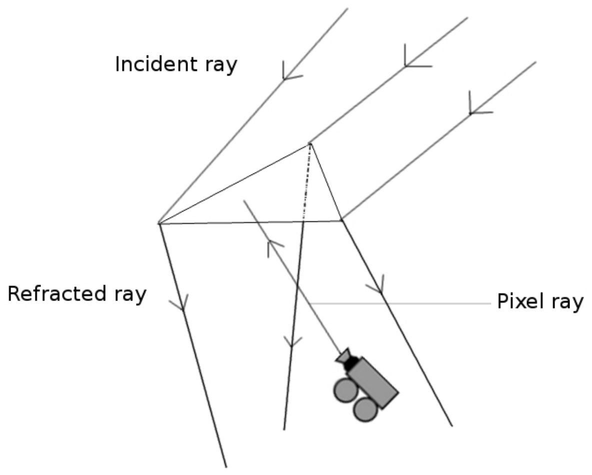


Figure 4.4: Only one intersection point on the specular triangle

There are certain cases in which there is only one intersection point, either on the caustic volume (Figure 4.3) or on the specular triangle (Figure 4.4). The first intersection point is considered to be the position of the camera and the second intersection point is the point found using intersection test. This happens when the camera is within a light shaft. If there is an intersection point, a caustic triangle can be formed. The intersection point is a point on the caustic triangle and the area is given by the formula,

$$A = |\vec{e_1}(s) \times \vec{e_2}(s) \cdot \vec{e_0}(s)| \tag{4.5}$$

where, $\vec{e}_0$, $\vec{e}_1$, and $\vec{e}_2$ are the edges of the specular triangle, s is the closest distance from the intersection point to the specular triangle edge. When the intersection point is on the caustic volume, the value of s is calculated using equation 4.3. When the intersection point is on the specular triangle, the value of s is zero. Thus the area of the caustic triangle and the specular triangle are the same. For case 3 and 4, when one of the intersection point is the camera itself, the value of s is calculated using the equation:

$$\vec{y} = \vec{X}_{cam} - \vec{p}_0 - u\vec{e}_1 - v\vec{e}_2 \tag{4.6}$$

where, $\vec{y}$ is a vector from the position of the camera to the specular triangle and the magnitude of $\vec{y}$ is the value of s which is the closest distance between the camera and the specular triangle. $\vec{p}_0$ is a vertex and $\vec{e}_1, \vec{e}_2$ are edges of the specular triangle. u and v are barycentric coordinates and they must satisfy three conditions. They are,

1. $0 \leq \mathrm{u} \geq 1$
2. $0 \leq \mathrm{v} \geq 1$
3. $0 \leq \mathrm{u+v} \geq 1$

If any one of the above condition fails, the value of s can be calculated using next method based on equation:

$$\vec{y} = \vec{X}_{cam} - \vec{p}_i - m\vec{e}_i \tag{4.7}$$

where,

$$m = \frac{\left(\vec{X}_{cam} - \vec{p}_i\right) \cdot \vec{e}_i}{|\vec{e}_i|^2} \tag{4.8}$$

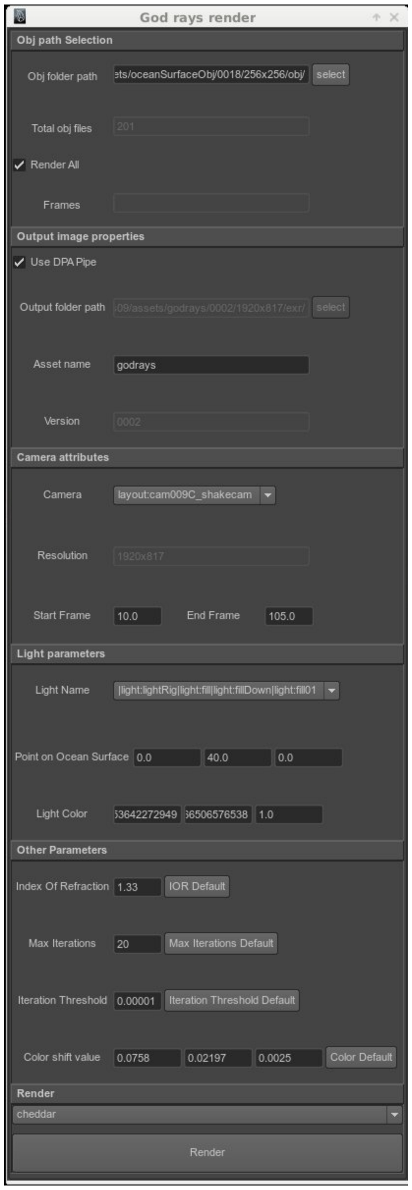
$\vec{p}_{\mathrm{i}}$ is a vertex and $\vec{e}_{\mathrm{i}}$ is an edge on the specular triangle. The value of m is calculated for all three vertices and the corresponding edges of the specular triangle and substitute in equation 4.7. The absolute minimum value of equation 4.7 of all three values provided m satisfies the condition $0 \leq m \geq 1$, is the value of s. If m value does not satisfy the condition for all three vertices, then the minimum value of the distance from the camera to one of the vertex of the specular triangle is value of s. The area of the caustic triangle is calculated by substituting the value of s in equation 4.5.

After calculating the area of the caustic triangle, the light intensity value at the intersection point can be calculated by dividing the area of specular triangle by the above result. This is represented as L_1 for first intersection point and L_2 is computed in a same way for the second intersection point. Using the light intensity value at the two intersection points and the distance from camera to the points, the color value can be computed using the equation 3.6 in chapter 3. Once the color calculations are computed for one triangle, the same process is repeated for the other triangles in the mesh. If there is color contribution from any other triangle for that pixel ray, the color values from each triangle are added together unless its a deep image. The process is reproduced for the next pixel and after repeating the procedure for all pixels, the values are stored as an image.

The renderer also provides an option for the user to render the god rays as a deep image. The deep image format uses deep points in addition to pixels. For storing the image as a deep image, the depth information has to be stored in the image along with the red, blue, green and alpha channel. A deep image C++ class provided by DPA student Kacey Coley was used to render deep images. As the program has already computed the distance r from the camera to the intersection point, the value is stored as the depth for each deep point. The difference in procedure between the 2D rendering and deep rendering is that there is no need for accumulating color for a single pixel when creating a deep image.

The volume renderer was written in C++. As *Peanut Butter Jelly* was a team production, anyone who was working on the short would be able to render godrays. A maya tool with a graphics user interface(GUI) allow users to manipulate the input parameters. The user interface is shown in Figure 4.5.

The first text box allows the user to select the directory which has the ocean surface obj files. The output directory points to the asset location where the final images are rendered and stored. If the useDPAPipe option is checked, the tool creates the asset location following the DPA pipeline. The string in the asset name text box will be the name of the directory and the base name for the frames followed by version and frame number. In the camera attributes section, the camera text box lists all the available cameras in the scene as a drop down list with the render camera as default. As the camera can be animated in Maya, the renderer needs to keep track of the position, aim vector and up vector of the camera for each frame. The maya tool for the renderer gets the attributes of the rendering camera in maya for each frame and matches it with the camera attributes present in the renderer. The tool also gets the image resolution and the frame range from render settings. It also lists all the lights available in the scene. When a light is selected, the position and color of the light are obtained from its' property menu. In the Other parameters section, the variable index of refraction (IOR) is the value for the second medium, which has the default value 1.33 for water. The parameter max iterations is the maximum number of iterations used for root finding to find the solution for variable t and the color shift value is the attenuation factor b used in color calculation. A sequence of obj files were used for water surface and for each frame an obj file is read. Scripts are generated for each frame and the jobs registered with the queue similar to that for other types of render tasks.


God rays render

Obj path Selection

Obj folder path ets/oceanSurfaceObj/0018/256x256/obj/ select

Total obj files 201

✓ Render All

Frames

Output image properties

✓ Use DPA Pipe

Output folder path ‹09/assets/godrays/0002/1920x817/exr/ select

Asset name godrays

Version 0002

Camera attributes

Camera layout:cam009C_shakecam

Resolution 1920x817

Start Frame 10.0 End Frame 105.0

Light parameters

Light Name |light:lightRig|light:fill|light:fillDown|light:fill01

Point on Ocean Surface 0.0 40.0 0.0

Light Color 53642272949 36506576538 1.0

Other Parameters

Index Of Refraction 1.33 IOR Default

Max Iterations 20 Max Iterations Default

Iteration Threshold 0.00001 Iteration Threshold Default

Color shift value 0.0758 0.02197 0.0025 Color Default

Render

cheddar

Render

Figure 4.5: Maya user interface

Chapter 5

Peanut Butter Jelly Results

The motivation to build this custom volume render was to generate god rays in support of the production of *Peanut Butter Jelly*. High quality visuals of godrays help to establish an authentic underwater environment. A rendered frame of god rays, color corrected in nuke is shown in Figure 5.2. The god rays generated using the custom renderer has fine and varying structures. The intensity of the color has a realistic gradient effect from the top of the image to the bottom as the intensity of the light decreases with depth. The quality of light shafts depends on size of the specular triangle. The structure of god rays can be sharp when there are many smaller triangles on the ocean surface mesh. The sides of the caustic volume are bilinear patches and the intersection test on each patch involves high computational complexity. Thus the quality and the rendering time depends on the number of triangles present in the ocean surface mesh. In order to generate high quality renders of god rays, the ocean surface mesh used for the production was of high quality, with more than 130000 triangles. The time to render a single frame on an 8 core machine varied from $18 - 21$ hours. *Peanut Butter Jelly* had 18 shots with the total frame count of 1211 and the god rays volume renderer was used in all of the shots. As the resources present in DPA lab were not sufficient to render god rays for all the frames within the production time, the Clemson University Palmetto Linux cluster was used for the rendering process. The Palmetto cluster is Clemson University's primary high-performance computing (HPC) resource [3], which has 1978 compute nodes and 20,728 cores. The libraries necessary for the custom volume render were installed on Palmetto along with the renderer C++ code. Pbs scripts were generated using the maya tool, transferred to the Palmetto cluster and submitted to the queue. Upon completion, the frames were transferred from Palmetto to the DPA pipeline assets. More than 42 years were spent on rendering god rays on Palmetto. The algorithm can be improved in the future by using a bounding volume hierarchy (bvh) [1] technique for intersection tests to decrease the rendering time.

During intial test cases, there were lot of distracting artifacts when the light shafts were very close to the camera. A sample image which has artifacts is shown in Figure 5.1. By setting the near plane of the camera to a distance not close to the camera, the artifacts could be eliminated with acceptible image quality. For *Peanut Butter Jelly*, the near plane parameter is a user input for the command line renderer.

Even though the renderer was built for the *Peanut Butter Jelly* production, it was incorporated into the DPA pipeline so that it can be reused in future productions. The results produced by combining god rays into the final image made drastic differences in the look of the shot. The final images of shot 01, shot 02 and shot 17 of the *Peanut Butter Jelly* film are shown in Figure 5.3 5.4 and Figure 5.5.

The renderer is also capable of rendering god rays as a deep image. To render an image as a deep image, the depth information is stored along with the red, green, blue and alpha channels. The depth is stored as deep points in addition to the pixels which has the color. The file size of a single deep image is extremely large, as it stores the depth information and color value for every intersection point. The production did not use the deep image feature as the 2D image satisified the production need. Figure 5.6 is a perspective view in nuke of a frame in which godrays are rendered as deep image.

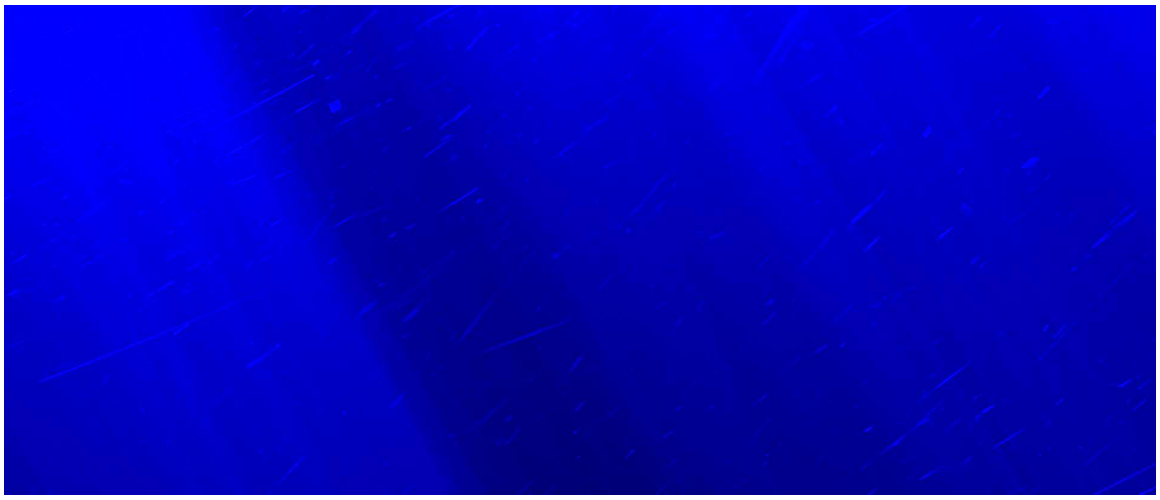


Figure 5.1: God rays with artifacts

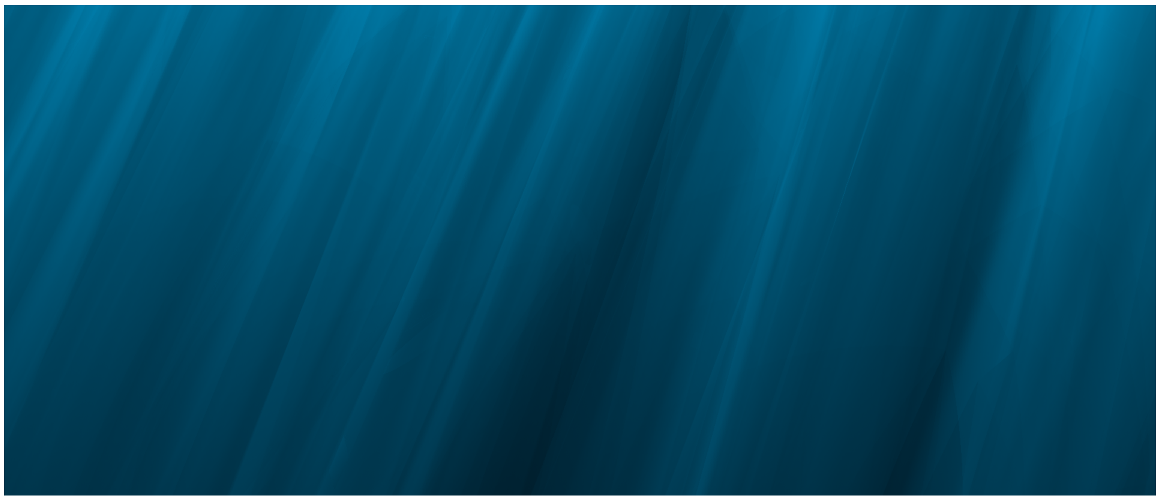


Figure 5.2: God rays color corrected in nuke

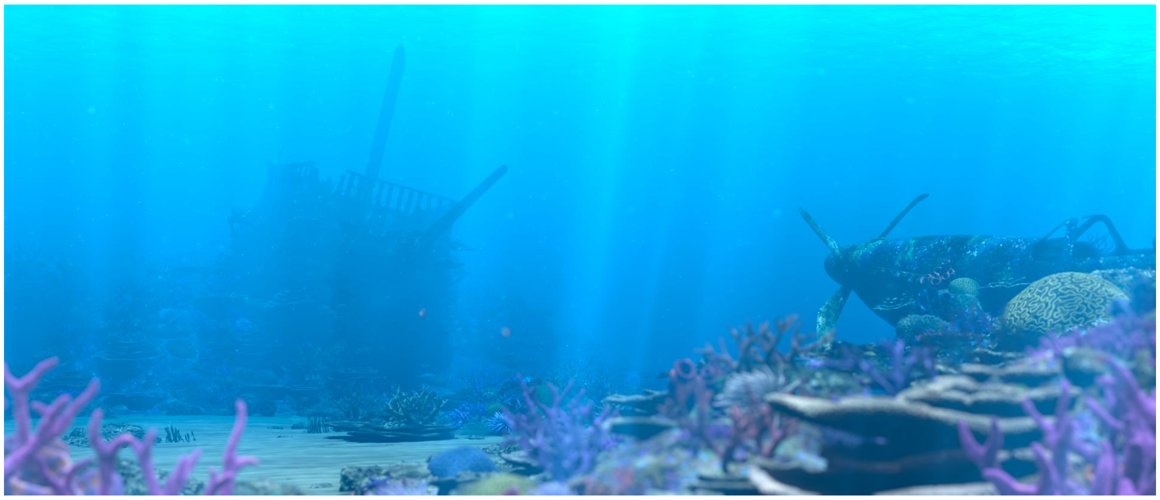


Figure 5.3: Shot 01 final image with god rays

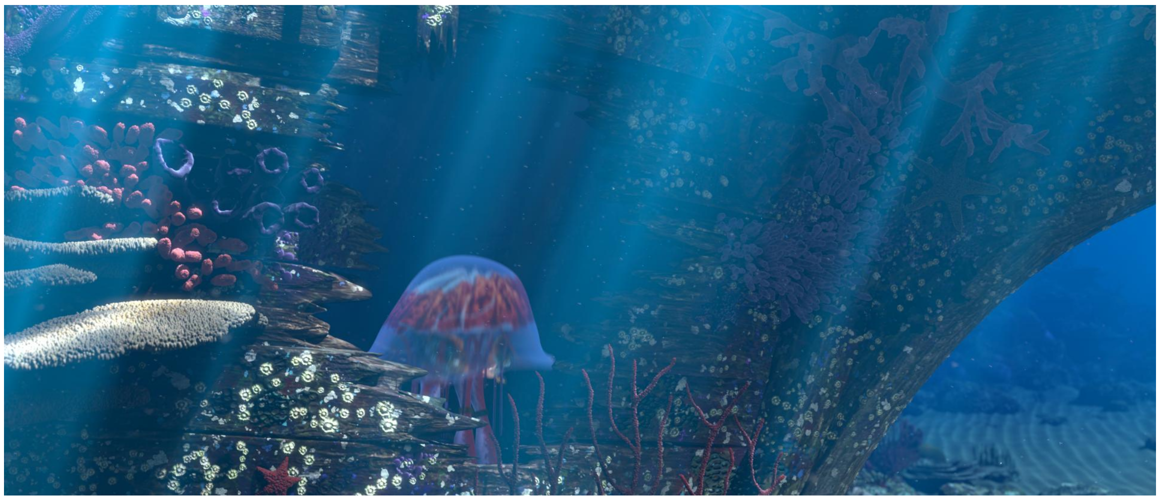


Figure 5.4: Shot 02 final image with god rays

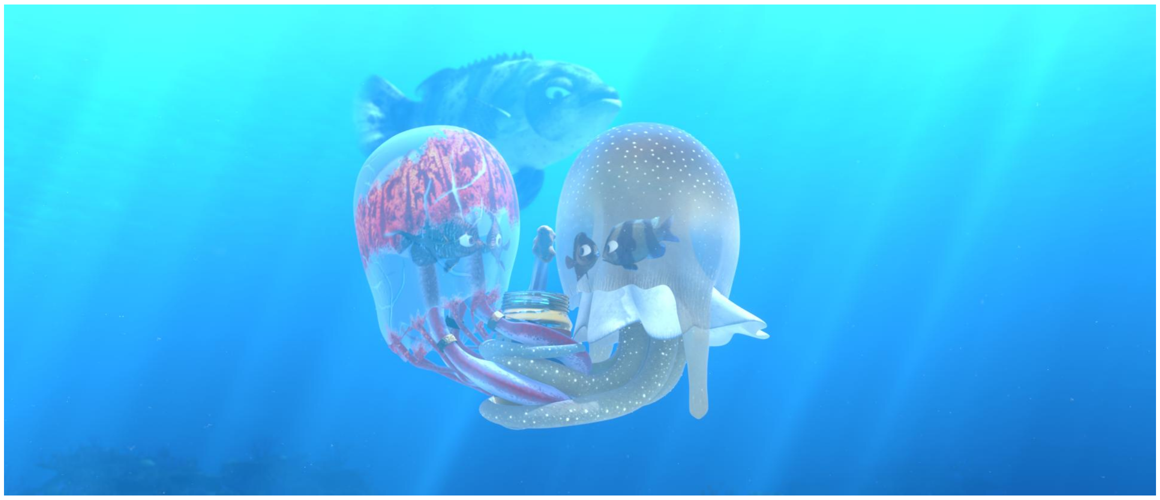


Figure 5.5: Shot 17 final image with god rays

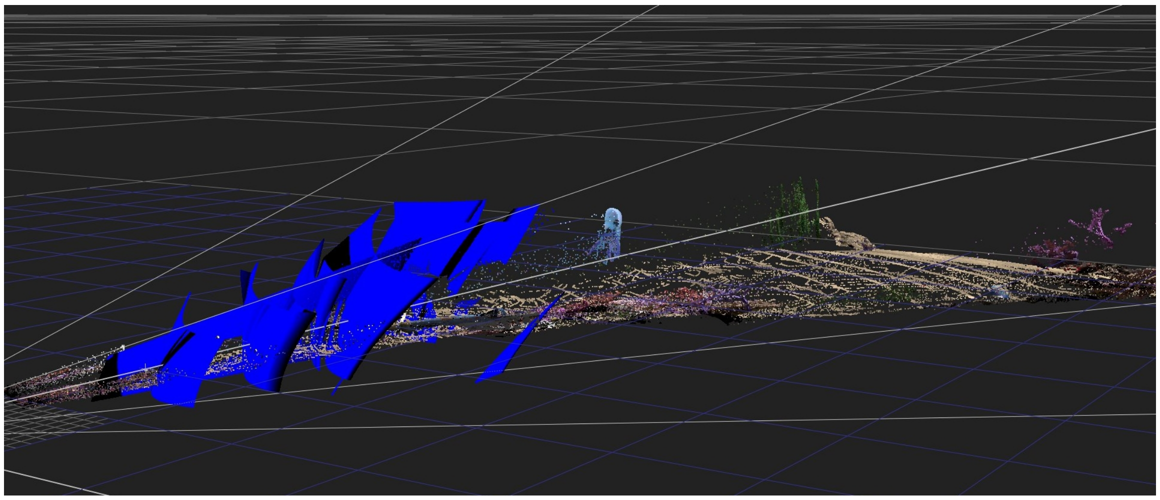


Figure 5.6: Godrays rendered as deep image

Bibliography

[1] Bounding volume hierarchy.
http://en.wikipedia.org/wiki/Bounding_volume_hierarchy.html.

[2] Corals at the apo islands, phillipines.
https://www.google.com/earth.html.

[3] Palmetto cluster.
http://citi.clemson.edu/palmetto.html.

[4] Secant lines.
http://en.wikipedia.org/wiki/Secant_line.html.

[5] Secant method.
http://en.wikipedia.org/wiki/Secant_method.html.

[6] Openexr 2.0.
http://www.openexr.com/, 2012.

[7] Writing atmosphere and interior shaders.
http://renderman.pixar.com/resources/current/rps/atmosphereAndInteriorShaders.html, March, 1997.

[8] crepuscular rays.
http://epod.usra.edu/blog/2008/04/lincoln-park-rays.html, March 3, 2006.

[9] Alexander Beaty. Diffuse attenuation.
http://wiki.fx.clemson.edu/mediawiki/index.php/PeanutButterJelly#DiffuseAttenuation.html, 2014.

[10] Enric Adrian Gener. Underwater god rays.
http://photography.nationalgeographic.com/photography/photo-of-the-day/mediterranean-sea-swimming.html, February 11, 2011.

[11] Carsten Dachsbacher Gabor Liktor. Real-time volume caustics with adaptive beam tracing. *Proceeding I3D*, pages 47–54, 2011.

[12] S. Lanza. Animation and rendering of underwater godrays. In *SHADERX 5*, pages 315–327. Charles River Media, 2007.

[13] Levoy. Efficient ray tracing of volume data. *ACM Transactions on Graphics*, 9(3):245–261, 1990.

[14] Tomas Möller and Ben Trumbore. Fast, minimum storage ray-triangle intersection. *Journal of graphics tools*, 2(1):21–28, 1997.

[15] Jerry Tessendorf. Simulating ocean water. 1999.

[16] Jerry Tessendorf and M Kowalski. Resolution independent volumes. In *ACM SIGGRAPH 2010 Courses*, 2010.

[17] Mark Watt. Light-water interaction using backward beam tracing. In *Computer Graphics (SIGGRAPH 90)*, pages 377–385. ACM Press, 1990.