Modeling and Rendering a Nighttime Bear and River Scene

David Hyde, Lingxiao Li, Tushar Paul

June 6, 2016

Abstract

For our final project, we created a scene with a scene with a bear in a riverbed at night. We wanted to highlight the fur of the bear, so we implemented the Marschner model, which consists of the highlight lobes. We modeled our scene in Maya and 3ds Max, including an intricate fur geometry consisting of about 550,000 hair strands, where each strand consists of 7 cylinders. We also used a fluid simulation in Maya to generate our water geometry. Finally, we used volumetric scattering to create a "glowing" firefly. Various effects and artistic direction are used throughout to achieve the most visually appealing result.

1 Introduction

For our final project, we wanted to create a physically-based rendering of a bear in a stream. We are inspired by the photograph below:

Figure 1: Inspiration for our scene

During our implementation, we decided to create a night scene instead of a daytime one in order to achieve the greatest visual drama in the scene and to best emphasize the foci of our project efforts. We also included a firefly to add a separate, whimsical, visually pleasing element to the scene. When considering all the elements we needed to render, we realized the scene is highly complex and intricate. For instance, there are both solid (bear, ground, rocks, trees) and fluid (river water) features, and these

features are coupled (e.g. water flows around the bear's legs). Additionally, there are very fine-scale features, such as hundreds of thousands of hairs, consisting of millions of cylinder primitives on the bear. Finally, in order to render fireflies correctly, we need to use advanced rendering techniques such as volumetric lighting in order to get the desired "glow" effect.

Due to these complex features, this project presented challenges for both modeling and rendering. On the rendering front, a naive model for hair (for example, a simple diffuse model, as shown in Appendix A) looks flat and unappealing, as it misses many of the often-subtle highlights present in creature hair that are required to make hair look natural.

Similarly, the human eye is also very skilled at identifying whether water is "real;" thus, great care must be taken to reproduce accurately simulated and rendered water. Additional rendering complications arise once water is introduced into the scene, such as the visual differences between wet and dry hair.

We address all these modeling and rendering challenges in the sections that follow. Descriptions of the contributions of the individual team members, as well as a reproduction of our final rendered image, are included.

2 Hair Modeling and Rendering

2.1 Hair Modeling

We modeled the hair for our bear using 3ds Max. In 3ds Max, it is possible to place "guide hairs" (splines placed at certain positions and orientations, anchored to the surface mesh) and use these guide hairs to generate an appealing, dense fur consisting of arbitrarily many hair strands. We place on the order of one hundred guide hairs on the bear. The guide hairs can be styled or "groomed" in 3ds Max using the software's built-in tools. These styling tools include changing hairs' lengths, straightening or curling hair, and making hair stand up or lay flat relative to the surface mesh. Given a collection of guide hairs, 3ds Max uses these to automatically generate a triangle mesh of hairs (however many the artist specifies). 3ds Max also lets us control the spline properties of the hair; for example, we specified that each hair strand should be comprised of seven linear segments.

As 3ds Max generates a "connected" triangle mesh, rather than a disjoint collection of line segments or cylinders (as the hair geometry actually appears), some work is needed to be done to convert this resultant geometry into a format appropriate for rendering in `pbrt`. We wrote a tool that parses the hair triangle mesh from 3ds Max and creates a `pbrt` file with each hair strand segment as its own cylinder. With this tool, we can customize the parameters of the hair cylinders that we generate. For example, while we used a uniform radius for each hair strand segment cylinder, we could have tweaked the tool so that the cylindrical segments taper off in radius as they go from the surface of the bear to the end of the hair strand. In theory, this would give a more realistic hair geometry; however, due to the vast number of strands in our scene and the minute dimensions of each strand, this tapering effect was not necessary for us to achieve a plausible hair geometry.

2.2 Hair Rendering

A major part of our final project involved rendering our hair correctly. We wanted to use a physically-based method that would accurately create the complex highlights seen in natural hair and fur. During our literature search, we looked at the Kajiya Kay and Marschner models [1] [2]. After reading the papers, it became clear that the Marschner model would give us the complicated highlights we were looking for, at the cost of implementation complexity.

2.2.1 Marschner Model

The Marschner model consists of three main terms: R, TT, and TRT. In these terms, R stands for *reflection* and T stands for *transmission*. The following diagram from [2] explains these terms in the context of the cross-section of a hair fiber:

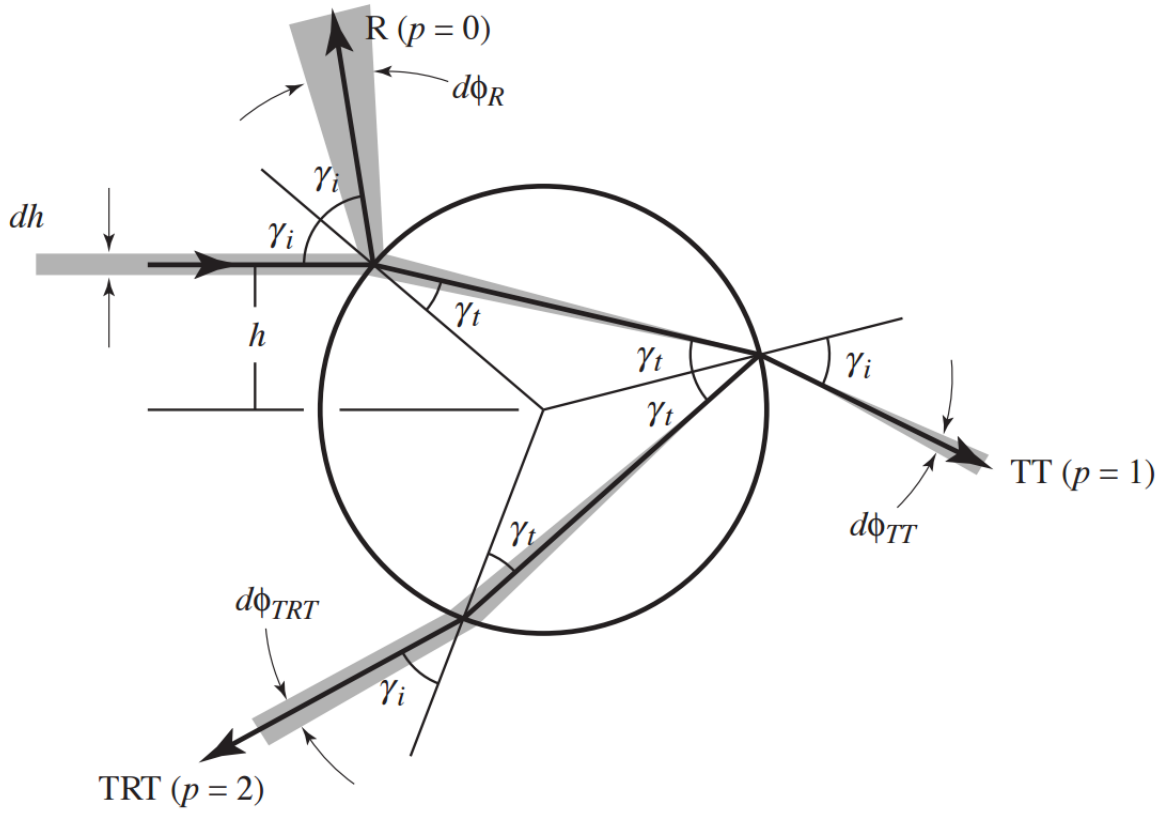

Figure 2: R, TT, and TRT terms of a circular cross-section of a fiber

As the diagram shows, these terms arise from the interaction of an incoming ray into the hair fiber. The hair *reflects* some light, which is the $p = 0$ or R case, *transmits* twice, which is is the $p = 1$ or TT case, or *transmits*, *refracts*, and then *transmits*, which is the $p = 2$ or TRT case. The angles γ_i and γ_t are a function of the incidence of the incoming ray, and the index of refraction η of the hair fiber. The TT and TRT terms will attenuate light as it transports through the hair, giving the TT and TRT terms the color of the hair. Because the R term does not go through the hair, it will be the same color as the incoming light. For brevity, we omit the equations that describe how to find these terms, and instead refer the reader to [2].

The above description presents a simple model in two dimensions. However, we can generalize the model to 3D using the Bravais index, as described in the paper. To do so, instead of using η, we use η' and η'', which are derived from the longitudinal angle of incidence θ of the ray using the following formulas:

$$\eta'(\theta) = \frac{\sqrt{\eta^2 - \sin^2\theta}}{\cos\theta}, \eta''(\theta) = \frac{\eta^2 \cos\theta}{\sqrt{\eta^2 - \sin^2\theta}}$$

Finally, we needed to account for the fact that hair is not actually a cylinder with a circular cross-section, but instead has an elliptical cross section. Thus we need to account for the eccentricity of the hair start. As the paper describes, we approximate the eccentricity by using η^* instead of η, where η^* is a function of eccentricity a and the half-angle ϕ_h, as described in [2].

Implementation of the Marschner model was quite complicated, and we initially struggled with the concepts in the paper. Our main confusion was about whether we would have to create a new subsurface scattering integrator, or if we could instead just use a standard `BSDF`. We missed the fact that the paper approximates the hair fiber to be very thin. Therefore, we were able to consider each ray as coming from the same point and integrate over a sphere, which was very similar to a standard `BSDF`.

After we passed our conceptual hurdles with understanding the Marschner model, we ran into several bugs in our code. Our initial implementation did not handle several edge cases (that were not explicitly defined in [2]) that caused numerical blowup, leading to extremely bright "speckles" in the fur, as shown in Appendix A.

We realized that for the $p = 2$ term, we needed to regularize the angle ϕ to some offset of 2π due to the polynomial approximation from the paper we were solving. Otherwise, the N_{TRT} term would not properly smooth out/dampen the caustics. Once we normalized all our angles, we were able to remove these speckles and get some realistic highlights.

One difference between our implementation of model and that described in the paper is that the paper integrates each term over the entire sphere, whereas we only integrate over a particular hemisphere per term. That is, we integrate over the same hemisphere as the incoming light ray for the R and TRT terms, but integrate over the opposite hemisphere for TT term. We originally thought that if we used a `BTDF` for the TT term, `pbrt` would automatically integrate over the correct hemisphere. However, we found that this was not exactly the case, so we had to override the various `sample` functions to sample light in the correct hemisphere.

2.2.2 Wet Hair

We were very intersted in rendering wet hair (e.g. the portions of the bear's hair that are near to or submerged beneath the water line) differently than dry hair, to emulate the real world. Conceptually, adapting the Marschner model to render wet hair involves both modifying the geometry of the hair (e.g. clumping some of the fur) and using a different rendering scheme that demonstrates the brighter specular area of the wet hair as well as darker shadows due to internal reflections, according to [3]. Owing to time constraints, we used the same Marschner model for wet hair except for changing the underlying fur texture to make the hair look darker with more specular highlights. The new wet hair texture is modified by applying darkening filters and adding more highlights in Photoshop.

Figure 3: Dry hair(left) vs. wet hair (right).

The actual diffuse color used in a Lambertian `BxDF` is the result of an interpolation of the dry hair and wet hair, based on the height of the intersection point in the world space. The hair on the bear's legs, and the lowermost portions of its body, subtly show our wet hair effect, in contrast to the dry hair on the rest of the bear.

3 Water Simulation and Rendering

3.1 Modeling and Simulation

The physically-based water surface is a key component of our overall scene. To generate physically-based water, we needed to use fluid simulation. Fluid simulation is a rich field of study; for our project, we wanted a fluid simulation technique that combined physical accuracy with modeler-friendly efficiency. Grid-based Navier-Stokes solvers are computationally demanding and more difficult to artistically direct; particle-based fluid techniques are cheap but suffer from less realistic or visually plausible results. Thus, in order to achieve our goal of combining accuracy and efficiency, we used Maya's built-in Bifröst fluid simulation framework for our project.

Bifröst uses the FLIP method to simulate fluid. FLIP largely resembles particle-based methods, but it has many nice additions, such as mass conservation, that make it a strong choice for graphics applications. Using FLIP methods for incompressible flow in graphics was pioneered by Robert Bridson in [4] and has remained very popular since then. Among the benefits of FLIP, "The FLIP method and its variants achieve a near total lack of numerical diffusion in the transport stage of the fluid simulation, since all quantities are advected on particles as opposed to a grid" [5]. For our purposes, FLIP is also a good choice due to its tight integration into Maya via the Bifröst framework [6]. Due to this integration, we are able to use artistic license with the fluid simulation results.

Within Maya, we added several rectangular prisms that we set as "emitters" from the Bifröst menu. These emitters were placed strategically to get visually appealing results (we had to try several variations in order to get the desired amount of water in different places in the river). We added the other major components of our scene – rocks, the bear's body, and the ground planes – as "colliders," meaning that any FLIP particles bounce off those objects rather than penetrate them. Finally, since particles still managed to escape our scene geometry, we added a "killplane" that deletes any particles that fly too far below the scene geometry. (Adding the killplane provided noticeable performance increases for the simulation.)

The particle results we obtained from Bifröst were aesthetically pleasing. However, to create results usable in `pbrt`, we had to generate a mesh based on the fluid particles, then export that mesh to a `pbrt`-compatible format. Bifröst has a built-in tool to generate a fluid mesh from particle results; however, we found that the fluid meshing tool performed poorly, despite numerous attempted configurations and relatively high-resolution particle results. Specifically, the automatically-generated mesh often contained noticeable holes. Moreover, some regions of the river seemed to be completely ignored by the meshing tool, despite many particles being present. Finally, some small areas of the mesh (especially near collisions, like around the bear's legs) contained folds and self-collisions, which made the rendered water too opaque and less realistic. To surmount these problems, we manually repaired some features of the mesh (e.g. deleting faces where folds occurred) and instanced copies of our foreground fluid mesh in the back of the river (where the meshing tool seemed to ignore). These approaches allowed us to successfully overcome the deficiencies of the Bifröst meshing tool.

After the Bifröst simulation and post-processing, we attempted to render our mesh in our scene. We found that meshing artifacts were overly apparent (it was easy to tell that the water was a triangulated surface). Thus, we returned to Maya and used the mesh smoothing tool to perform one iteration of subdivision on our fluid mesh. This made the mesh resolution sufficiently fine that mesh artifacts were no longer apparent in our final result.

Once we obtained our final fluid mesh, we exported it from Maya as an `obj` file, which we then converted to a triangle mesh in `pbrt` format using the `obj2pbrt` tool.

3.2 Rendering

The human eye is skilled at determining whether food or drink is "real." Thus, realistic rendering of water is a significant challenge for graphics. We were inspired by several graphics papers for water rendering: [8], [9], [10], and [11]. Ultimately, due to time constraints, our model for water rendering was relatively rudimentary; however, we were able to achieve artistically pleasing results and could easily substitute a more rigorous rendering model in the future.

In `pbrt`, our water surface is represented by instances of the `TriangleMesh` class. The core of our rendering technique is shading these meshes using `pbrt`'s `mix` material. For our mixture, we combine `pbrt`'s `glass` material type with the `uber` material type. The former provides refraction and specular highlights, while the latter provides colored diffuse reflection and translucency. We needed the `glass` material to obtain refraction since the `uber` material does not refract light opacity values less than one (note: we used the physically correct index of refraction for water, 1.33). Similarly, we needed the `uber` material since the `glass` material does not support color or opacity properties. By combining these two materials and carefully setting each of their properties, we obtained a physically plausible water material.

When we rendered our scene with the water included, we noticed that the water surface seemed to lack a certain vitality or turbulence. The water seemed too smooth given that the mesh represents an active river and a large creature is presumably wading through the water (causing significant deformations in the flow). Furthermore, we were not satisfied with the specular reflections of the moon light coming off the water. To enhance the visual quality of the water, without compromising the structure of the physically-based mesh, we applied a bump map to the water. The bump map affects computed normals for points on the face of each triangle, which produces noticeably different reflections in the overall

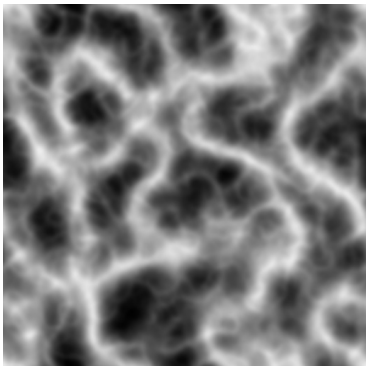

Figure 4: The bump map applied to the water meshes in our scene. The bump map is scaled and deformed to fit onto each face of each triangle mesh. Using the bump map for our water surface yields visually improved turbulence and reflections.

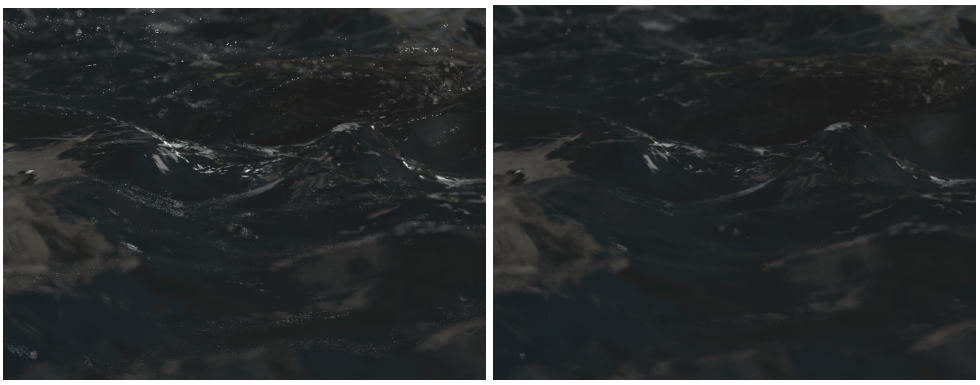

Figure 5: Before (left) and after (right) applying a 1 pixel-wide filter to our output image. The filter successfully eliminates small, spurious specular highlights (white) without impacting the overall image.

rendering. We show in Figure 3.2 the bump map we used, which is stretched to fit onto each face of each triangle mesh.

We observed one additional problem with our water surface. The raw output image from `pbrt` had very small (one or several pixels wide) specular highlights on the water surface that appeared to be spurious. These specular highlights came from the two area lights in our scene interacting with the glass component of our mixed material for the water. We surmise these were due to improper normal information for the triangle meshes or overly strong area lights. While further adjusting the area light settings in our `pbrt` scene is one valid approach for alleviating this problem, due to time constraints, we found a rudimentary solution that successfully fixed the spurious specular highlights. We applied the Minimum filter in Photoshop to the `pbrt` output image with a radius of 1 pixel. This means that for each pixel i in our final image, we look at the one-ring of pixels around i and set the value of i to be the minimum of i and the minimum value over the one-ring. The qualitative effect of this single-pixel-wide filter, as described in [12], is "applying a choke (erosion)—shrinking white areas and spreading out the black areas." We show in Figure 3.2 a comparison of a small portion of our image before and after applying this filter, demonstrating the elimination of spurious, tiny specular highlights. In the end, the effect was quite minor, but helped the image be more well-composed.

4 Scene Design and Construction

We designed and constructed the scene, including texturing and lighting, in Maya. The scene geometry is composed of: several small rocks on the left near the camera and several large stony edifices on the far side of the river; five trees spread width-wise and depth-wise throughout the scene; several small bushes and rocks in the background; two planes composing the visible portion of a skybox; and the bear

and water meshes. We obtained the models from TurboSquid.com.

The geometry, except for the bear and skybox planes, came textured (however, we needed to manually make a number of edits in our `pbrt` files to correctly process the alpha masks that came with the models' textures). For the bear, we used a very low-resolution texture on the body of the bear that at least was brown and variegated (this texture is almost entirely obscured by the rendered fur). For the skybox, we found an image at [13] and used a Python cube map script (adapted from a StackOverflow solution) to generate cube map pieces from the equirectangular source image.

We use one spherical area light to illuminate the skyboxes; this area light is placed far behind and above the main portion of the scene. We also use one area light as the "moon," placed above and to the right of the bear. We experimented with using an environment map for our scene, but we found that the environment map drastically reduced the visibility of the different reflection terms in the bear's fur. Finally, as discussed in the next section, we use a global scattering volume to simulate a fog effect.

One trick we discovered when exporting from Maya to `pbrt` was that "freezing" the transformations in Maya allowed us to obtain correct transformations when objects were rendered in `pbrt`. For exporting to `pbrt`, we used a combination of the `pbrtMayaPy` Maya plugin [14] and Maya's built-in `obj` exporter tool (which also exports a corresponding `mtl` file for the texture information). For `obj` output, we use `obj2pbrt` to finally convert into `pbrt` format.

5 Miscellaneous

5.1 Camera

We use a perspective camera with the effect of depth of field. Depth of field centers the audience's attention to the bear and the firefly, while blurring out the less refined objects in the background (e.g. the trees and the riverbank). We achieve depth of field by setting a non-zero lens radius for the perspective lens in our `pbrt` scene file.

5.2 Volumetric Fog

Since the lighting of the scene is mainly due to a spherical area light (the moon) above the bear, the whole scene appears to be too bright to be real. Also, given the fact that there is usually a layer of fog above a river (due to water evaporation), we chose to add a global scattering volume whose intensity decreases exponentially with respect to the height. This effect simulates a global fog in the environment and helps make the lighting for the scene more realistic.

5.3 Surface Integrator

With the moon light being the only main light in the scene, we used `pbrt`'s direct lighting surface integrator as our surface integrator of choice. Direct lighting gives a good balance of rendering time and result quality, especially in the shadowy regions underneath the bear.

During the course of our experimentation, we tested other integrators that provide global illumination, such as path tracing and the metropolis renderer. However, the resulting images often contained lots of noise due to the poor sampling of the huge number of light transport paths. This issue is especially magnified in an open-world scene with large light sources since the number of paths that needs to be sampled has significantly increased. On the other hand, we also want to keep the lower body of the bear darker to emulate the wet hair effect. While the metropolis renderer provided sufficiently de-noised results, the global illumination added to the scene was displeasing. In the end, direct lighting provided the best performance while also giving us a pleasant overall look for our scene.

5.4 The Daring Firefly

To add liveliness to the image (and also make the bear less lonely), we put a glowing firefly close to the nose of the bear. We find the mutual gazing between the firefly and the bear adds lots of meaning into the image. In the scene description, the firefly itself is simply a spherical area light. To create the glowing aura of the firefly, we made use of `pbrt`'s `volumegrid` that lets us define the density of the volumetric substance over a 3D grid. The grid is generated based on each cell's distance to the center, with a quadratic decay.

When creating the firefly, we originally had more than one firefly. However, the image looked unnatural because we didn't have the volumetric scattering effect for the fireflies. Unfortunately, once we figured out how to use the volumetric scattering effect, we did not have enough time to place all the fireflies, and settled on a single firefly. A sample image with multiple fireflies, without the volumetric lighting, is in Appendix A.

6 Performance Considerations and Timing

To accelerate the rendering of our scene, we used a bounding volume hierarchy with up to seven primitives allowed per node in the tree. We use the default surface area heuristic for partitioning the primitives when building the tree. We found that the bounding volume hierarchy greatly accelerated our results; we did not compare to other acceleration structures, such as kd-trees, since we achieved satisfactory performance with our bounding volume hierarchy.

We ran our renders on a local machine with two ten-core Intel Xeon E7-2860 processors, each core running at 2.27GHz. With this processing power, we were able to generate our final render at 4K Ultra HD resolution (3840px $\times$ 2160px) in 3675.0 seconds (just over one hour). Our final render was run with 64 samples per pixel; we found that further increasing samples per pixel yielded no noticeable improvements in the output image.

7 Individual Contributions

Although this project was a collaborative effort, we had a "split" of tasks assigned among the group members. David handled the modeling of the bear, fur, rocks, trees, and the fluid simulation of the creek (including modeling, simulation, and rendering of the water). Because the Marschner model was so complex, Tushar and Lingxiao collaborated on implementing and debugging the model. Lingxiao implemented the firefly, and Tushar implemented the wet hair. All members contributed with parameter tuning and artistic judgments, and all members collaborated in writing the final report.

8 Final Image

9 Conclusions and Future Work

In conclusion, we successfully implemented our bear scene, including overcoming various modeling and rendering challenges. We implemented the Marschner model for hair to render the bear's fur, and we were able to get nice highlights on the body of the bear. We implemented a firefly using volumetric scattering to create an aura. We used a fluid simulation to create water geometry, used a mixture of materials and bump maps to render the water, and created a convincing river scene in Maya. Finally, we used depth-of-field and volumetric fog effects to add realism to the scene.

Source code, scene files, and high-resolution images for our project and final renders are all available upon request.

10 Acknowledgments

We would like to acknowledge Pat Hanrahan and Mike Mara for guiding us on the implementation of our Marschner Model. We would also like thank our friends for critiquing intermediate images and guiding us towards the artistic direction of the final shot.

11 References

References

[1] Kajiya, James T., and Timothy L. Kay. "Rendering fur with three dimensional textures." ACM Siggraph Computer Graphics. Vol. 23. No. 3. ACM, 1989. http://www.cs.virginia.edu/~mjh7v/bib/Kajiya89.pdf

[2] Marschner, Stephen R., et al. "Light scattering from human hair fibers." ACM Transactions on Graphics (TOG). Vol. 22. No. 3. ACM, 2003. http://graphics.stanford.edu/papers/hair/hair-sg03final.pdf

[3] Gupta, Rajeev, and Nadia Magnenat-Thalmann. "Interactive rendering of optical effects in wet hair." Proceedings of the 2007 ACM symposium on Virtual reality software and technology. ACM, 2007. http://nishitalab.org/user/witawat/resources/papers/wetHair_pg2012.pdf

[4] Zhu, Yongning, and Robert Bridson. "Animating sand as a fluid." ACM Transactions on Graphics (TOG) Vol. 24 No. 3. ACM, 2005.

[5] Seymour, Mike. "The Science of Fluid Sims." fxguide, 2011. https://www.fxguide.com/featured/the-science-of-fluid-sims/

[6] "Bifröst Overview and Concepts." Maya User Guide, 2014. https://knowledge.autodesk.com/support/maya/learn-explore/caas/CloudHelp/cloudhelp/2015/ENU/Maya/files/GUID-43D655E2-45A3-4FD8-847C-6622AF22995C-htm.html

[7] "Create Hair, Fur, Or Instanced Geometry." Maya User Guide, 2016. https://knowledge.autodesk.com/support/maya/learn-explore/caas/CloudHelp/cloudhelp/2016/ENU/Maya/files/GUID-C8B2F14C-192B-40D9-BDC2-5F5A87B34BC8-htm.html

[8] Enright, Douglas, Stephen Marschner, and Ronald Fedkiw. "Animation and rendering of complex water surfaces." ACM Transactions on Graphics (TOG) Vol. 21 No. 3. 736-744. ACM, 2002. http://kucg.korea.ac.kr/seminar/2002/src/pa-02-43.pdf

[9] Mitchell, Jason L. "Real-time synthesis and rendering of ocean water." ATI Research Technical Report 121-126. ATI, 2005. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.70.9112&rep=rep1&type=pdf

[10] Iglesias, Andres. "Computer graphics for water modeling and rendering: a survey." Future generation computer systems Vol. 20 No. 8. 1355-1374. 2004. http://www.sciencedirect.com/science/article/pii/S0167739X04001013

[11] Liang, Jianming, Gong, Jianhua, and Li, Yi. "Realistic rendering for physically based shallow water simulation in Virtual Geographic Environments (VGEs)." Annals of GIS, Vol. 21 No. 4. 301-312. 2015.

[12] Filter Effects Reference. "Photoshop Filter Effects Reference". N.p., n.d. Web. 07 June 2016. https://helpx.adobe.com/photoshop/using/filter-effects-reference.html#other_filters

[13] Mt Dale Comms Tower at Night. Flickr. Yahoo!, n.d. Web. 07 June 2016. https://www.flickr.com/photos/8380845@N06/17172435186/

[14] Volodymyrk. Volodymyrk/pbrtMayaPy. GitHub. N.p., n.d. Web. 07 June 2016.https://github.com/Volodymyrk/pbrtMayaPy

Appendices

A Intermediate Images

Figure 6: Bear with a simple diffuse model for rendering hair

Figure 7: Bear with speckling bug in Marschner model

Figure 8: Bear with multiple fireflies without volumetric scattering

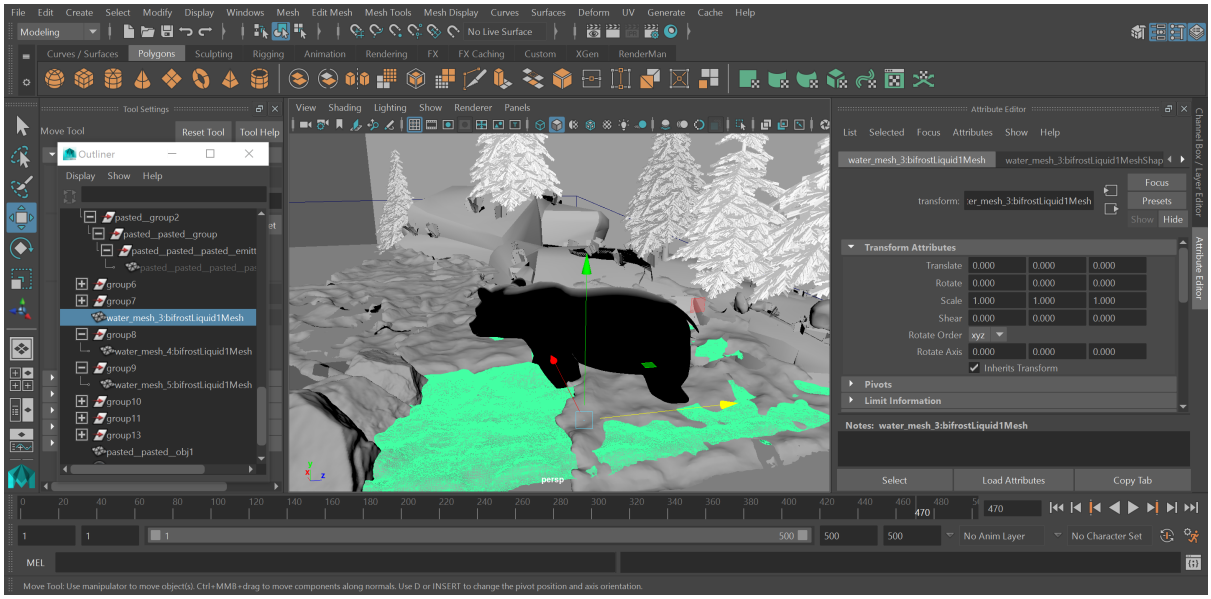

Figure 9: Arranging the scene in Maya. One of the fluid mesh components, in green, is selected.

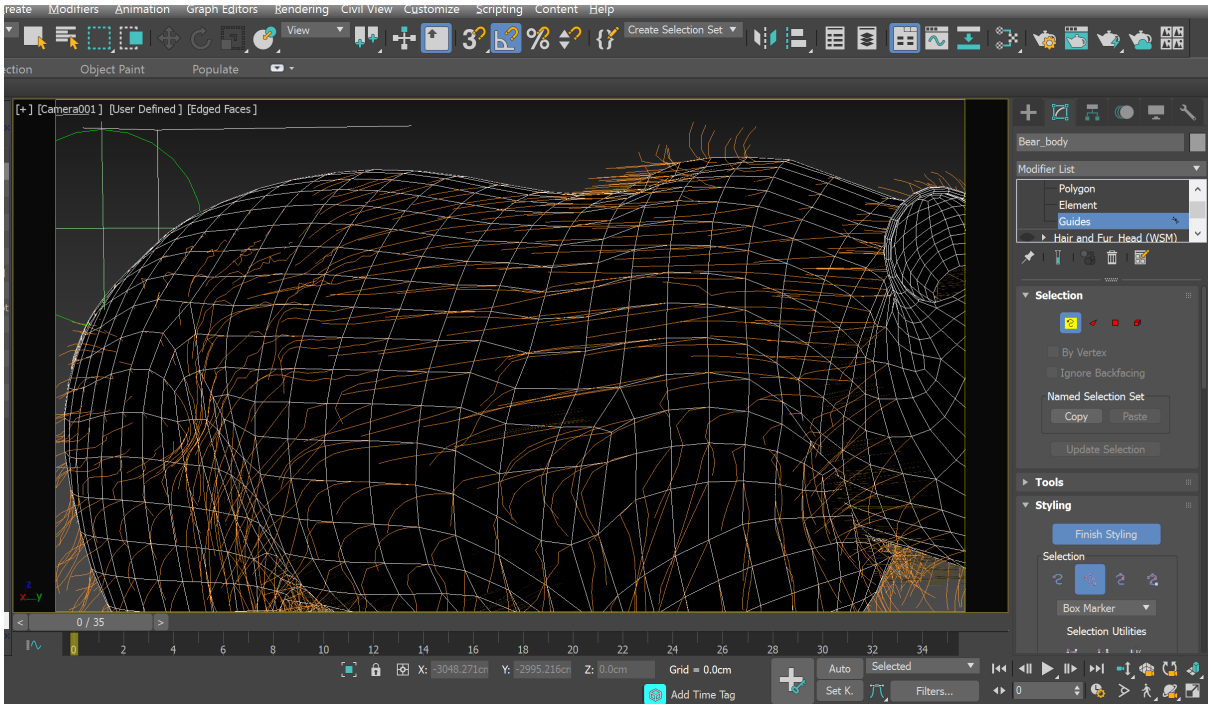

Figure 10: The guide hairs (orange) for our bear (black and white mesh) as seen in 3ds Max. The green crosshairs represent one of the grooming tools. The guide hairs in this image have been styled in an extreme way to emphasize the artist directability of the hair in 3ds Max. With these guide hairs, 3ds Max generates a dense fur mesh as seen in our final images.