5-2015

Surfacing Jellyfish for Peanut Butter Jelly

Brianne Campbell
Clemson University

Surfacing Jellyfish for *Peanut Butter Jelly*

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
Brianne Campbell
May 2015

Accepted by:
Dr. Timothy Davis, Committee Chair
Dr. Jerry Tessendorf
Dr. Kelly Caine

Abstract

This thesis discusses the surfacing techniques and process used in the creation of two main characters and the supporting crowd characters in the short animation, *Peanut Butter Jelly*. The goal was to create a custom RenderMan shader that could be used for both characters, and that had qualities of refraction and subsurface scattering, to allow the characters to appear soft and translucent like jellyfish. Shader writing in RenderMan, map painting in Mari, and design decisions are covered to explain the way the goal was achieved. The process for creating an interesting secondary character that cooperated seamlessly with the main jellyfish to form its eyes is also discussed. Ultimately, these visually cohesive and engaging characters lend to the overall success of the final renders.

Acknowledgments

I would like to thank the Lord for His favor, guidance and love in my pursuit of finding the place for the gifts and talents He bestowed upon me. To my family and dearest friends, thank you for your love, encouragement and support as I follow my ever-expanding dreams that you believed in before I did. Thank you to my peers, who provided the constructive feedback that pushed me to new heights of creativity. Finally, I want to thank my committee members, Dr. Timothy Davis, Dr. Jerry Tessendorf, and Dr. Kelly Caine, for their dedication to my educational interests and consistent support during my time at Clemson University.

Table of Contents

Title Page . . . . . **i**

Abstract . . . . . **ii**

Acknowledgments . . . . . **iii**

List of Figures . . . . . **v**

1 Introduction **1**

2 Background **5**
2.1 SLIM Shader Language and Interface in Maya 5
2.2 Refraction 7
2.3 Subsurface Scattering 8
2.4 Ray Tracing Ray Information 9
2.5 Primitive Variables (primVars) 10
2.6 Secondary Outputs (AOVs) 10

3 Implementation, Workflow, and Processes **12**
3.1 Creating Fish as Jellyfish Eyes 12
3.2 Writing a Jellyfish Shader 18
3.3 Creating the Pirate Jellyfish Captain 19
3.4 Creating the Flyboy Jellyfish Captain 23
3.5 Creating the Jellyfish Gangs 24

4 Final Results **28**
4.1 Jellyfish Eyes 29
4.2 Pirate Jellyfish Captain and Pirate Jellyfish Crowd 30
4.3 Flyboy Jellyfish Captain and Flyboy Jellyfish Crowd 33
4.4 Additional Renders of Characters 34

5 Future Applications and Conclusion **37**
5.1 Future Applications 37
5.2 Summary and Conclusion 38

Appendix **39**
Jellyfish Shader Code 39

References **59**

List of Figures

1.1 17th Century English Merchant Vessel and Pirates [3] [1] . . . 2
1.2 P-51 Mustang and WWII Flyboy [10] [17] . . . 2
1.3 Jellyfish Eyes Visual Development . . . 3

2.1 Shader Interface in Maya . . . 6
2.2 Snell's Law of Refraction [5] . . . 7
2.3 Refraction of a Pencil in a Glass of Water [18] . . . 7
2.4 Theory of Subsurface Scattering [16] . . . 8
2.5 Subsurface Scattering in Milk Renders [4] . . . 9
2.6 Ray Depth Through a Sphere . . . 9
2.7 Secondary Outputs (AOVs) [11] . . . 10

3.1 Pirate and Flyboy Characters Visual Development . . . 13
3.2 Jellyfish Eyes Visual Development . . . 13
3.3 Potter's Angelfish [6] . . . 14
3.4 Left and Right Sides of Pirate Fish in Mari . . . 15
3.5 Pirate Fish Diffuse Maps . . . 15
3.6 Damsel Fish and Military Facepaint Inspiration [9] [19] . . . 16
3.7 Left and Right Sides of Flyboy Fish in Mari . . . 16
3.8 Flyboy Fish Diffuse Maps . . . 17
3.9 Pirate Jellyfish at Different Indices of Refraction . . . 18
3.10 Diffuse Map, Grayscale Map, Final Result . . . 20
3.11 Purple-Striped Jellyfish [7] . . . 20
3.12 Bandanna Before and After Color Corrections . . . 21
3.13 Bell Tentacles Before and After Surfacing Hides Geometry . . . 21
3.14 Brain and Tentacles Maps . . . 22
3.15 Pirate Ring Reference Images . . . 22
3.16 Pirate Rings . . . 23
3.17 White Spotted Jellyfish . . . 24
3.18 Flyboy Spots Map . . . 25
3.19 Polyps on Flyboy . . . 25
3.20 Gang Member Selection in Maya . . . 26
3.21 Pirate Gang Color Options . . . 27
3.22 Flyboy Gang Color Options . . . 27

4.1 *Peanut Butter Jelly* Shot 01 . . . 28
4.2 *Peanut Butter Jelly* Shot 02a . . . 29
4.3 *Peanut Butter Jelly* Shot 17a . . . 29
4.4 *Peanut Butter Jelly* Shot 02b . . . 30
4.5 *Peanut Butter Jelly* Shot 05 . . . 30
4.6 *Peanut Butter Jelly* Shot 12 . . . 31

4.7 *Peanut Butter Jelly* Shot 04 31
4.8 *Peanut Butter Jelly* Shot 09a 32
4.9 *Peanut Butter Jelly* Shot 14 32
4.10 *Peanut Butter Jelly* Shot 16 33
4.11 *Peanut Butter Jelly* Shot 10 33
4.12 *Peanut Butter Jelly* Shot 15 34
4.13 *Peanut Butter Jelly* Shot 06 34
4.14 *Peanut Butter Jelly* Shot 07 35
4.15 *Peanut Butter Jelly* Shot 09b 35
4.16 *Peanut Butter Jelly* Shot 13 36
4.17 *Peanut Butter Jelly* Shot 17b 36

Chapter 1

Introduction

This thesis focuses on the surfacing of the main characters for the production of the animated short, *Peanut Butter Jelly*. The concept of the production was to create a photo-realistic, yet stylized, underwater feature in which two rival groups of jellyfish engage in an epic battle over a coveted peanut butter jar. Along with the challenges that come with animating, lighting and compositing an underwater environment, a unique set of challenges arise specifically when surfacing two main jellyfish characters and their corresponding crowd characters.

During the visual development of the production, the two jellyfish groups were given general themes based on the wreckages they inhabited within the environment. To form distinct jellyfish groups, two drastically different types of wreckages from two different time periods were selected. The first wreckage was an English merchant vessel from the 17th Century, which is reminiscent of a classic sunken pirate ship. This ship inspired the concept for the first group of jellyfish to adopt the characteristics of classic pirates (Figure 1.1). The second wreckage was a World War II P-51 Mustang long-range, single-seat fighter plane. The men who piloted these planes were known as "flyboys," which inspired the look for the second group of jellyfish (Figure 1.2).

Visual development also revealed that creating eyes for the jellyfish would not be a simple task. Based on feedback given by an industry professional from DreamWorks, the eyes were not modeled directly onto the model since real jellyfish do not have eyes, and implementing such an approach would result in a cartoon-like character, which is contrary to the partially photo-realistic goal of the production. The resulting plan for the eyes, therefore, was to create a secondary character that could function independently from the jellyfish, but also form a symbiotic relationship with it

Figure 1.1: 17th Century English Merchant Vessel and Pirates [3] [1]

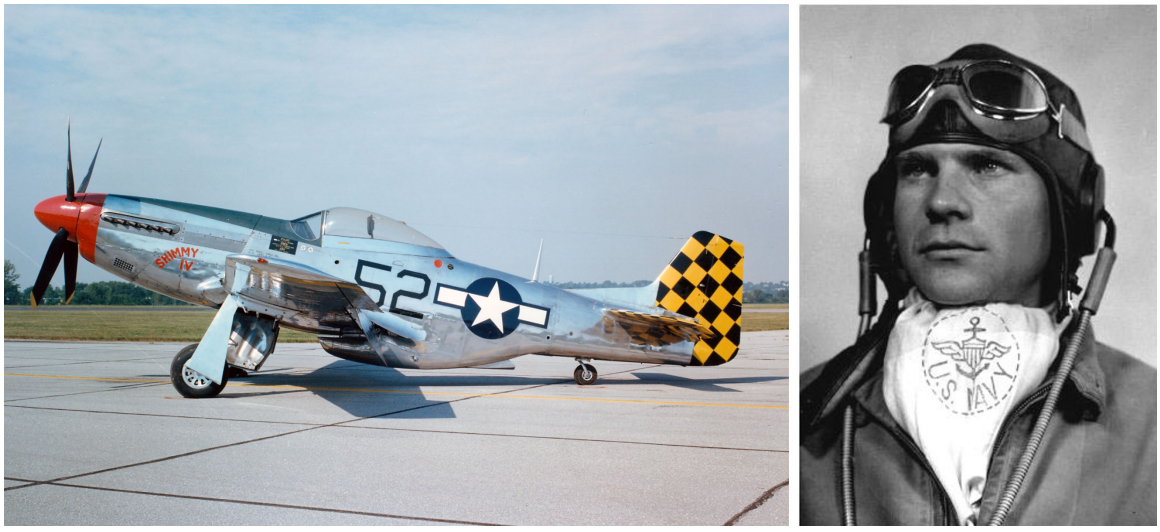

Figure 1.2: P-51 Mustang and WWII Flyboy [10] [17]

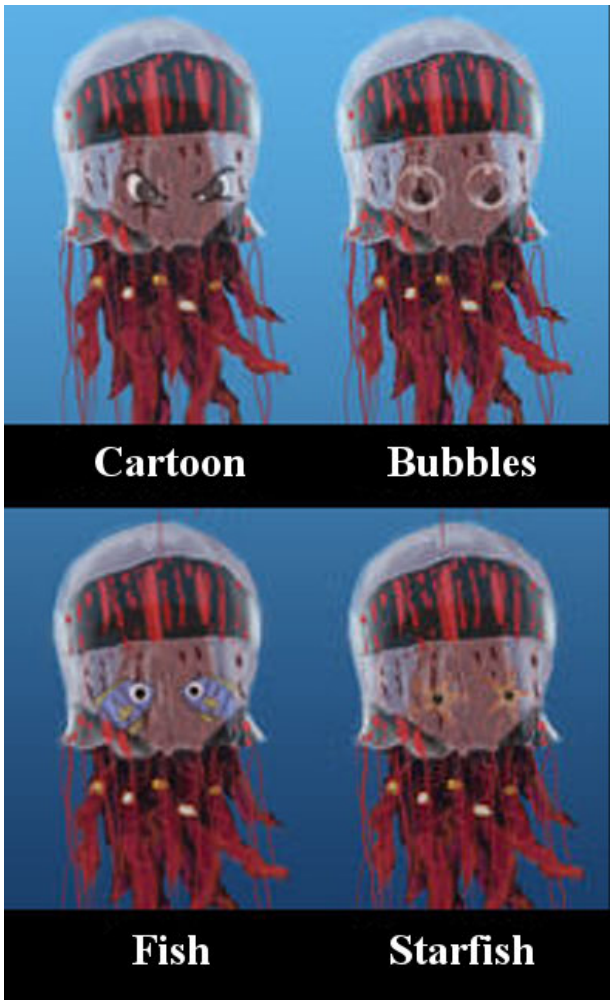

Figure 1.3: Jellyfish Eyes Visual Development

to form its eyes. After attempting a few different concepts for eyes that included bubbles, starfish and algae, a fish was selected as the best option for the secondary character (Figure 1.3).

Once the primary design features were established for the characters themselves, another challenge for surfacing was the partially translucent and refractive material, with brilliant saturated colors, of the jellyfish's exterior. Lastly, creating a shader that was versatile enough to use on both main characters was important for allowing the two groups to exhibit traits of the same species, but also easily distinguishable during the fast-paced action shots.

This thesis explains the way the aforementioned challenges were approached, and the process by which they were ultimately addressed. Chapter 2 discusses the specific software and methods

for the shader development itself. Chapter 3 explains the process for the surfacing design choices, textures used, and texture map created in Mari. Chapter 4 displays the resulting final renders of the fully surfaced characters in the environment. Finally, Chapter 5 discusses the application of techniques used for surfacing in this production for future work.

Chapter 2

Background

For the production of *Peanut Butter Jelly*, Pixar's RenderMan, in combination with Maya, was used because of its versatility, speed, and capability to implement advanced rendering features [13]. With RenderMan, a custom shader could be written with features that created the desired look of the jellyfish characters from a library of existing functions and techniques already available. Some of the features used included the SLIM shader language and Maya interface, refraction, subsurface scattering, ray information, primitive variables, and secondary outputs.

2.1 SLIM Shader Language and Interface in Maya

RenderMan shaders are written in the Slim file format with the extension ".sl," and are based on the TCL programming language [15]. Along with the ability to write a custom shader using this language, a custom interface can be created for Maya. Compiling the shader using the command "shader" and adding the flag "-C" allows the interface that organizes the variables of the shader within Maya to generate. The user of the shader can then make quick adjustments in Maya using the interface, rather than making changes to the code and recompiling. Some features that can be added to the interface include float values, value sliders, check boxes, texture map file browsers, and color pickers. Figure 2.1 shows parts of the shader interface in Maya for the jellyfish shader written for this production.

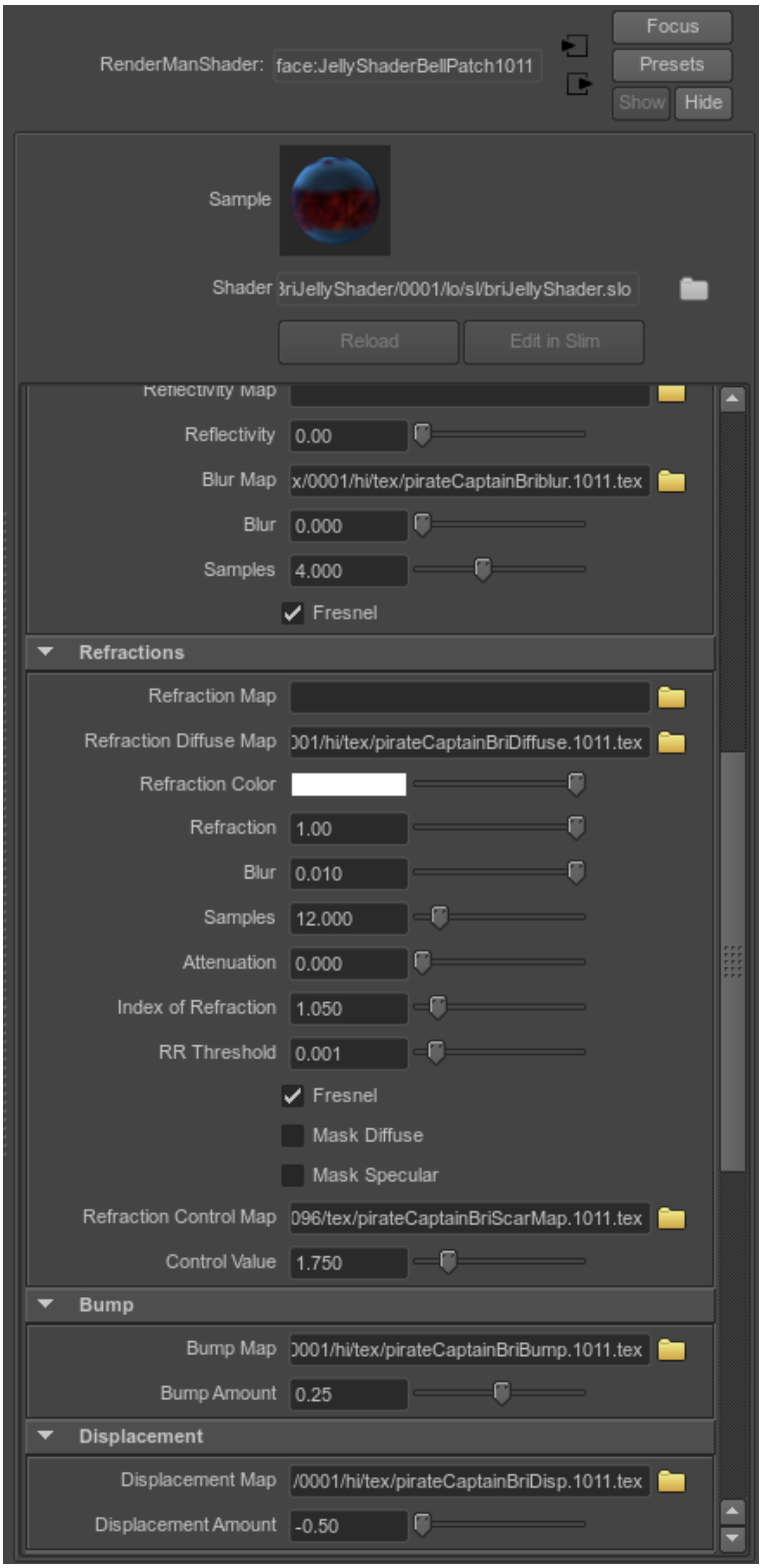

Figure 2.1: Shader Interface in Maya

2.2 Refraction

Refraction alters the direction of a light ray as it passes through one type of material to another type of material due to the different indices of refraction of those materials. To put it simply, if a light ray is passing through water in a glass, the angle of the light ray will change because glass and water have two different indices of refraction. Snell's Law explains the relationship between the angle of incidence (θ_1) and the angle of refraction (θ_2) (Figure 2.2).

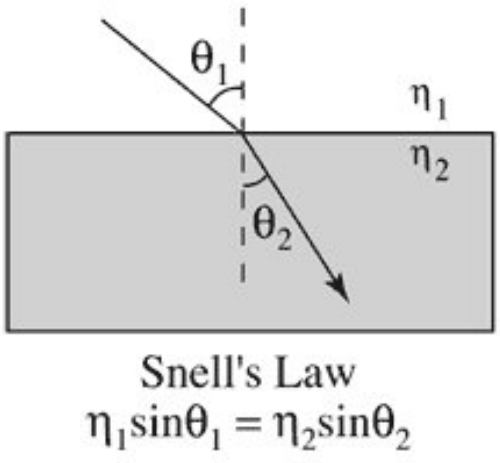

Figure 2.2: Snell's Law of Refraction [5]

Figure 2.3: Refraction of a Pencil in a Glass of Water [18]

Different materials have different indices of refraction, e.g., air is 1.0, water is 1.3, glass is 1.52 [2] (Figure 2.3). Due to these differences, refraction plays a large role in an underwater environment since light reacts differently in water than it does in air. Pixar's *Finding Nemo (2003)* used a combination of reflection and refraction to obtain the proper look of the fish tank in the film as light traveled through the air, glass and then water [5]. Although Pixar's technique for using refraction in the tank could not directly be applied to our entirely underwater environment, their use of physics to explain refraction was fundamental to the way we handled refraction in the jellyfish's bell.

2.3 Subsurface Scattering

Subsurface scattering is a rendering effect that simulates the way light is diffused inside a translucent medium [16]. Skin, wax, and in the case of the *Peanut Butter Jelly* production, coral and jellyfish, have subsurface scattering as a primary visual feature. The theory of subsurface scattering explains that when a photon hits a translucent surface, it bounces around inside and exits at a point it did not enter (Figure 2.4).

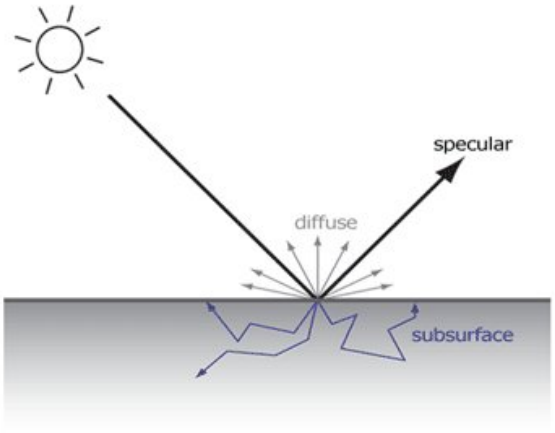

Figure 2.4: Theory of Subsurface Scattering [16]

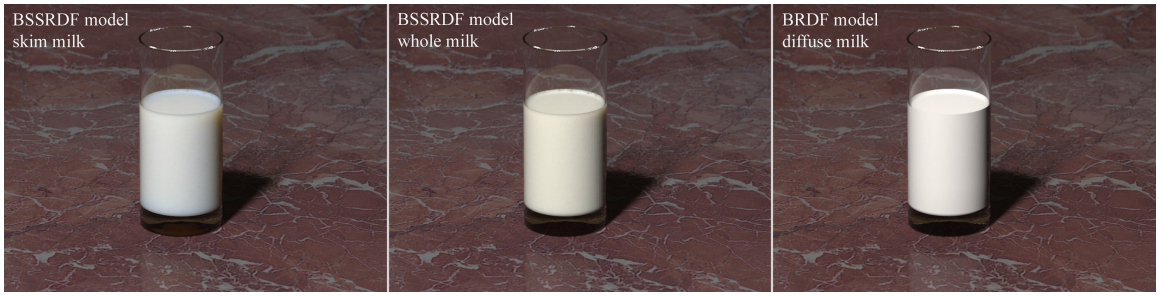

Figure 2.5: Subsurface Scattering in Milk Renders [4]

Subsurface scattering is an important feature for realistic rendering. As Figure 2.5 demonstrates, when rendering a translucent medium (in this case milk), subsurface scattering is necessary to show how light passes through it. The image on the far right shows a basic diffuse rendering of milk,which appears similar to a solid white cylinder. The images in the middle and on the left show different levels of subsurface scattering based on the density of the milk.

2.4 Ray Tracing Ray Information

Another feature available in RenderMan is the ability to use the function `rayinfo()` to provide information about the rays when the shader is called [8]. Some of the information that `rayinfo()` provides includes the ray depth, type, label, origin, direction, and length. The depth of a ray is particularly useful when determining the number of surfaces through which a ray has passed. The depth of a ray that has not passed through an object at all has a depth of zero, which increases as the ray passes through each additional surface (Figure 2.6).

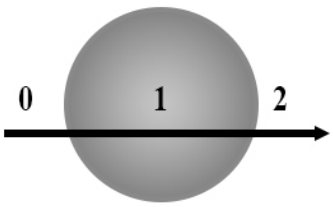

Figure 2.6: Ray Depth Through a Sphere

2.5 Primitive Variables (primVars)

Primitive variables, or primVars, are a mechanism used in RenderMan that allow the attachment of specific data to objects (primitives) within a scene that can be passed from an object to its shader at render time [12]. The power in using primVars lies in the ability to use one shader and overwrite values within it by changing the primVar value of the object. This feature is extremely advantageous when thousands of instances of the same object must be placed in a scene, as only one shader is required since the primVars can control color changes at render time. On a smaller scale, this approach also means that conditions can be created within a shader that will only activate on specified primitives.

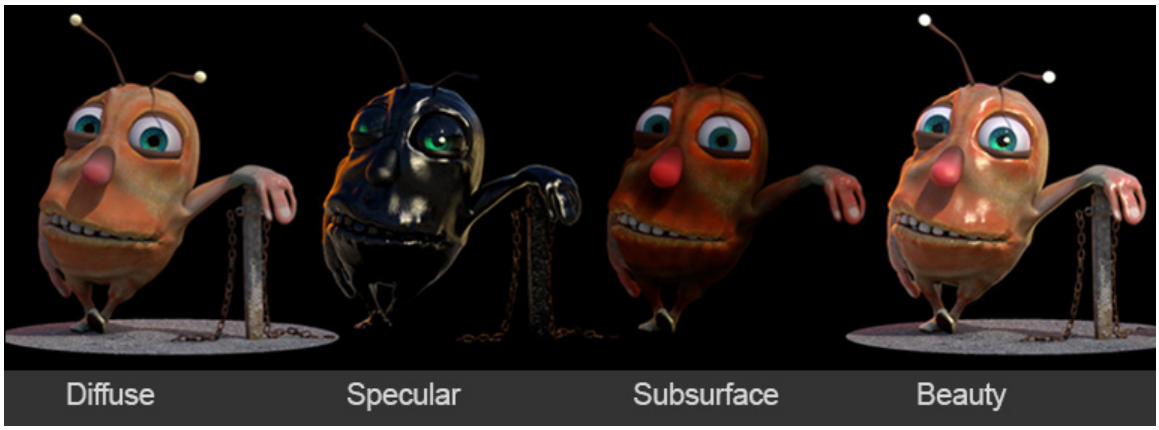

Figure 2.7: Secondary Outputs (AOVs) [11]

2.6 Secondary Outputs (AOVs)

Secondary outputs, known more commonly in RenderMan as arbitrary output variables, or AOVs, are image files that are composed of different components of the overall lighting equation [14]. Figure 2.7 shows an example of three commonly output AOVs and the final render, also referred to in RenderMan as the "Beauty" pass. RenderMan calculates these different components automatically for the final render; thus, no additional computation is necessary to output the data already available within the shader. The availability of different passes with no additional effort is a major advantage of using RenderMan as opposed to other rendering software. Common AOVs include: occlusion (occ),

diffuse, specular, subsurface scattering (sss), reflection, refraction, beauty, mask, and ambient. In addition to these common AOVs, any unique AOV desired can be written into the shader for output with the others.

The topics covered in this chapter are explored further in the discussion of the implementation and design process of the jellyfish characters. Chapter 3 also includes the shader writing process in Renderman specifically for the jellyfish, code for which is available in Appendix A. The use of refractions and subsurface scattering is explained as well from an aesthetic perspective. The interaction between ray depth and primVars to handle refractions in the jellyfish is also addressed, along with a discussion of a unique AOV for the Flyboy Captain Jellyfish.

Chapter 3

Implementation, Workflow, and Processes

Visual development and the vision of the director created an overall look for each of the main jellyfish characters and their crowd characters. These early character designs and model paint-overs created a starting point for creating the different maps necessary in surfacing. Using the visual development images displayed in Figure 3.1 as inspiration, textures and images of actual fish and jellyfish were collected to aid in the texture map painting process. In addition to the diffuse map, a grayscale map was created to ensure that colors were applied only where desired, blurring was controlled where necessary, and displacements and bumps were used to give the character dimension. After these techniques were refined, duplicating them and making minor changes for each member of the gang required only minimal time and effort. In the following sections, the process for developing the jellyfish's eyes, the method for writing a jellyfish shader in Renderman, and the design choices for creating the pirate and flyboy characters will be discussed.

3.1 Creating Fish as Jellyfish Eyes

Since the jellyfish eyes were fish that functioned independently as characters, rendering them in a visually interesting way as individual entities was important. However, they could not be distracting once integrated as part of the larger jellyfish. A single side of each of two fish would

Figure 3.1: Pirate and Flyboy Characters Visual Development

Figure 3.2: Jellyfish Eyes Visual Development

act as the left and right eyes of the jellyfish, respectively, and thus needed to be relatively similar; however, each also needed to be visually different as though they were two separate fish (Figure 3.2). In general, integrating minute differences in the texture of a character is important when creating any organic creature since patterns should not be repeated in a mechanical way. In these fish, one necessary trait was that the eyes needed to stand out so that the viewer recognized them as belonging to the jellyfish. Another important trait was that they needed to be clearly understood since the eyes were the primary source of expression for the jellyfish. Finally, the dorsal fin of the fish would double as the jellyfish's eyebrows to give the animators enhanced expression; therefore, it needed to be set apart from the body of the fish in a natural way.

Figure 3.3: Potter's Angelfish [6]

For the Pirate Jellyfish character, whose design called for saturated reds, oranges and yellows, a Potter's Angelfish was a suitable candidate for the eyes (Figure 3.3). The pattern on the body is a tiger-striped pattern and it matches nicely alongside the bandanna pattern of the Pirate Jellyfish. Additionally, the natural fiery oranges of the dorsal fin create a strong eyebrow for the jellyfish, especially in combination with the dark teals and blues in the body, which provide the necessary contrast for emphasizing the white eyes. As a creative touch, a small scar was added to the eye on one side of the fish to show its personality connection to the ruggedly scarred Pirate Jellyfish character. Figure 3.4 displays the two sides of the Pirate Fish in Mari, while Figure 3.5 displays the diffuse map once exported from Mari and ready for attachment to the fish shader in Maya.

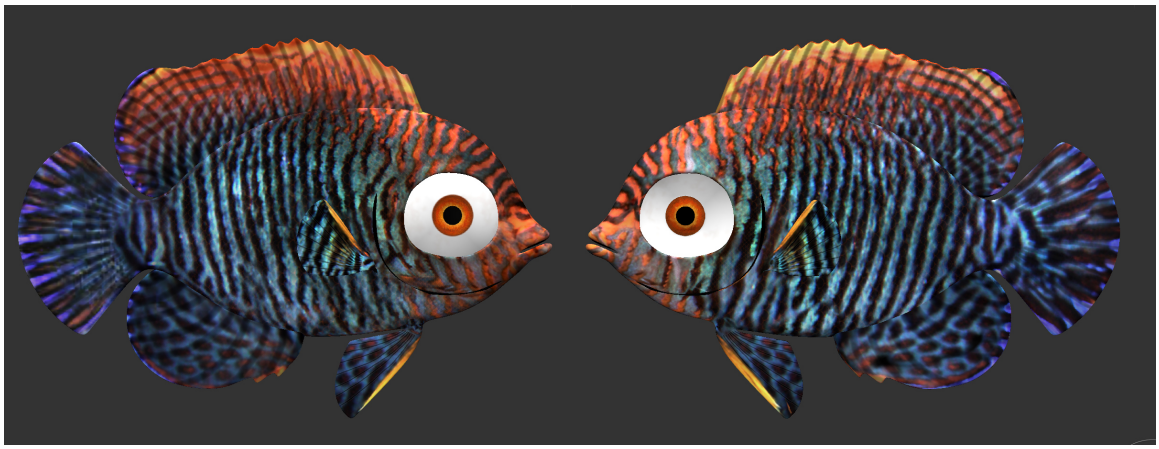

Figure 3.4: Left and Right Sides of Pirate Fish in Mari

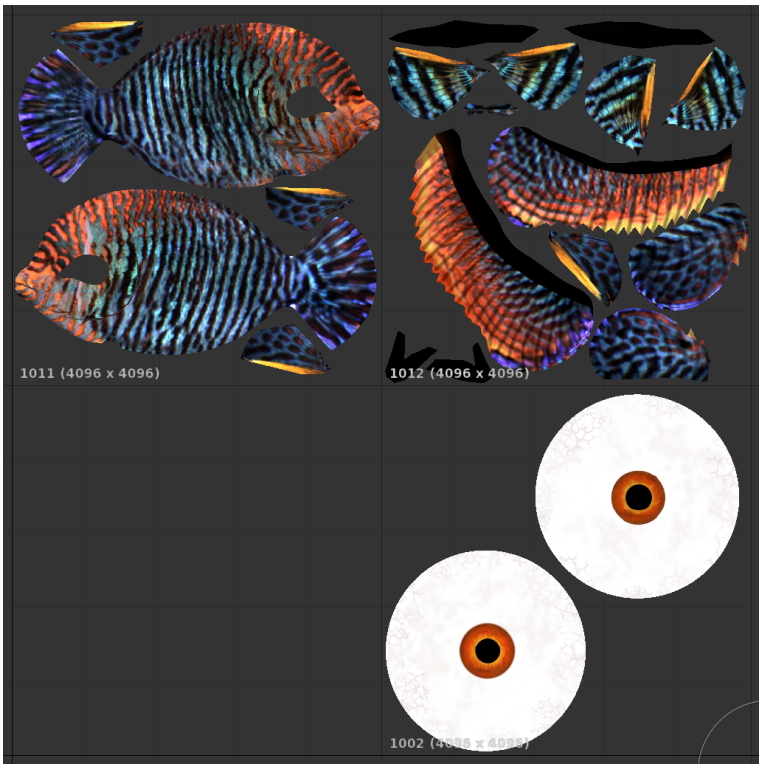

Figure 3.5: Pirate Fish Diffuse Maps

Figure 3.6: Damsel Fish and Military Facepaint Inspiration [9] [19]

For the Flyboy Jellyfish character, which was much cleaner with muted browns and yellows, as well as simpler patterns, the design was more inspired by the character's theme rather than color palette. The inspiration for the striping pattern on the flyboy fish was the Black and White Damsel fish, which mimics the look of military face paint (Figure 3.6). This pattern drew attention to and enhanced the Flyboy Jellyfish's military-inspired personality.

The colors of the fish consisted of a yellow-brown that matched the color scheme of the flyboy, and a dark blue contrasting stripe to ensure the eyes still stood out strong from the body of the fish. The dorsal fin was given a light blue and white overlay to create the eyebrow and to connect the fish to the Flyboy Jellyfish's electricity-generating power. Figures 3.7 and 3.8 show each side of the Flyboy's fish eyes and the diffuse maps from Mari.

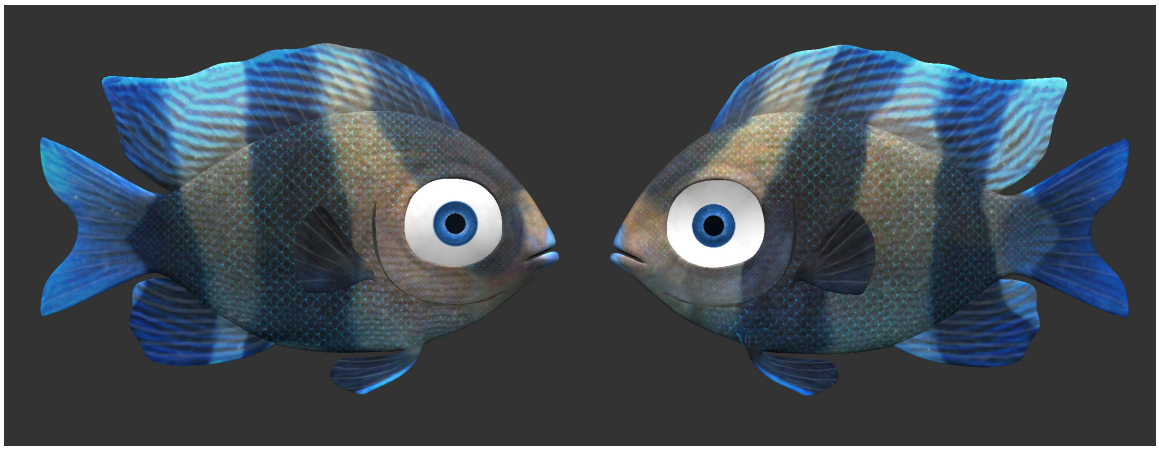

Figure 3.7: Left and Right Sides of Flyboy Fish in Mari

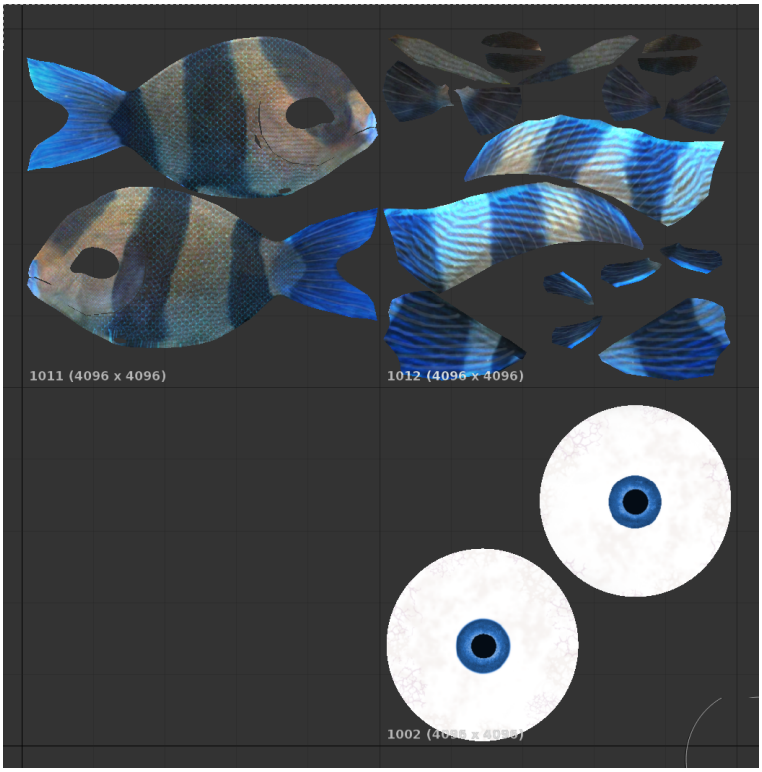

Figure 3.8: Flyboy Fish Diffuse Maps

Figure 3.9: Pirate Jellyfish at Different Indices of Refraction

3.2 Writing a Jellyfish Shader

Creating a customized shader in RenderMan for the jellyfish allowed for the control and implementation of features such as refraction and subsurface scattering, which were necessary for creating the desired look of the main characters. Although the diffuse maps for the two kinds of jellyfish would need to be different, the overarching features would remain the same; therefore, these features were written directly into the shader itself, and included: refraction, subsurface scattering, and the use of primVars and ray depth to control refractions.

Refraction is a key element in underwater environments since water is a denser medium than air. Further, the jellyfish bell needed to be a translucent and refractive material. For the jellyfish, a variety of indices of refraction were tested to find an appropriate value that worked. Even though standard jelly has an index of refraction of 1.38, in order to prevent too much distortion within the bell, since visibility of the fish eyes through the bell was crucial, an index of refraction of 1.05 worked best for the characters because it did not distort the fish or inner parts of the jellyfish within the bell. Figure 3.9 shows the Pirate Jellyfish with different indices of refraction.

In order to create the soft edge around the jellyfish, an existing technique for subsurface scattering that simulates a velvet appearance was implemented. The technique combines an outer edge color with an inner color using the smoothstep function available in the RenderMan library, which creates a gradient using the geometry normal to determine where the edge begins. In this production, the outer color was a light blue to mimic the ocean water's glow around the jellyfish, which was added directly to the diffuse map, while the inner color would simply be the diffuse map:

```
cOut = color sToL(color texture(sDiffuseMap, ss, tt));
```

```
cOut += sssColor;

cIn = mix(cDiffuse, sDiffuseMap, eyeMap);
```

The subsurface scattering is controlled in the shader based on light dependence. If the subsurface scattering is light-dependent, then the brighter the light, the more intense the subsurface scattering will be where the object is illuminated. If the light source is limited, the subsurface can be made more independent to help create the glow. For the jellyfish, the shader's subsurface scattering was mostly light-dependent.

Another feature written into the custom jellyfish shader required a special map created in Mari to determine the amount of the diffuse map to display in the refractions and subsurface scattering passes. With black being zero and white being one, the shader uses the float values in the grayscale map and applies it to the diffuse map using the RenderMan mix function:

```
cRefrac = mix(cRefrac, color texture(sDiffuseMap, ss, tt)), eyeMap);
```

Figure 3.10 shows the diffuse map, the special grayscale map (eyeMap), and the final result when rendered with a blue ocean environment sphere. The map was created in Mari by duplicating the diffuse layers, e.g. the spotting or bandanna, and desaturating and brightening those layers to the desired resulting value between zero and one. The values were important because the jellyfish's eyes needed to be clearly seen as they are expressing and moving around within the bell, but not absolutely clear to demonstrate that the fish eyes were encapsulated within the jellyfish bell.

3.3 Creating the Pirate Jellyfish Captain

The primary color scheme for the Pirate Jellyfish Captain included saturated reds, oranges, and yellows with deep purple accents. To create an interesting and believable bandanna for the jellyfish, a pattern from an image of a Purple-Striped Jellyfish was used (Figure 3.11). In Mari, a feature called the "paint-through" tool allows the artist to use any image as a stamp to paint it directly onto the model. With the paint-through tool, the bandanna pattern was created using the Purple-Striped Jellyfish image, followed by a filter feature also available in Mari for hue correction and color curve adjustment to change the purples and pinks to reds and yellows. Figure 3.12 shows the bandanna before and after the color corrections. These features allow for quick, easy tweaks to textures that do not initially match the desired color scheme of a character.

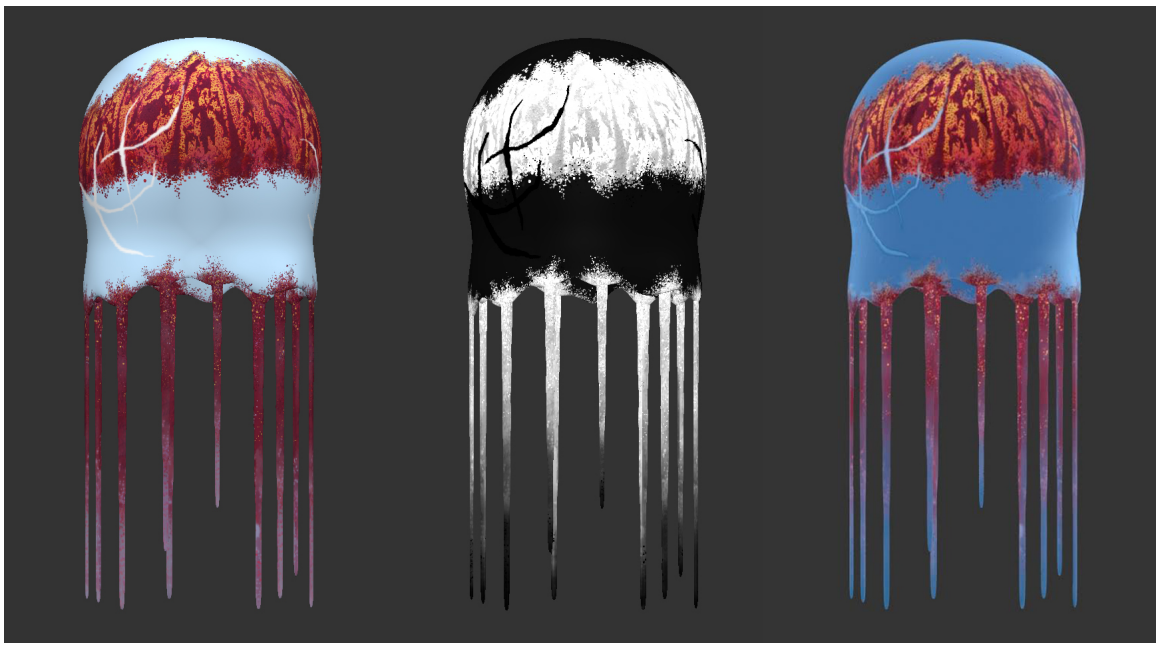

Figure 3.10: Diffuse Map, Grayscale Map, Final Result

Figure 3.11: Purple-Striped Jellyfish [7]

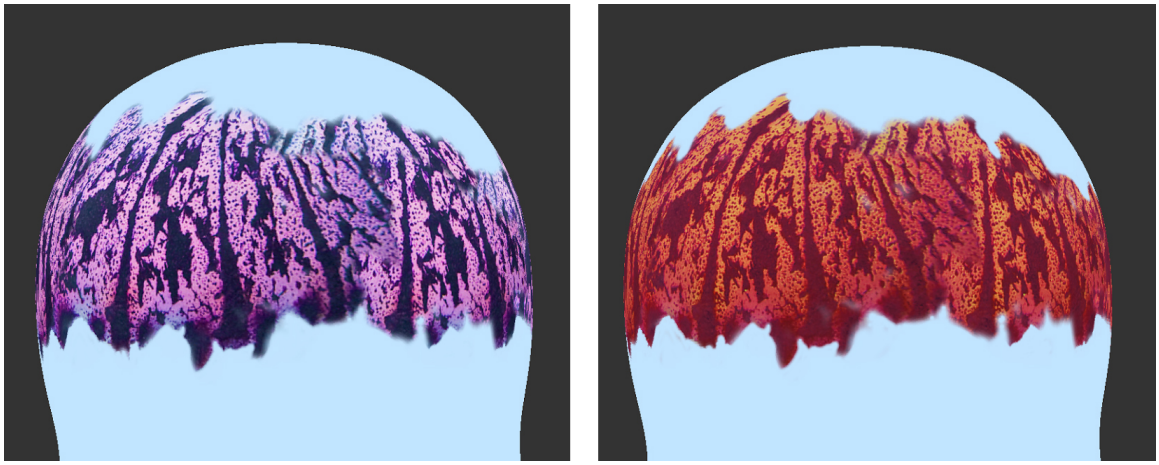

Figure 3.12: Bandanna Before and After Color Corrections

Figure 3.13: Bell Tentacles Before and After Surfacing Hides Geometry

Additionally, the color-corrected bandanna served as the color palette for the remainder of the jellyfish because the texture layer itself could be exported and used as another paint-through image. Mimicking the spotting pattern seen on real jellyfish, the edges of the bandanna were extended to eliminate the harsh line for a more organic appearance. The spotting pattern was repeated along the bottom edge of the bell of the jellyfish as well because although this pattern can be seen in real jellyfish, the surfacing here was crucial for hiding the connection points between the geometry of the small tentacles and the geometry of the bell (Figure 3.13). The spotting pattern was repeated when creating the larger tentacles of the jellyfish by applying it to the edges of the red-purple spine down the center of the tentacle, and again along the ridges of the brain of the jellyfish (Figure 3.14).

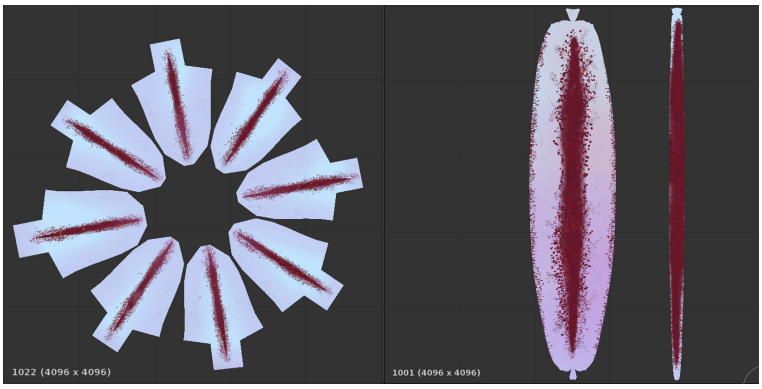

Figure 3.14: Brain and Tentacles Maps

Figure 3.15: Pirate Ring Reference Images

In conjunction with the jellyfish shader, a displacement map was created to produce scarring across the jellyfish bell. These scars made the Pirate Jellyfish Captain appear rugged and war-worn to fit his brutish personality. This same map was used within the shader to brighten and distort the refractions just within the scars; thus, the bell appeared to exhibit scar tissue in it, but just in those areas.

The final feature of the Pirate Jellyfish Captain was the three rings on his front tentacles. These rings were created based on two ring references seen in Figure 3.15. Using Mari, the ring images were painted onto the jellyfish ring followed by a layer of dirt and grime to mimic the underwater aging of the metal. Two of the rings were the same, but a hue shift was applied to the silver versions to create a gold version. Within the shader, reflectivity was added to give them some reaction to the environment, but was limited using a reflectivity map to mimic the way dirt layers prohibit reflectivity (Figure 3.16).

Figure 3.16: Pirate Rings

3.4 Creating the Flyboy Jellyfish Captain

The director did not want the jellyfish to have any areas on the bells that were entirely transparent or that created a visible shape, like aviator goggles, because it would look unnatural as though the eyes were swimming in a fishbowl or looking through a window. To avoid creating the appearance of such a window on the Flyboy Jellyfish Captain's face, a gentle gradient of color was applied from the top of the bell down toward the base. Using a White-Spotted Jellyfish (Figure 3.17) as a reference, white polyps were added to the dome of the jellyfish's bell. Although organic, these polyps were still fairly uniform and evenly distributed, and were given dimension by using a simple spot map painted in Mari (Figure 3.18). First, the map was used in the diffuse layer to show the white spots across the dome. Second, this same map was used as a positive displacement to raise the bumps from the dome. Lastly, within the shader itself, the texture map was read in using a blur function available in the RenderMan library. This blurred map could then be used to create a color based on the existing refraction colors on the bell by multiplying by two to brighten it:

```
spotMap = texture(sSpotMap, ss, tt, "blur", 0.005);

color glowSpots = spotMap * (cRefrac * 2);
```

The result, when mixed into the final color, simulated a subtle glow around the polyps, making them seem iridescent. Additionally, since the spotMap could also be written as an AOV, the compositors could control the intensity of the glow based on the shot. Figure 3.19 shows the result of the spot map in black and white. To maintain a level of consistency across the entire

Figure 3.17: White Spotted Jellyfish

Flyboy Jellyfish Captain, the polyps and the glowing maps were also applied to the tentacles. Using a similar brown-to-yellow-to-transparent gradient as the one on the bell, the tentacles were given a brown tone at the top extending to a yellow middle ending at transparent tips.

3.5 Creating the Jellyfish Gangs

After the designs for the Pirate and Flyboy Captains were finalized, the gang members could be created easily based on the existing maps and shaders that worked for the captains. To provide the layout artist more control over the gang members present in each individual shot, the RenderMan jelly shader was written with the functionality to choose the gang member to display in the shader in Maya (Figure 3.20). This feature minimized the number of shaders that moves through the pipeline, and removed the need to attach different shaders to rigs in the layout or lighting stages.

For the Pirate's gang, the Pirate Jellyfish Captain's bandanna was reused, but reduced slightly in size, to show their subordination to the captain. Using the same maps in Mari used for the captain, a hue shift toward more red, orange, yellow, or purple created the options for his gang

Figure 3.18: Flyboy Spots Map

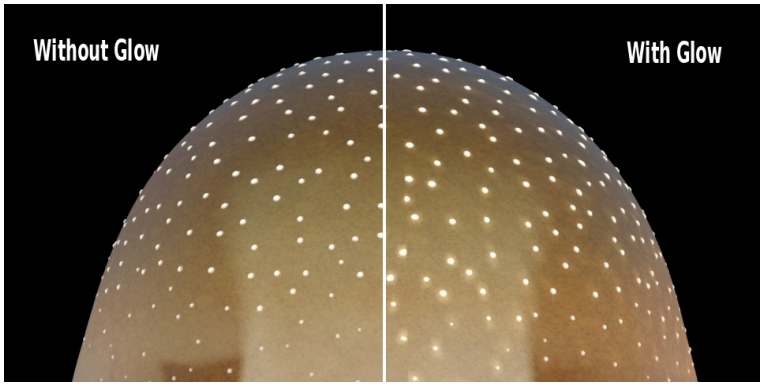

Figure 3.19: Polyps on Flyboy

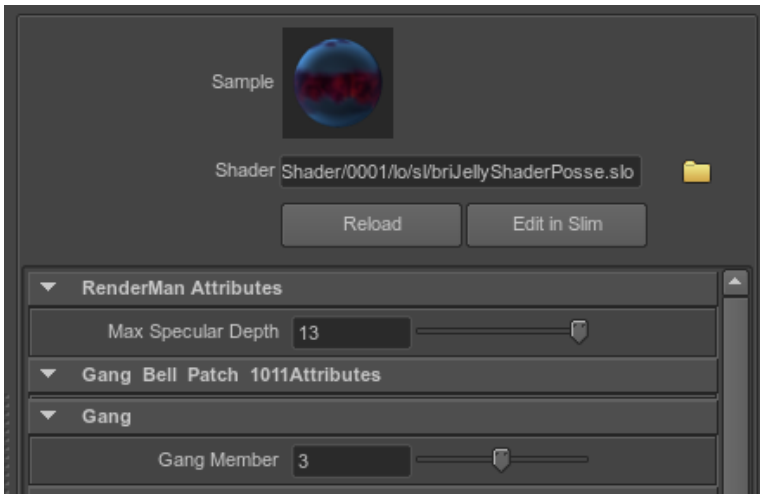

Figure 3.20: Gang Member Selection in Maya

members. Figure 3.21 displays the Pirate gang member options.

Similar to the Pirate's gang, the Flyboy's gang was also hue shifted in Mari toward more orange, green, blue, or pink. To maintain some similarities to the original color scheme, a subtle yellow transition remained under the muted primary dome color. The number of spots on the top of the bell was also decreased for the gang members to signify their lower rank. Figure 3.22 displays the Flyboy gang member choices.

After the surfacing for the characters was completed via the aforementioned processes, the characters were lit and composited into the final shots. Each character was lit using a blue ocean environment sphere using most of the environment items in the scene for refractions, shadows, and reflections. Chapter 4 displays the resulting final renders of the characters in various shots of *Peanut Butter Jelly.*

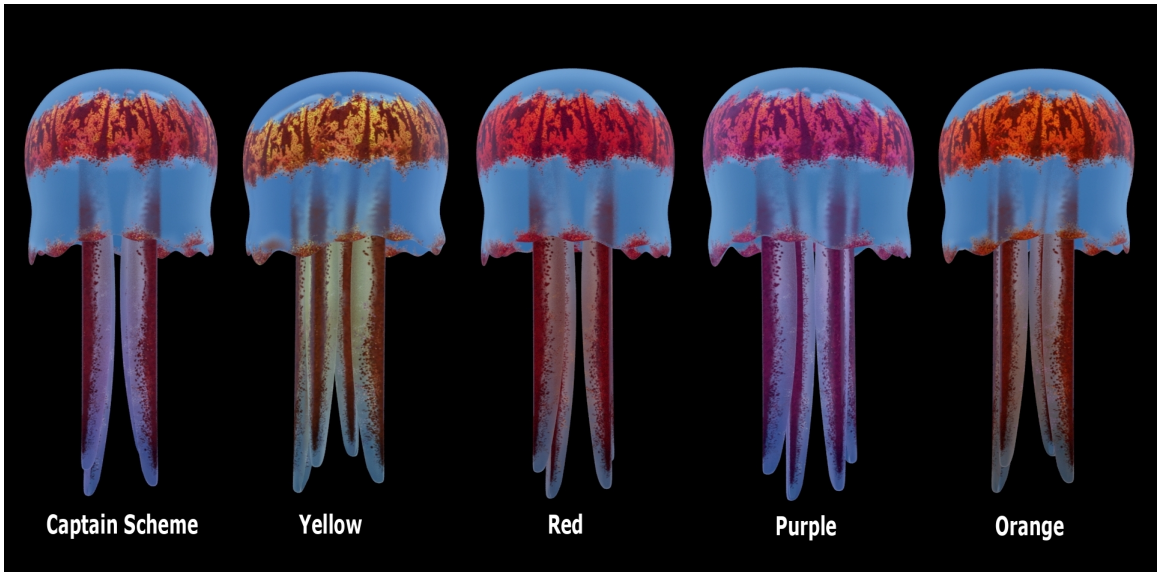

Figure 3.21: Pirate Gang Color Options

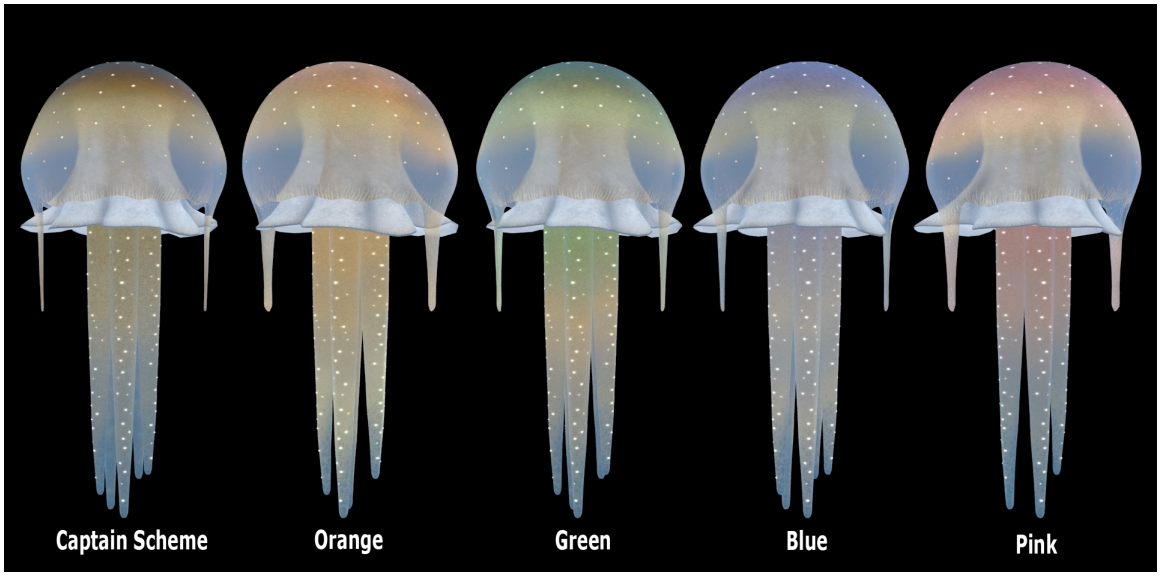

Figure 3.22: Flyboy Gang Color Options

Chapter 4

Final Results

The surfacing techniques used for *Peanut Butter Jelly* created an end result that worked well for all shots of the production. Due to the customizability of the shader used for the characters and the secondary outputs (AOVs) available within that shader, the rendering for the characters was fairly quick and easy to adjust in compositing based on shot requirements. Ultimately, minimal adjustments were needed. The resulting final renders of the production are included in this chapter.

Figure 4.1: *Peanut Butter Jelly* Shot 01

4.1 Jellyfish Eyes

Of the eighteen shots within *Peanut Butter Jelly*, only three shots show the fish outside of the jellyfish bells. The Pirate Jellyfish Captain's fish eyes are present in all three of those shots, but the Flyboy Jellyfish Captain's fish eyes appear only in one. In Shot 01 (Figure 4.1), the fish can be seen swimming around the coral reef at the bottom-right corner of the screen. In Shot 02 (Figure 4.2), the fish are swimming toward the hole in the ship approaching from screen right. Finally, in Shot 17 (Figure 4.3) all of the fish are center screen.

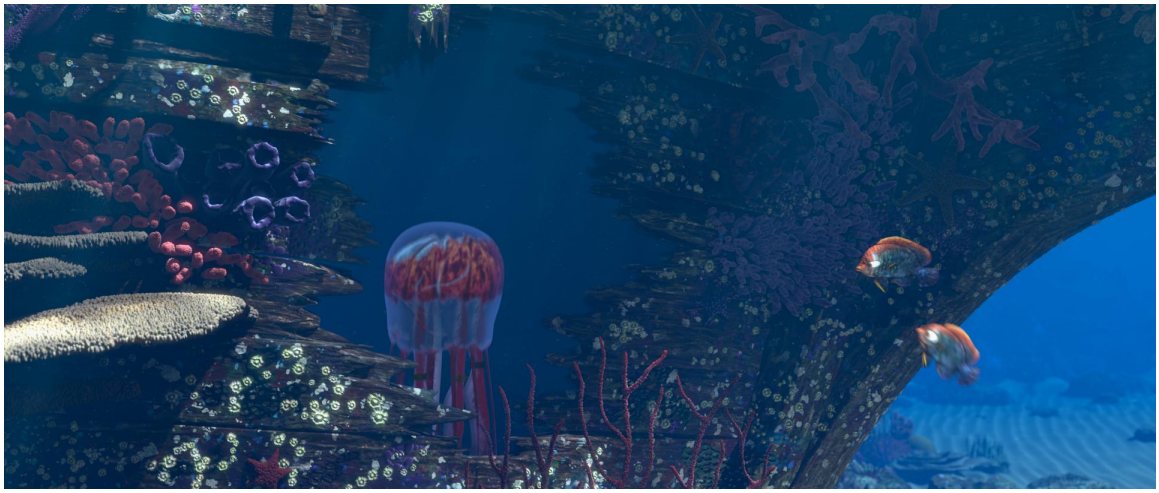

Figure 4.2: *Peanut Butter Jelly* Shot 02a

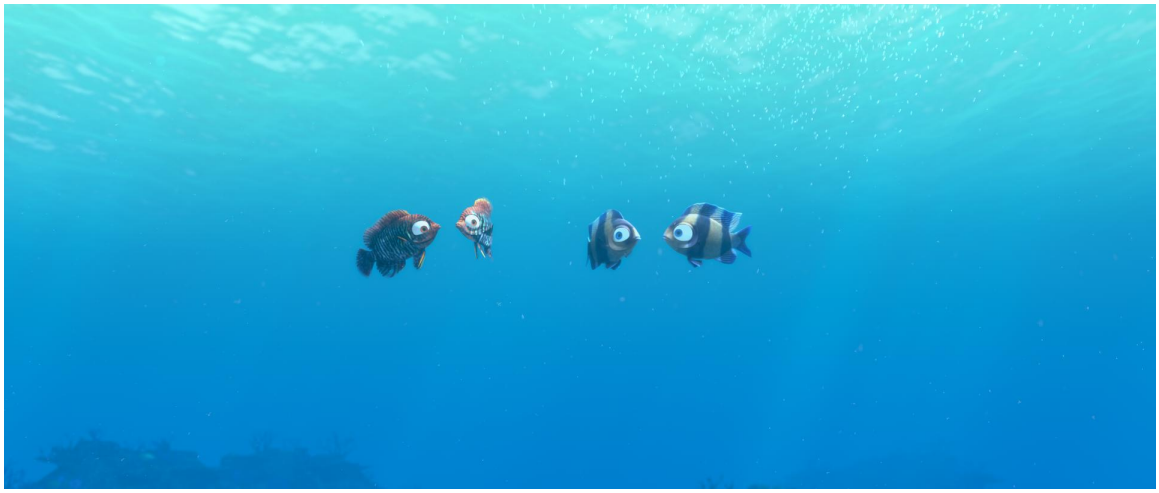

Figure 4.3: *Peanut Butter Jelly* Shot 17a

4.2 Pirate Jellyfish Captain and Pirate Jellyfish Crowd

The Pirate Jellyfish Captain is present by himself in three of the eighteen shots (Figures 4.4, 4.5 and 4.6). The Pirate Jellyfish Captain's crowd characters appear with him in four of the eighteen shots (Figures 4.7, 4.8, 4.9 and 4.10). The surfacing of those characters is demonstrated in each of these rendered shots.

Figure 4.4: *Peanut Butter Jelly* Shot 02b

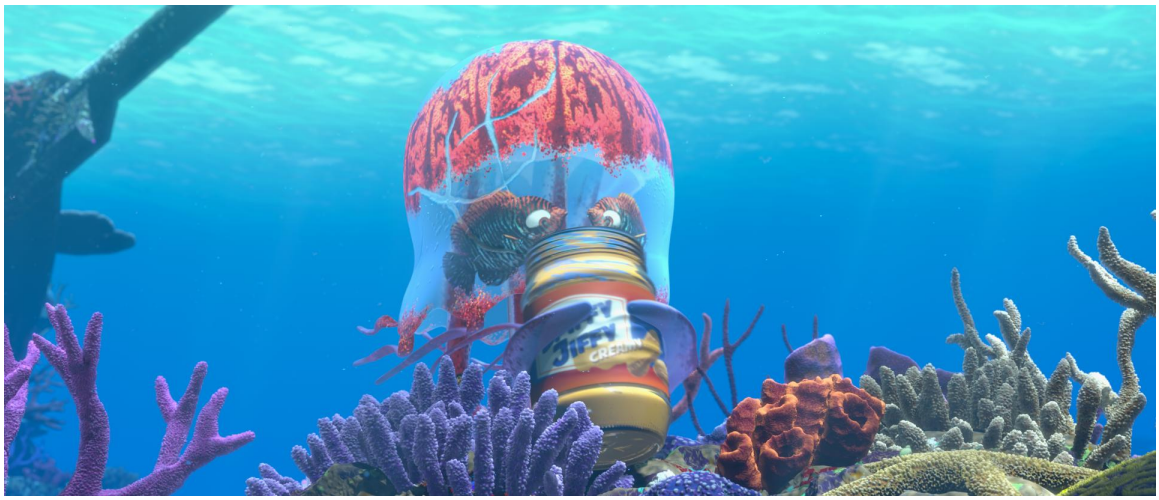

Figure 4.5: *Peanut Butter Jelly* Shot 05

Figure 4.6: *Peanut Butter Jelly* Shot 12

Figure 4.7: *Peanut Butter Jelly* Shot 04

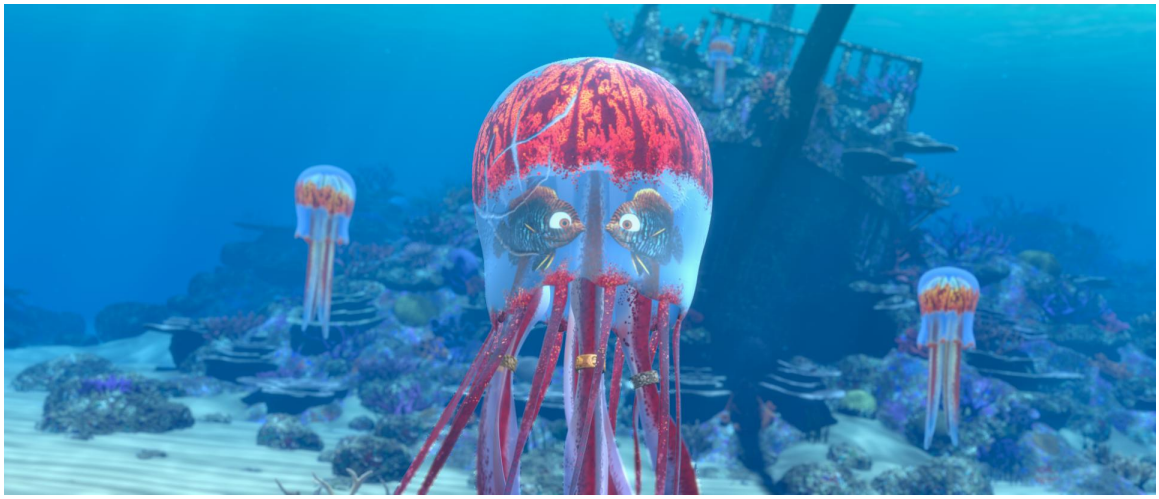

Figure 4.8: *Peanut Butter Jelly* Shot 09a

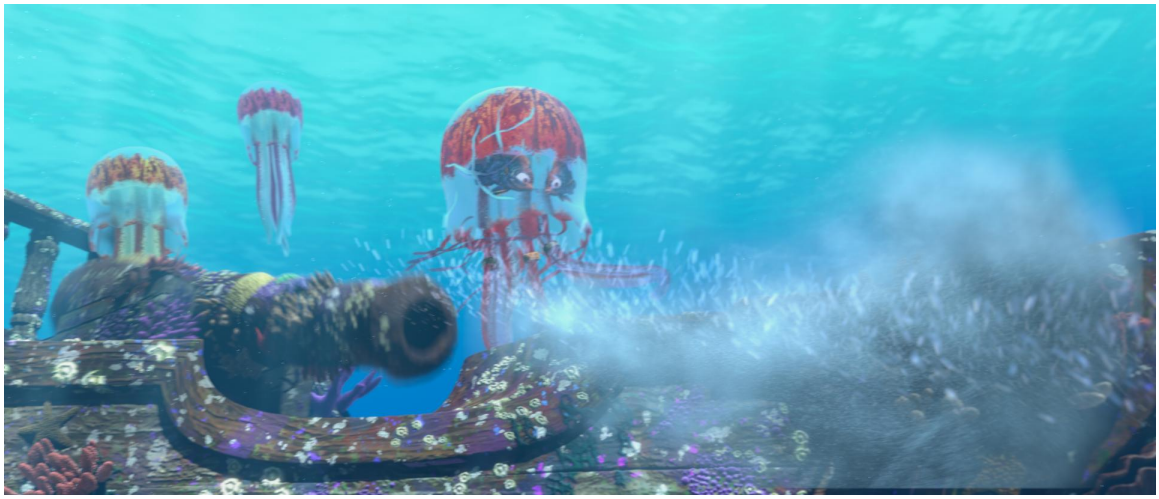

Figure 4.9: *Peanut Butter Jelly* Shot 14

Figure 4.10: *Peanut Butter Jelly* Shot 16

4.3 Flyboy Jellyfish Captain and Flyboy Jellyfish Crowd

The Flyboy Jellyfish Captain is the second main character introduced in the short animation, and as such, is present in only two of the eighteen shots by himself. The short contained no shots where only the Flyboy Captain and his corresponding crowd characters are present without a Pirate Jellyfish. Figures 4.11 and 4.12 show the final renders of those shots.

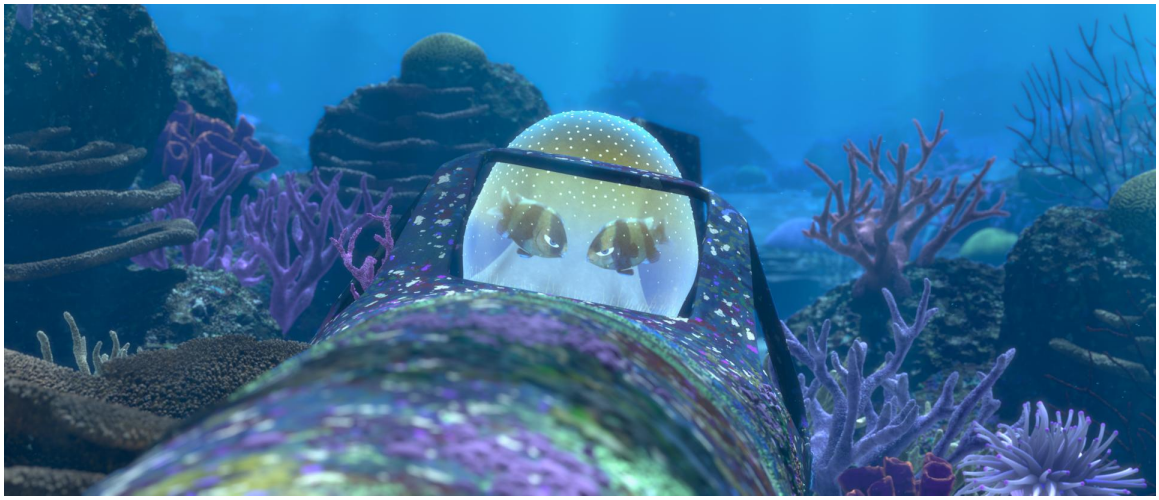

Figure 4.11: *Peanut Butter Jelly* Shot 10

Figure 4.12: *Peanut Butter Jelly* Shot 15

4.4 Additional Renders of Characters

The two captains and their gangs come into contact with each other in five of the eighteen shots. Figures 4.13 through 4.17 capture the character surfacing in those shots.

Figure 4.13: *Peanut Butter Jelly* Shot 06

Figure 4.14: *Peanut Butter Jelly* Shot 07

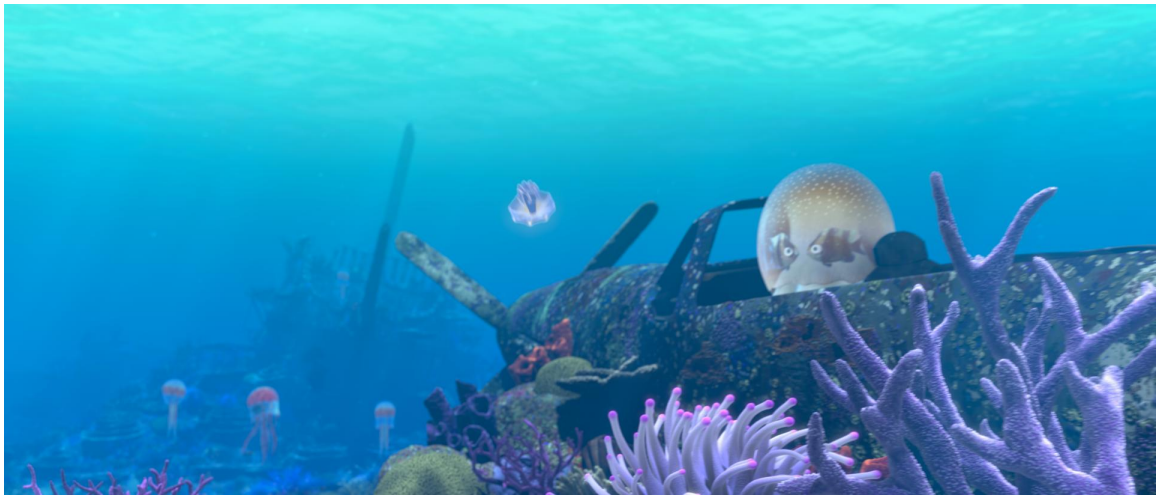

Figure 4.15: *Peanut Butter Jelly* Shot 09b

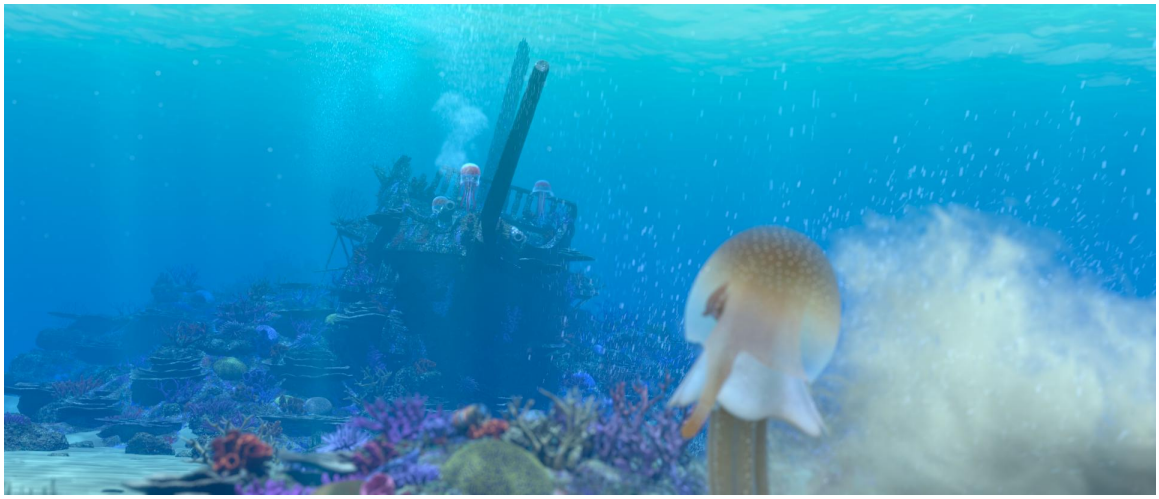

Figure 4.16: *Peanut Butter Jelly* Shot 13

Figure 4.17: *Peanut Butter Jelly* Shot 17b

Chapter 5

Future Applications and Conclusion

The complex and unique nature of the surfacing completed for this production has great value for surfacing in future productions. Much of the code, techniques, and application can be replicated and adapted for surfacing other characters or even environmental elements or props.

5.1 Future Applications

The process for implementing maps created in Mari, and altering them within the Renderman shader would greatly benefit other productions. The ability to create different types of maps in Mari quickly by duplicating paint layers and manipulating them using filters allows the surfacing artist to easily create grayscale maps that can be read into the shader and used to specifically manipulate the appearance of different render layers (i.e., diffuse or refraction) on the geometry surface. The jellyfish shader is a great example of this feature because two separate characters with unique traits used the same shader in addition to their own specialty maps and variable controls, i.e., the Pirate Jellyfish Captain's scars and the Flyboy Jellyfish Captain's spots. Also, the use of primVars in this production allowed a single shader type to be applied to multiple parts of a character, thus reducing the number of shaders passing through the pipeline. This result decreased the heaviness of the scene and enabled lighters to locate control items more easily in the Maya Hypershade.

5.2 Summary and Conclusion

The ultimate goal for the main character surfacing was to create two primary characters using one custom shader and two secondary characters to function both independently and symbiotically as the main character's eyes. The main characters needed to appear as refractive, translucent jellyfish in a cartoon-stylized sense that worked well with the somewhat photo-realistic yet stylized underwater environment. Employing RenderMan's versatile shader language and Mari's intuitive map painting tools and filters, the surfaces of these characters transferred from loose visual ideas to dynamic animated characters that bring color, personality and life to the animated short production, *Peanut Butter Jelly.*

Appendix

Jellyfish Shader Code

```
//Note: Parts of this code were adapted from open-source RenderMan code

/****** THE DISPLAY FOR THE SHADER IN MAYA ******/
/*
<meta id= "slim"><![CDATA[
slim 1 appearance slim {
  instance surface briJellyShader briJellyShader {

  collection void colJellyExtras {
    state open
    label "Gang Member"
    parameter float gangMember {
      label "Gang Member"
      range {1.0 5.0 1.0}
      default 1
    }
  }
  collection void colDiffuse {
    state open
    label "Diffuse"
    parameter string sDiffuseMap {
      label "Diffuse Map"
      subtype texture
      default ""
    }
    parameter color cDiffuse {
      label "Diffuse Color"
      default {0.5 0.5 0.5}
    }
    parameter float diffuseMix {
      label "Diffuse Amount"
      range {0.0 1.0 0.01}
      default 1
```

```
  }
}
collection void colAmbient {
  state open
  label "Ambient"
  parameter string sAmbientMap {
    label "Ambient Map"
    subtype texture
    default ""
  }
  parameter color cAmbient {
    label "Ambient Color"
    default {0.0 0.0 0.0}
  }
  parameter float ambientMix {
    label "Ambient Amount"
    range {0.0 1.0 0.01}
    default 1
  }
}
collection void colSpecular {
  state open
  label "Specular"
  parameter float roughness {
    label "Roughness"
    default 0.2
  }
  parameter string sSpecularMap {
    label "Specular Map"
    subtype texture
    default ""
  }
  parameter color cSpecular {
    label "Specular Color"
    default {0.5 0.5 0.5}
  }
  parameter float specularMix {
    label "Specular Amount"
    range {0.0 1.0 0.01}
    default 1
  }
}
collection void colTrans {
  state open
  label "Transparency"
  parameter string sTransparency {
    label "Transparency Map"
    subtype texture
    default ""
  }
  parameter float fTransparency {
```

```
      label "Transparency"
      range {0.0 1.0 0.01}
      default 0.0
    }
  }
  collection void colReflec {
    state open
    label "Reflections"
    parameter string sReflectivity {
      label "Reflectivity Map"
      subtype texture
      default ""
    }
    parameter float fReflectivity {
      label "Reflectivity"
      range {0.0 1.0 0.01}
      default 0.0
    }
    parameter string sBlurMap {
      label "Blur Map Captain"
      subtype texture
      default ""
    }
    parameter string sBlurMapGang {
      label "Blur Map Gang"
      subtype texture
      default ""
    }
    parameter float reflecBlur {
      label "Blur"
      default 0.0
    }
    parameter float reflecSamps {
      label "Samples"
      default 4
    }
  }
  collection void colRefrac {
    state open
    label "Refractions"
    parameter string sRefraction {
      label "Refraction Map"
      subtype texture
      default ""
    }
    parameter string captainDiffuse {
      label "Captain Map"
      subtype texture
      default ""
    }
    parameter string oneDiffuse {
```

```
  label "Map Gang1"
  subtype texture
  default ""
}
parameter string twoDiffuse {
  label "Map Gang2"
  subtype texture
  default ""
}
parameter string threeDiffuse {
  label "Map Gang3"
  subtype texture
  default ""
}
parameter string fourDiffuse {
  label "Map Gang4"
  subtype texture
  default ""
}
parameter string fiveDiffuse {
  label "Map Gang5"
  subtype texture
  default ""
}
parameter color cRefraction {
  label "Refraction Color"
  default {1.0 1.0 1.0}
}
parameter float fRefraction {
  label "Refraction"
  range {0.0 1.0 0.01}
  default 0.0
}
parameter float refracBlur {
  label "Blur"
  default 0.0
}
parameter float refracSamps {
  label "Samples"
  default 4
}
parameter float refracAtten {
  label "Attenuation"
  default 0.05
}
parameter float refracIndex {
  label "Index of Refraction"
  default 1.33
}
parameter float RRthreshold {
  label "RR Threshold"
```

```
      default 0.001
    }
    parameter float maskDiffuse {
      label "Mask Diffuse"
      subtype switch
      default 0
    }
    parameter float maskSpecular {
      label "Mask Specular"
      subtype switch
      default 0
    }
    parameter string sRefractControlMap {
      label "Refraction Control Map"
      subtype texture
      default ""
    }
    parameter float controlLevel {
      label "Control Value"
      default 1.0
    }
  }
  collection void colBump {
    state open
    label "Bump"
    parameter string sBumpMap {
      label "Bump Map"
      subtype texture
      default ""
    }
    parameter float bumpAmount {
      label "Bump Amount"
      range {-5 5 0.01}
      default 0
    }
  }
  collection void colDisp {
    state open
    label "Displacement"
    parameter string sDispMap {
      label "Displacement Map"
      subtype texture
      default ""
    }
    parameter float dispAmount {
      label "Displacement Amount"
      range {0 50 0.01}
      default 0
    }
  }
  collection void colOcc {
```

```
  state open
  label "Occlusion"
  parameter float calcOcc {
    label "Calculate"
    subtype switch
    default 0
  }
  parameter float occSamples {
    label "Samples"
    range {0 1024 1}
    default 128
  }
  parameter float occMaxVar {
    label "Max Variation"
    range {0 1 0.001}
    default 0.02
  }
}
collection void colSSS {
  state open
  label "SSS"
  parameter float sssMix {
    label "SSS Amount"
    default 0.2
  }
  parameter float sssEdgeMix {
    label "NL vs LD"
    range {0.0 1.0 0.01}
    default 0.3
  }
  parameter float sssEdgeDiffusion {
    label "NL Edge Diffusion"
    default 0.1
  }
  parameter float sssEdginess {
    label "LD Edge Diffusion"
    default 0.1
  }
  parameter float sssStepBottom {
    label "smoothStep Bottom"
    default 0.1
  }
  parameter float sssStepTop {
    label "smoothStep Top"
    default 0.9
  }
  parameter color sssInner {
    label "Inner Color"
    default {0.0 0.0 0.0}
  }
  parameter color sssOuter {
```

```
      label "Outer Color"
      default {1.0 1.0 1.0}
    }
    parameter color sssHighlight {
      label "Outerglow Color"
      default {0.0178 0.1604 0.3124}
    }
    parameter float sssHLIntensity {
      label "Outerglow Intensity"
      default 2.0
    }
    parameter float sssHLMin {
      label "Innerglow Intensity"
      default 0.2
    }
    parameter float sssUseRefrac {
      label "Use Refraction Color"
      subtype switch
      default 0
    }
  }
  collection void colJellyExtras {
    state open
    label "JellyExtras"
    parameter string sEyeMap {
      label "Eye Map Captain"
      subtype texture
      default ""
    }
    parameter string sEyeMapGang {
      label "Eye Map Gang"
      subtype texture
      default ""
    }
    parameter string sSpotMap {
      label "Spot Map Captain"
      subtype texture
      default ""
    }
    parameter string sSpotMapGang {
      label "Spot Map Gang"
      subtype texture
      default ""
    }
  }
  }
}
]]></meta>
*/

/****** BEGIN SHADER CODE ******/
```

```
class briJellyShader(
  color cDiffuse = 0.5;
  string sDiffuseMap = "";
  color cAmbient = 0;
  string sAmbientMap = "";
  string sTransparency = "";
  float fTransparency = 0;
  color cSpecular = 0.5;
  string sSpecularMap = "";
  uniform float ambientMix = 0.0;
  uniform float diffuseMix = 0.85;
  uniform float specularMix = 1;

  uniform float refracIndex = 1.33;
  uniform float refracAtten = 1;
  float RRthreshold = 0.001;

  /* REFLECTIONS */
  float fReflectivity = 0;
  string sReflectivity = "";
  float reflecBlur = 0;
  float reflecSamps = 4;

  /* REFRACTIONS */
  float fRefraction = 0;
  string sRefraction = "";
  string captainDiffuse = "";
  string oneDiffuse = "";
  string twoDiffuse = "";
  string threeDiffuse = "";
  string fourDiffuse = "";
  string fiveDiffuse = "";
  string sRefractControlMap = "";
  float controlLevel = 1;
  float refracBlur = 0;
  float refracSamps = 4;
  color cRefraction = 1;
  float maskDiffuse = 0;
  float maskSpecular = 0;

  uniform float roughness = 0.2;

  string sBlurMap = "";
  string sBlurMapGang = "";
  string sEyeMap = "";
  string sEyeMapGang = "";
  string sSpotMap = "";
  string sSpotMapGang = "";

  float uOffset = 0;
```

```
    float vOffset = 0;

    /* BUMP */
    string sBumpMap = "";
    float bumpAmount = 0;

    /* DISPLACEMENT */
    string sDispMap = "";
    float dispAmount = 0;

    /* OCCLUSION */
    float calcOcc = 0;
    float occSamples = 128;
    float occMaxVar = 0.02;

    /* SSS */
    color sssInner = 1;
    color sssOuter = 1;
    float sssEdgeMix = 0.3;
    float sssMix = 0.5;
    float sssEdgeDiffusion = 0.1;
    float sssEdginess = 6;
    float sssUseRefrac = 0;
    float sssStepTop = 0.7;
    float sssStepBottom = 0.0;
    float sssHLIntensity = 2;
    float sssHLMin = 0.2;
    color sssHighlight = (0.0178, 0.1604, 0.3124);

    /* OUTPUT AOVS */
    output varying color _occ = 0;
    output varying color _diffuse = 0;
    output varying color _sss = 0;
    output varying color _ssslight = 0;
    output varying color _specular = 0;
    output varying color _mask = 0;
    output varying color _ambient = 0;
    output varying color _reflection = 0;
    output varying color _refraction = 0;
    output varying color _beauty = 0;
    output varying color _zdepth = 0;
    output varying color _sdepth = 0;
    output varying color _difCalc = 0;
    output varying color _mScar = 0;
    output varying color _spotglow = 0;

    //Captain by default
    float gangMember = 0;
)
{
    /***** DISPLACEMENT & BUMP SHADER *****/
```

```
varying normal nbd = 0;

public void displacement(output point P; output normal N){

  color maxC = (1, 1, 1);
  color minC = (0, 0, 0);
  float nD = 0;
  point PP = transform("shader", P);
  nbd = normalize(N);

  float ss, tt;
  ss = mod(s, 1);
  tt = mod((1.0 - t), 1);

  /* DISPLACEMENT CALCULATION WITH MAP */
  if(sDispMap != ""){
    float dmap = sToL(color texture(sDispMap, ss, tt))[0];
    nD = dmap * dispAmount;
    P+= normalize(N) * nD;
    N = calculatenormal(P);
  }

  /* BUMP CALCULATION WITH MAP */
  if(sBumpMap != ""){
    float bmap = sToL(color texture(sBumpMap, ss, tt))[0];
    N = calculatenormal(P + (bmap * normalize(N) * bumpAmount));
  }
}

/***** SURFACE SHADER *****/
public void surface(output color Ci, Oi){
    string overallDiffuse;

/*** SET DIFFUSE REFRACTION MAP BASED ON GANG MEMBER ***/
if (captainDiffuse != "" && oneDiffuse != "" && twoDiffuse != ""
  && threeDiffuse != "" && fourDiffuse != "" && fiveDiffuse != ""){
  if (gangMember == 0){
    overallDiffuse = captainDiffuse;
  }
  if (gangMember == 1){
    overallDiffuse = oneDiffuse;
  }
  if (gangMember == 2){
    overallDiffuse = twoDiffuse;
  }
  if (gangMember == 3){
    overallDiffuse = threeDiffuse;
  }
  if (gangMember == 4){
    overallDiffuse = fourDiffuse;
  }
```

```
  if (gangMember == 5){
    overallDiffuse = fiveDiffuse;
  }
}

extern vector I;
normal Nn = normalize (N);
vector In = normalize(I);
normal Nf = faceforward(Nn,In);
Nn = Nf;

point PP = transform("shader", P);

vector V = normalize(-I);
float normalDot = clamp(V . Nn, 0, 1);

float objid = 0;
readprimvar("Objectid", objid);

float ss, tt;
ss = mod(s, 1);
tt = mod((1.0 - t), 1);

float raydepth = 0;
rayinfo("depth", raydepth);

/*** SET PROPER SPOT MAP (CAPTAIN VS CROWD) ***/
float spotMap = 0;
if (sSpotMap != "" && sSpotMapGang){
  if (gangMember < 1){
    spotMap = texture(sSpotMap, ss, tt, "blur", 0.005);
  }
  else{
    spotMap = texture(sSpotMapGang, ss, tt, "blur", 0.005);
  }
}

/**** OCCLUSION ****/
float killocc = 0;
readprimvar("Killocc", killocc);
float occ = 0;
if(killocc == 0 && raydepth == 0){
  occ = occlusion(P, Nn, occSamples, "maxvariation", occMaxVar);
}

/**** TRANSPARENCY ****/
float fTrans = 1.0 - fTransparency;
if(sTransparency != ""){
  fTrans = 1.0 - sToL(color texture(sTransparency, ss, tt))[0];
}
```

```
/**** DIFFUSE ****/
color cDif;
if(sDiffuseMap != ""){
  cDif = sToL(color texture(sDiffuseMap, ss, tt));
}else{
  cDif = sToL(cDiffuse);
}
color difCalc = diffuse(Nn);
cDif *= difCalc * diffuseMix;

/**** SPECULAR ****/
color cSpec;
if(sSpecularMap != ""){
  cSpec = sToL(color texture(sSpecularMap, ss, tt));
}
else{
  cSpec = sToL(color texture(overallDiffuse, ss, tt));
}
cSpec *= specular(Nn, V, roughness) * specularMix;

/**** AMBIENT ****/
color cAmb;
if(sAmbientMap != ""){
  cAmb = sToL(color texture(sAmbientMap, ss, tt));
}
else{
  cAmb = sToL(cAmbient);
}
cAmb *= ambientMix;

/**** REFLECTION / REFRACTION ****/
color ci, hitci, cRefrac, cReflec;
float Kt, Kr, kr, kt;
vector reflDir, refrDir;

// relative index of refraction
float eta = (In.Nn < 0) ? 1/refractIndex : refractIndex;

if(sReflectivity != ""){
  Kr = sToL(color texture(sReflectivity, ss, tt))[0];
}
else{
  Kr = fReflectivity;
}

if(sRefraction != ""){
  Kt = sToL(color texture(sRefraction, ss, tt))[0];
}
    else{
  Kt = fRefraction;
    }
```

```
float fRefractControl = 0;
if(sRefractControlMap != ""){
  fRefractControl = texture(sRefractControlMap, ss, tt);
  if(fRefractControl > 0.5){
    Kt = controlLevel;
  }
}

float maxImportance;
rayinfo("importance", maxImportance); // max of (r,g,b) importance
float reflSam = Kr * kr * reflecSamps * maxImportance;
float refrSam = Kt * kt * refracSamps * maxImportance;

float dist = 0;
float sampCone = reflecBlur;

cReflec = 0;
cRefrac = 0;

// Shoot reflection rays
if (reflSam > RRthreshold) { // shoot 1 or more rays
  color weight = color(Kr * kr);
  float sam = max(round(reflSam), 1);
  ci = 0; hitci = 0;
  gather("illuminance", P, reflDir, sampCone, sam, "weight", weight,
      "surface:Ci", hitci) {
    ci += hitci;
  }
  cReflec += Kr * kr * ci / sam;
}
else if (reflSam/RRthreshold > random()) { // Russian roulette: 0/1
    ray
  ci = 0; hitci = 0;
  gather("illuminance", P, reflDir, sampCone, 1, "surface:Ci",
      hitci) {
    ci += hitci;
  }
  cReflec += ci; // no mult by Kr * kr to increase weight of
      surviving rays
}

color extColor = sToL(cRefraction);
if(Kt > 0){
  if(maskDiffuse == 1){
    cDif *= 1.0 - Kt;
  }
  if(maskSpecular == 1){
    cSpec *= 1.0 - Kt;
  }
  float extinction = refracAtten;
```

```
}

/**** BLUR ****/

/*** SET PROPER BLUR MAP (CAPTAIN VS CROWD) ***/
float blurMap = 0;
if (sBlurMap != "" && sBlurMapGang){
  if (gangMember < 1){
    blurMap = sToL(color texture(sBlurMap, ss, tt))[0];
  }
  else {
    blurMap = sToL(color texture(sBlurMapGang, ss, tt))[0];
  }
  sampCone = blurMap;
}
else {
  sampCone = refracBlur;
}

/**** Set Object IDs for PrimVars ****/
string strid = "0";
if (objid < 0.1){
  strid = "0";}
if (objid > 0.5 && objid < 1.5){
  strid = "1";}
if (objid > 1.5 && objid < 3){
  strid = "2";}
if (objid > 2.5 && objid < 4){
  strid = "3";}
if (objid > 3.5 && objid < 5){
  strid = "4";}
if (objid > 4.5 && objid < 6){
  strid = "5";}

if (refrSam > RRthreshold) { // shoot 1 or more rays
  color weight = color(Kt * kt);
  float sam = max(round(refrSam), 1);
  ci = 0; hitci = 0;
  gather("illuminance", P, refrDir, sampCone, sam,"weight", weight,
       "surface:Ci", hitci, "ray:length", dist, "label", strid) {
    if(-In.Nn > 0) {
      //Clamp the extinction color to prevent surfaces from being
      //100% color filters or absorb 100% of light
      extColor[0] = max(0.05, min(0.95, extColor[0]));
      extColor[1] = max(0.05, min(0.95, extColor[1]));
      extColor[2] = max(0.05, min(0.95, extColor[2]));
      dist *= 0.0328084;

      ci[0] += exp(-extinction * dist * (1-extColor[0])) * hitci[0];
      ci[1] += exp(-extinction * dist * (1-extColor[1])) * hitci[1];
      ci[2] += exp(-extinction * dist * (1-extColor[2])) * hitci[2];
```

```
    }
    else {
      ci += hitci;
    }
  }
  cRefrac += Kt * kt * ci / sam;
}
else if (refrSam/RRthreshold > random()) { // Russian roulette: 0/1
   ray
  ci = 0; hitci = 0;
  gather("illuminance", P, refrDir, sampCone, 1,
       "surface:Ci", hitci, "ray:length", dist, "label", strid) {
    if(-In.Nn > 0) {
      //Clamp the extinction color to prevent surfaces from being
      //100% color filters or absorb 100% of light
      extColor[0] = max(0.05, min(0.95, extColor[0]));
      extColor[1] = max(0.05, min(0.95, extColor[1]));
      extColor[2] = max(0.05, min(0.95, extColor[2]));

      dist *= 0.0328084;

      ci[0] += exp(-extinction * dist * (1-extColor[0])) * hitci[0];
      ci[1] += exp(-extinction * dist * (1-extColor[1])) * hitci[1];
      ci[2] += exp(-extinction * dist * (1-extColor[2])) * hitci[2];
    }
    else {
      ci += hitci;
    }
  }
  cRefrac += ci; // no mult by Kt * kt to increase weight of
     surviving rays
}

/*** SET PROPER EYEMAP (CAPTAIN VS CROWD) ***/
float eyeMap = 0;
    if (sEyeMap != "" && sEyeMapGang){
  if (gangMember < 1){
    eyeMap = texture(sEyeMap, ss, tt);
  }
  else{
    eyeMap = texture(sEyeMapGang, ss, tt);
  }
}

/**** SUBSURFACE (SSS) ****/
/*** INNER SSS ***/
color cIn = sToL(sssInner);
if (overallDiffuse != ""){
  cIn = sToL(color texture(overallDiffuse, ss, tt));
    if (sEyeMap != "" && sEyeMapGang != 0){
```

```
    cIn = mix(cDiffuse, cIn, eyeMap);
  }
}
else{
  cIn = sToL(sssInner);
}

/*** OUTER SSS ***/
color cOut = sToL(sssOuter);
if (overallDiffuse != ""){
  cOut = color sToL(color texture(overallDiffuse, ss, tt));

  //As long as it's not the Flyboy Bell or Tentacle
  if (objid == 4){
    if (eyeMap < 0.5){
      //For any dark areas, show more of the glow color
      cOut += sssHighlight * sssHLMin;
    }
    else {
      //For any light areas, brighten and show the color map
      cOut += sssHighlight * sssHLIntensity;
    }
  }
  //If it is the Flyboy Bell
  else {
    //for any light areas, brighten and show the color map
    cOut += sssHighlight * sssHLIntensity;
  }
}
else{
  color cOut = sToL(sssOuter);
}

color minC = 0;
color maxC = 1;

normal Nfv = faceforward (Nn, In);

/*** VELVET ***/
vector H;               /* Bisector vector for Phong/Blinn */
vector Ln;              /* Normalized vector to light */
float cosine, sine;
float nDot = smoothstep(sssStepBottom, sssStepTop, normalDot);
normalDot = nDot;
color cSheen = 1;
color outerEdge = mix(minC, maxC, clamp(nDot, 0, 1));
vector lightView;

float backscatter = 0;
float roughness = 0.1;
color velvet = 0;
```

```
vector lv = 0;
color llum = 0;
float ladd = 0;
velvet = 1.0 - (difCalc * (1.0 - outerEdge));

color edgeCombine = ((sssEdgeMix * outerEdge) + ((1.0 - sssEdgeMix) *
    (velvet)));
color sssFinal = mix(cOut, cIn, edgeCombine[0]) * sssMix;

color cRefracLift = color (0.0178, 0.1604, 0.3124);

if (raydepth > 0){
  sssFinal = 0;
  cSpec = 0;
}
else{
  cRefrac = max(cRefrac, cRefracLift *0.1);
}

/*** SET LABEL FOR PRIMVARS ***/
string lbl = "";
rayinfo("label", lbl);

/**** PIRATE REFRACTIONS ****/

/*** BELL (PRIMVAR 0) ***/
if (overallDiffuse != "" && objid < 0.1){
  //Refraction outside of the bell
  if (raydepth == 0) {
    //Blue from environment kills reds, lift them by moving channels
    cRefrac[0] = mix(cRefrac[0], cRefrac[2], eyeMap);
    cRefrac[1] = mix(cRefrac[1], min(cRefrac[1] * 1.8, 1), eyeMap);
    cRefrac *= sToL(color texture(overallDiffuse, ss, tt));
  }
  //Refraction of bell inside of the bell
  if (lbl == "0" && raydepth == 2){
    cRefrac = mix(cRefrac, sToL(color texture(overallDiffuse, ss,
       tt)), eyeMap);
  }
}

/*** TENTACLE (PRIMVAR 1) ***/
if (overallDiffuse != "" && objid > 0.5 && objid < 1.5){
  //Refraction of tentacle inside of the bell
  if (lbl == "0" && raydepth == 2){
    //Blue from environment kills reds, lift them by moving channels
    cRefrac[0] = mix(cRefrac[0], cRefrac[2], eyeMap);
    cRefrac[1] = mix(cRefrac[1], min(cRefrac[1] * 1.8, 1), eyeMap);
    cRefrac = mix(cRefrac, sToL(color texture(overallDiffuse, ss,
       tt)), eyeMap);
  }
```

```
  //Refraction of tentacle outside of the bell
  if (raydepth == 0){
    cRefrac = mix(cRefrac, sToL(color texture(overallDiffuse, ss,
       tt)), eyeMap);
  }
}

/*** BRAIN (PRIMVAR 2) ***/
if (overallDiffuse != "" && objid > 1.75 && objid < 2.5){
  //Refraction of brain inside of the bell
  if (lbl == "0" && raydepth >0){
    //Blue from environment kills reds, lift them by moving channels
    cRefrac[0] = mix(cRefrac[0], cRefrac[2], eyeMap);
    cRefrac[1] = mix(cRefrac[1], min(cRefrac[1] * 1.8, 1), eyeMap);
    cRefrac = mix(cRefrac, sToL(color texture(overallDiffuse, ss,
       tt)), eyeMap);
  }
  //Refraction of brain outside of the bell
  if (raydepth == 0){
   cRefrac = cRefrac * sToL(color texture(overallDiffuse, ss, tt));
  }
}

/**** FLYBOY REFRACTIONS ****/

/*** BELL (PRIMVAR 3) ***/
if (overallDiffuse != "" && objid > 2.75 && objid < 3.5){
  //Refraction outside of the bell
  if (raydepth == 0) {
    //Make bell more yellow
    eyeMap = clamp(eyeMap, .05, 1);
    cRefrac[0] = mix(cRefrac[0], cRefrac[2] * 1.5, eyeMap);
    cRefrac[1] = mix(cRefrac[1], cRefrac[2] * 1.8, eyeMap);
    cRefrac = mix(cRefrac, sToL(color texture(overallDiffuse, ss,
       tt)), eyeMap);
  }
  //Refraction inside of the bell
  if (lbl == "3" && raydepth == 2){
    cRefrac = mix(cRefrac, sToL(color texture(overallDiffuse, ss,
       tt)), eyeMap);
  }
}

/*** SKIRT (PRIMVAR 4) ***/
if (overallDiffuse != "" && objid > 3.5 && objid < 4.5){
  //Refraction of skirt inside the bell
  if (lbl == "3") {
    //Make sure you capture the skirt part in between the two bell
       layers
    if (raydepth == 1 || raydepth == 2){
```

```
        cRefrac = mix(cRefrac, sToL(color texture(overallDiffuse, ss,
            tt)), .5);
      }
    }
    //Refraction of skirt outside the bell
    if (raydepth == 0){
      eyeMap = clamp(eyeMap, 0, .4);
      cRefrac = mix(cRefrac, sToL(color texture(overallDiffuse, ss,
          tt)), eyeMap);
    }
}

/*** TENTACLE (PRIMVAR 5) ***/
if (overallDiffuse != "" && objid > 4.5 && objid < 5.5){
    //Refraction of tentacle outside of skirt only (due to opaque
        skirt)
    if (raydepth == 0){
      //Make tentacals more yellow
      cRefrac[0] = mix(cRefrac[0], cRefrac[2] * 1.5, eyeMap);
      cRefrac[1] = mix(cRefrac[1], cRefrac[2] * 1.8, eyeMap);
      cRefrac = mix(cRefrac, sToL(color texture(overallDiffuse, ss,
          tt)), eyeMap);
    }
    else{
      cRefrac = cRefrac;
    }
}

/*** SPOTS GLOW COLOR ***/
color glowSpots = spotMap * (cRefrac * 2);

/***** FINAL MIXING *****/
color cFinal = cDif + cSpec + cAmb + cReflec + cRefrac + sssFinal +
    glowSpots;
cFinal *= fTrans;
cFinal = clamp(cFinal, minC, maxC);

/*** ZDEPTH ***/
_zdepth = getzdepth(P);
_sdepth = getsdepth(P);

/*** OUTPUT ***/
_occ = 1.0 - occ;
_sss = sssFinal * sssMix;
_diffuse = cDif;
_specular = cSpec;
_reflection = cReflec;
_refraction = cRefrac;
_beauty = cFinal;
_mask = Os;
_ambient = cAmb;
```

```
    _ssslight = sssFinal * difCalc;
    _difCalc = difCalc;
    _mScar = fRefractControl;
    _spotglow = glowSpots;

    Oi = fTrans;
    Ci = cFinal;

    }
}
```

References

[1] Pauline Pirates & Privateers Blog. Post tools of the trade: Stepping out (november 18, 2009). http://paulinespiratesandprivateers.blogspot.com/2009_11_01_archive.html, Accessed: February 2015.

[2] Hyper Physics: Website Resource for High School Students. Index of refraction table. http://hyperphysics.phy-astr.gsu.edu/hbase/tables/indrf.html, Accessed: February 2015.

[3] Deryck Foster. "sea venture" painting. http://repeatingislands.files.wordpress.com/2013/06/sea-venture.jpg, Accessed: February 2015.

[4] Henrick Wann Jensen Stephen R. Marchner Marc Levoy Pat Hanrahan. A practical model for subsurface light transport. *Proceedings of SIGGRAPH'2001*, 2001.

[5] Thomas Jordan. Reflections and refractions in "finding nemo". *RenderMan, Theory and Practice*, pages 119–124, 2003.

[6] AC Kevin. Reefers anonymous website. http://i411.photobucket.com/albums/pp198/AlohaCorals/18FEB13_BluePotters_35i_YVB1_003.jpg, Accessed: February 2015.

[7] Douglas Klug. Purple striped jellyfish photograph. http://calphotos.berkeley.edu/imgs/512x768/0000_0000/1112/1688.jpeg, Accessed: February 2015.

[8] David M. Laur. Programmable ray tracing. *RenderMan, Theory and Practice*, pages 13–30, 2003.

[9] Siteseen Ltd. Three striped damsel fish. http://www.educationalresource.info/tropical-marine-fish/28-three-striped-damsel-fish.htm, Accessed: February 2015.

[10] National Museum of the US Air Force. P51 mustang. http://www.nationalmuseum.af.mil/shared/media/photodb/photos/071024-F-1234S-008.jpg, Accessed: February 2015.

[11] Pixar. Grand tour part 4: Character. http://rendermansite.pixar.com/view/TGT_Character, Accessed: February 2015.

[12] Pixar. Primitive variables. http://rendermansite.pixar.com/view/how-to-primitive-variables, Accessed: February 2015.

[13] Pixar. Renderman. http://renderman.pixar.com/view/p-renderman, Accessed: February 2015.

[14] Pixar. Secondary outputs and aovs. http://rendermansite.pixar.com/view/secondary-outputs-and-aovs, Accessed: February 2015.

[15] Pixar. Slim file format. http://renderman.pixar.com/resources/current/rms/slim_File_Format.html, Accessed: February 2015.

[16] Pixar. Subsurface scattering. http://rendermansite.pixar.com/view/subsurface-scattering, Accessed: February 2015.

[17] Wikipedia. Photograph of roy marlin voris. http://en.wikipedia.org/wiki/Roy_Marlin_Voris, Accessed: February 2015.

[18] Roy Winkelman. "refraction of pencil in cup of water" photo showing how a pencil appears to be bent when entering a cup of water due to refraction. http://etc.usf.edu/clippix/pix/refraction-of-pencil-in-cup-of-water_medium.jpg, December 2012.

[19] Zoravar. Camouflage face-painted azerbaijani special forces showing off their israeli-made tavor assault rifles, communication gear, "cool gloves" and other goodies during a military parade in baku. http://img181.imageshack.us/img181/104/am5af4.jpg, Accessed: February 2015.