CS 348B Report: Rendering Cotton Candy

Chenlin Meng, Hubert Teo, Jiren Zhu

June 12, 2018

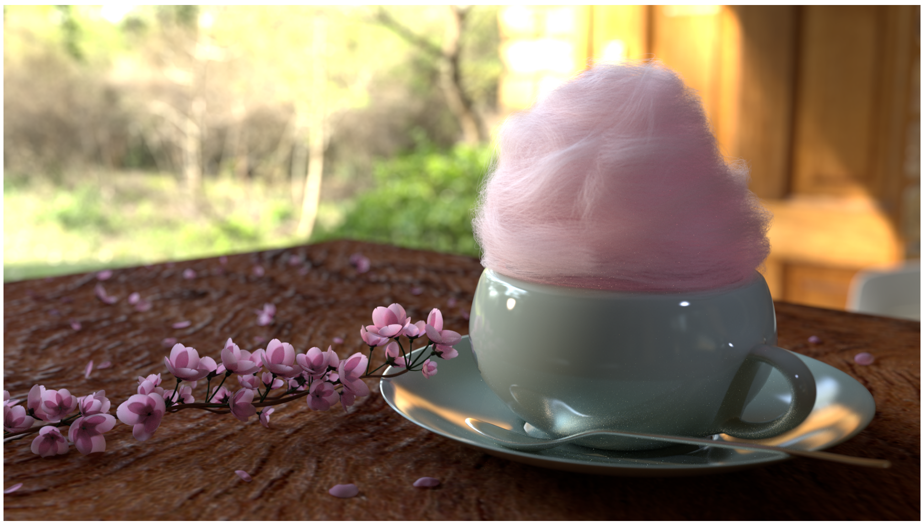


Figure 1: The final rendered image at 1920 × 1080, 4096 samples per pixel.

Who doesn't like pure sugar made ethereal and light? In this project, we chose cotton candy as our subject because it is an interesting material with not much precedent in rendering and a lot of room for experimentation.

1 On Cotton Candy

Cotton candy has an intricate internal structure that stems from the way it is produced. Molten suger is centrifuged out of tiny holes in a spinning extruder. As the sugar rushes past air, it cools rapidly, forming long, thin fibres that arrange themselves in a haphazard fashion. We are especially intrigued by the way that this structure influences the appearance of cotton candy. To investigate the visual phenomena that we should be focussed on modelling, we consulted many reference images of cotton candy.

Cotton candy displays a remarkable variety in appearance. Figure 2(a) shows that the internal structure of cotton candy is indeed formed by long, thin fibres. However, it also shows some significant fine structure and textural differences from the relatively uniform cotton candy in 2(c). Furthermore, 2(b) shows that

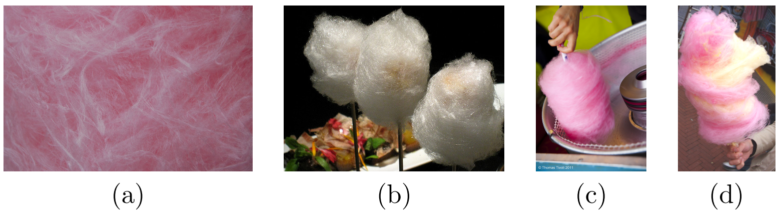


Figure 2: Reference images to illustrate the way light behaves in cotton candy. (a) https://www.flickr.com/photos/sparklykate/3943900977 (CC BY-NC 2.0), (b) https://www.flickr.com/photos/tannazie/3292364143 (CC BY-SA 2.0), (c) https://www.flickr.com/photos/67792682@N03/6171344221 (CC BY 2.0), (d) https://commons.wikimedia.org/wiki/File:Suikerspin.png (CC BY-SA 3.0)

cotton candy can also have specular highlights at certain positions. The relative intensity of these highlights depends on the fibre thickness: thicker fibres behave more like glass cylinders with specular highlights, while a dense distribution of thinner fibres results in a diffuse appearance.

Figure 2(c) is perhaps the most informative. Notice that the reflection of the light on the left is mostly white, whereas the indirectly-lit right half of the cotton candy is very saturated. This tells us that the color of cotton candy comes from transmission and absorption through the sheer number of glass-liked colored sugar fibres. The white appearance of the left side is due to reflection from the fibres, which are individually specular but appear diffuse in aggregate.

Figure 2(d) also shows that the way that cotton candy is twirled around the stick can have a great impact on its visual appearance, and that we should aim to have more large-scale structure in order to create visual interest.

2 Model

The driving question of our project is then: how can we represent cotton candy in a way that makes it possible to render? Previous approaches on 3D artist forums model cotton candy as diffuse gas volumes to achieve a cloud-like appearance but without the fibrous detail. There are also some that use procedural noise-controlled heterogeneous volumes to create the surface structure, but this is hard to control and does not give a particularly realistic result.

2.1 Direct representation

Our first attempt was to model all the cotton candy fibres directly. We conducted a few tests using Blender's modelling tools and the Cycles render engine to see if we could achieve the desired texture.

The first approach, in Figure 3(a), was to use Blender's hair particle system to comb extra-long hairs emitted on a single plane such that they bend around and form a hairball. We then applied a combination of the transmissive and a reflective hair shaders available in Cycles. This produced a ball that has the right characteristics on the macro level, but appears like synthetic fibres when inspecting it up close.

The second approach, in Figure 3(b), was to instance a fixed set of curves at

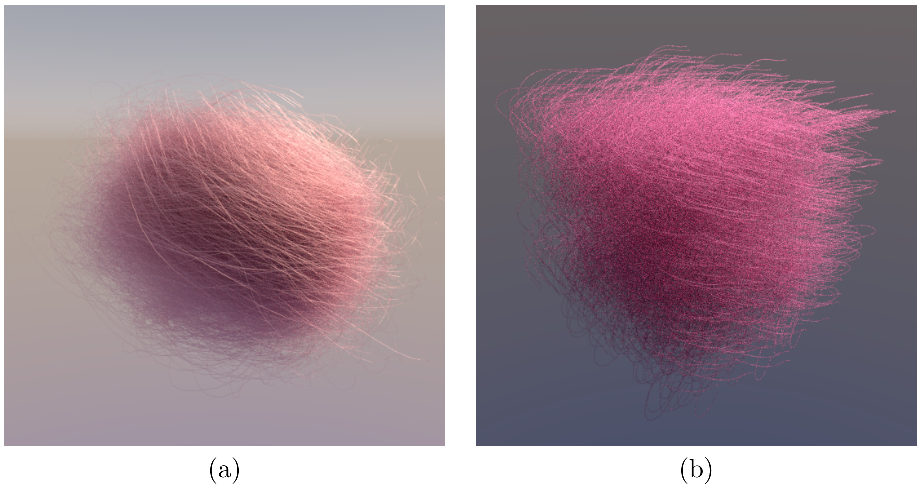


Figure 3: Two early test renders using Blender's modelling tools. Neither is convincing, but demonstrates that it is possible to render a large number of cylindrical curves.

random positions and with partially-randomized orientations so that the curves align in the same general direction. This makes it easier to control the surface appearance of the fibres, but the repetition removes from the believability of the final product.

We were stumped for a while, but thankfully Feng Xie, our amazing TA, suggested that we use a medium to approximate the inner part of a cotton candy blob. This makes sense because scattering from infinitesimally small fibres randomly oriented should converge to uniformly random scattering, which should be well-approximated by even a homogeneous volume. Then, we would only have to model and represent the curves on the surface on the cotton candy blob. This makes the problem of rendering the cotton candy seem far more tractable.

2.2 Custom curve generation

Armed with the previous experiments, we realized that we would need a custom way to generate curves to our liking, and wrote a script to do so. The idea is to generate a "sheet" of curves between two Bézier splines of the same degree and number of segments, much like a 2D bezier surface. Each new curve will have the same number of control points, and each new control point is completely determined by the index of the corresponding pair of points from the guide curves. Figure 4 illustrates this process by displaying the curves generated between two completely straight Bézier splines.

We integrated this curve generation scheme into Blender as a plugin. Then, we sculpted a simple mesh for the inner shape of the homogeneous volume, and modelled pairs of Bézier splines to cover its surface. The curves were then manually tweaked to remove stray curves and for additional variation. Another custom script then exports them with randomized widths into PBRT as cylindrical curves. The workflow is illustrated in Figure 5.

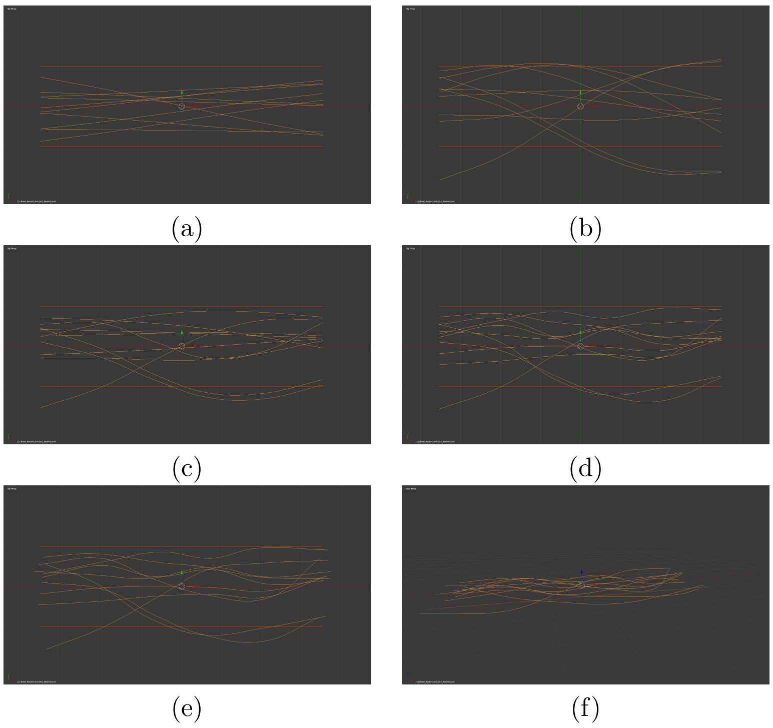


Figure 4: Several layers of randomization to generating random curves between two guide curves. (a) Pick random endpoints in $[0, 1]$, and linearly interpolate between each pair of corresponding control points. (b) Add sine waves of random amplitude and phase to make the curves curve and cross each other. (c) Randomize the frequency of the waves to control deviation. (d) Random lateral (between the two curves) displacement. (e) Random parallel (the derivative of the point in stage (a)) displacement. (f) Random normal displacement (perpendicular to the two other directions) to control the apparent volume of the resulting set of curves.

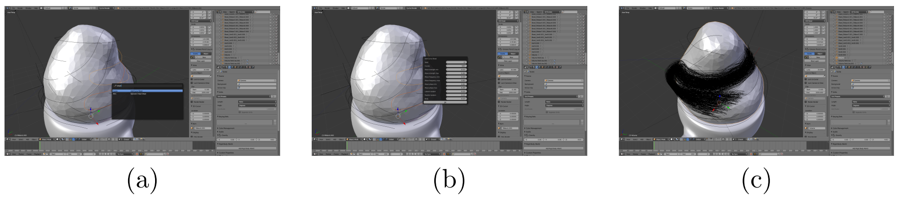


Figure 5: Modelling workflow. (a) Model pairs of guide splines on the surface of the inner volume. (b) Choose parameters for the random sheet generation process. (c) Only minimal tweaking is required to get the desired look from the resulting curves.

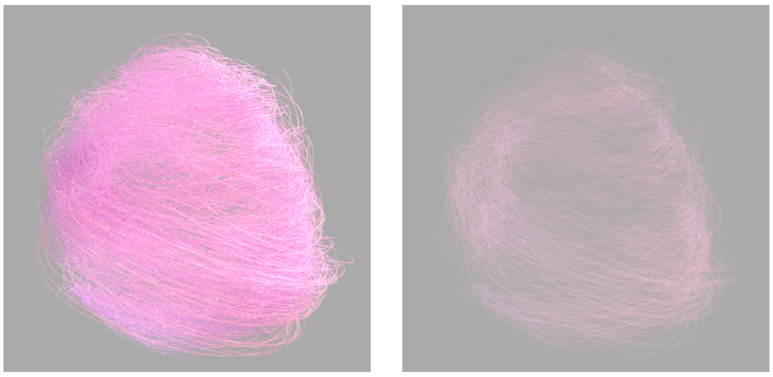

(a) (b)

Figure 6: Curves rendered with the Disney BSDF in PBRT. (a) has thicker curves than (b).

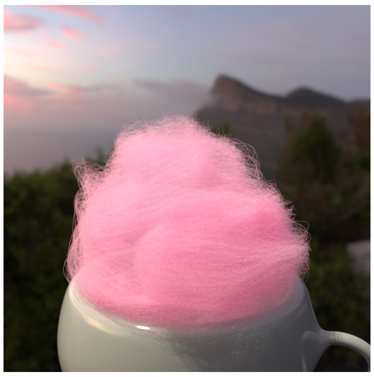


Figure 7: Candy with the Translucent material, with 4096 spp.

2.3 Cotton Candy BSDF

The next step was to approximate the appearance of the cotton candyby picking the right material for the hair and volume. We also tuned its parameters for the right visual characteristics as explained in Section 1. We found that an important part of the wispy appearance of cotton candy is picking the right hair width, as illustrated in Figure 6. Too thick and it appears like synthetic fibre, but too thin and the curves become barely visible.

Disney and Translucent We experimented with the Disney and Translucent material in PBRT, but found them to be unsuitable. The Disney material is significantly darker with high transmissiveness. This forces us to make it more reflective, so it appears too hard and makes the candy look like fabric fibres (see Figure 6). The Translucent material gives promising, saturated results because we can separately control the specular and diffuse reflection/absorption spectra. However, it is slow to evaluate and has a low sample efficiency (see Figure 7)

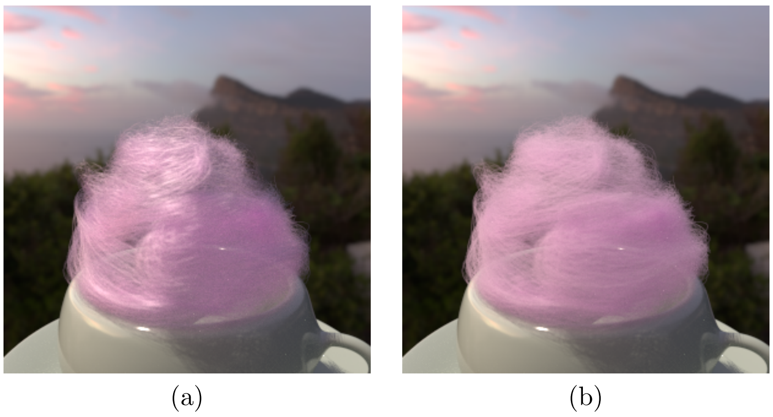


Figure 8: Two hair texture cotton candies. (a) has $\beta_m = 0.2, \beta_n = 0.8$, which makes it very shiny. (b) has $\beta_m = 0.8, \beta_n = 0.8$, which makes it appear softer.

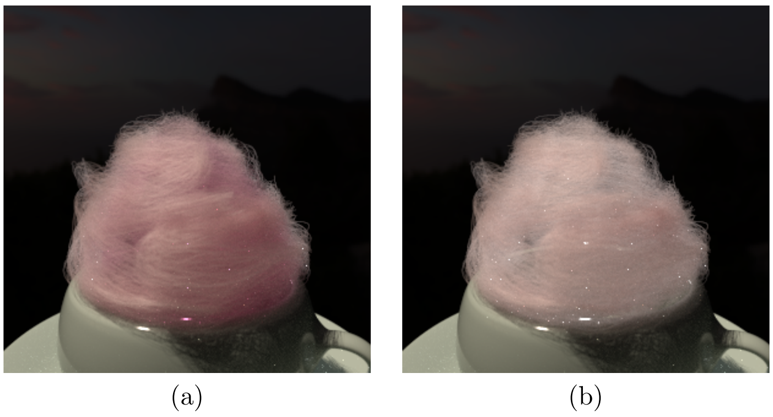


Figure 9: (a) Candy with a volume. (b) Candy without a volume. Without the volume, the amount of curves present is not enough to block eyesight. Further, the absorption of the volume colors the candy.

Hair We settled on the Hair material. The hair model in pbrt is an improved version of Marschner hair model [MJC$^+$03, dFH$^+$11, dMH13]. With high longitudal and azimuthal roughness $\beta_m = 0.6, \beta_n = 0.8$, we can soften the texture of the curves. If the roughness is too low, the highlights on the candy look too shiny and the image becomes unnatural. See Figure 8 for a comparison. Finally, we used the PBRT implementation of [CBTB16] to specify an approximate candy fibre color directly.

Volume We also use a homogeneous medium at the core of our cotton candy model to approximate multiple scattering and absorption under the surface. Since the fibres represented by curves have a small width and little chance for volumetric absorption to affect the overall color, the volume is responsible for most of the saturation in the model. Highlights on the cotton candy have less color saturation as they stem from specular reflection on the hair BSDF. Areas lit mostly by 2-3 bounces of scattering within the cotton candy gain more saturation as light rays undergo the absorption of both candy fibre curves and the volume. See Figure 9.

2.4 Layers of curves

The final cotton candy model is made of four components, each of which are critical to the final appearance of the model.

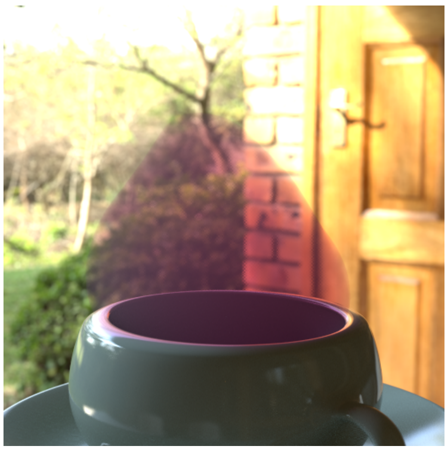
Inner volume As discussed earlier, this provides most of the color that is visible in the cotton candy, and also removes the need to fill the entire interior of the model with too many curves.

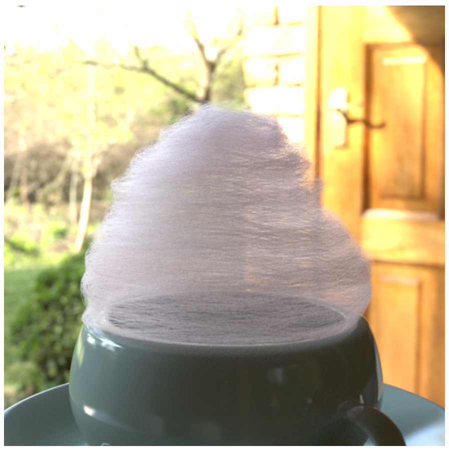
Base layer This is a layer of curves that tightly hugs the inner volume with slightly thicker fibers. The purpose of these curves is to hide the edges of the volume in the final render. However, they produce cotton candy that, while still realistic, are too uniform and are not visually interesting.

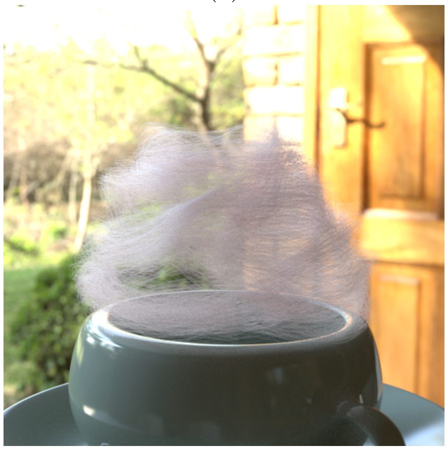
Ribbons These curves twist, and fold over themselves arbitrarily over the base layer. This creates regions of higher and lower density, as well as a few outcrops that stand out from the model. The denser regions that are closer to the sun become brighter than the others, creating structure and making the cotton candy visually interesting

Cross web One of the features of cotton candy is that imperfections in the way it is twirled control its appearance greatly. There are some fibres that run mostly perpendicular to the mostly horizontal direction and form a spiderweb-like layer over the cotton candy. This is mostly only visible on the sunlit edge, but adds a lot to the realism of the image.

These four components are independently shown in Figure 10. To show how critical each component is to the final appearance of the image, Figure 11 shows what happens when certain components are removed.


(a)


(b)


(c)

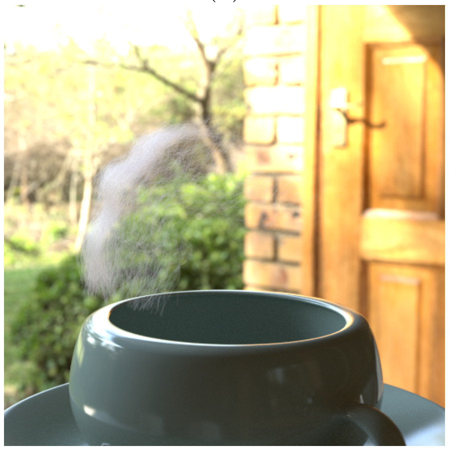
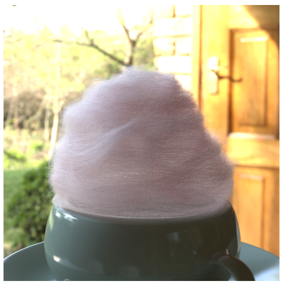
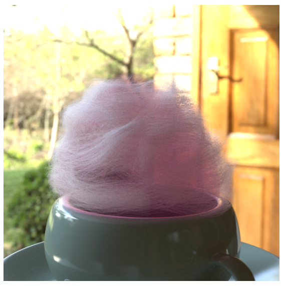

(d)

Figure 10: Four components of the cotton candy model. (a) Inner volume. (b) Base layer. (c) Ribbons. (d) Cross web.


(c)


(d)

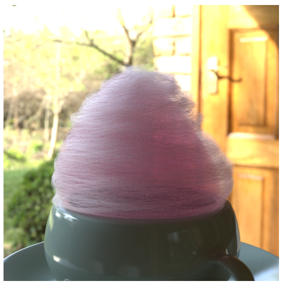


Figure 11: What happens if certain components are removed. (a) No volume. Cotton candy appears empty and desaturated. (b) No base layer. Edges of inner volume are visible. (c) No ribbons and web. Model appears too visually uniform and uninteresting.

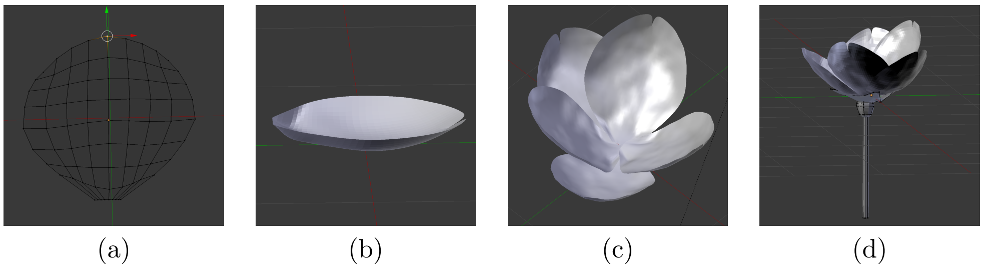


Figure 12: (a) Modeling petal from plane. (b) Petal model after tuning. (c) Putting petals together. (d) Adding stem to petals.

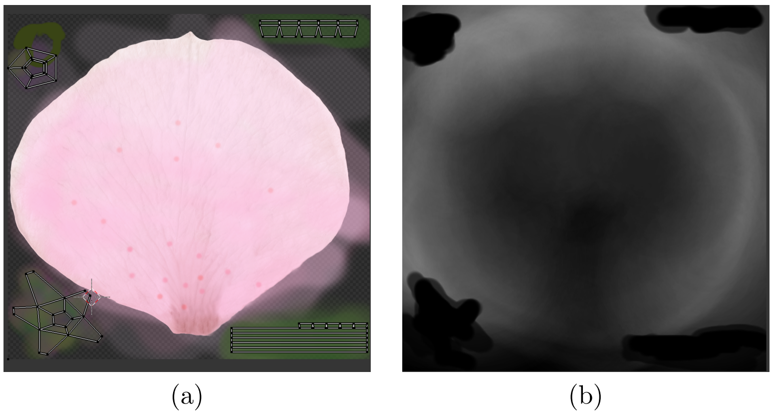


Figure 13: (a) Petal colored texture map. (b) Petal transparency map.

3 Scene Composition

3.1 Sakura Branch

We modeled the sakura branch using Blender. To make the petals, we started from a plane mesh. Next we adjusted the vertices to create the petal contour. Finally, by deforming the plane and applying solidify modifers, we achieved a realistic model of the petals. Then we made a flower stem to join the petals together, see Figure 12.

Previously, we also added details such as flower core, flower filament and flower anther to each of the flower. However, as it turned out later, this model took up took much space. For each flower, it had around 8,000 vertices, since we have lots of flowers on the branch, this is too much than we want. Thus, at the end, we sticked to the modified version we have right now, which only has less than 500 vertices.

So we rely on textures to achieve detailed look. An image of a sakura petal was projected onto the texture map and texture painting in Blender was used to add more detail, see Figure 13. We used the same process to paint the texture for flower stems. Additionally, we used a translucency map that makes the edges of the petals more translucent than the center to approximate the varying thickness of flower petals, which thicken at the base. These texture maps are applied on the Translucent material in PBRT to achieve the final look.

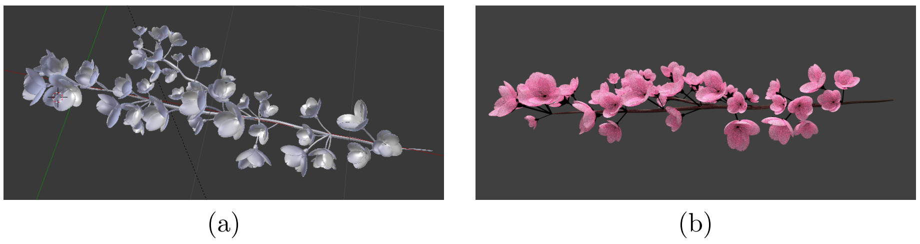


Figure 14: (a) Branch model in blender. (b) Branch model in blender with texture.

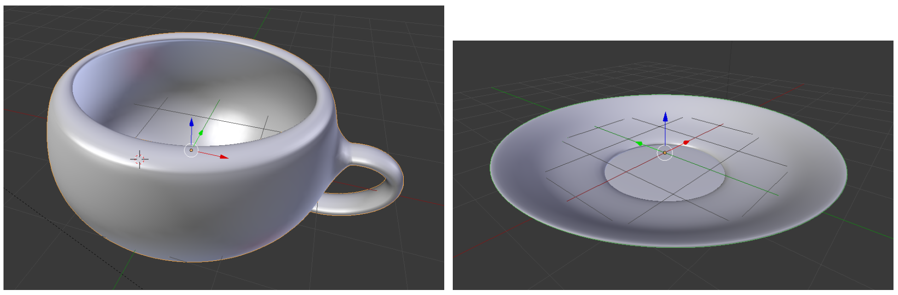


Figure 15: Cup and plate model in blender

Finally, we modeled and textured a branch. By joining flowers to the branch, we have completed modeling the entire sakura branch, see Figure 14.

3.2 Cup and plate

We built the cup and plate completely from scratch in Blender. To generate the ceramic material for pbrt, we used `layerlab` [JdJM14], which is a Python-based toolbox for computations involving layered materials. By setting the albedo to $[0.6, 0.675, 0.62]$, the index of refraction $\eta = 1.5$ and roughness $\alpha = 0.02$ we get the bsdf ceramic material we have now.

3.3 Table

The table is a simple plane with beveled edges that are ultimately not visible in the scene. We textured it by passing a wood texture through CrazyBump [Cla] (a freely available and very powerful texture processing program) to extract a specular map and unlit diffuse albedo map. The specular map modulates both the specularity and roughness (inversely) of the otherwise featureless plane. Wood grain is shinier on the surface than its grooves, and has many fine striations. A desaturated, high-contrast version of the wood texture was thus used as a bump map to simulate the fine detail of the wood grain without additional geometry.

The petals on the table were placed with a particle system and a physics simulation that allows them to fall on the table.

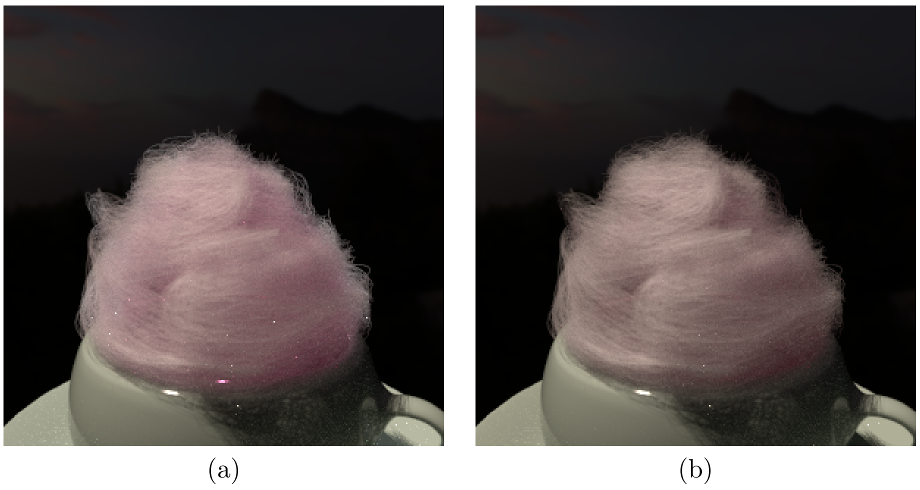


Figure 16: (a) Backlight. (b) No backlight. A backlight adds to color saturation from volumetric absorption. Moreover, it functions like a rim light and highlights the edge of the cotton candy.

3.4 Lighting

We decided to use a lovely environment map of a veranda (CC0) from HDRI Haven [Zaa]. This indoor scene has great lighting that comes mostly from one direction, which is perfect for lighting the cotton candy to show maximum detail.

An important feature of the final scene, which we also added thanks to Feng's suggestions, is the configuration which uses the environment map as a backlight. With a backlight shining through the candy, fibres on the edge appear translucent, which makes them feel softer. The additional color saturation picked up from the volume (we did not model curves on the back of the model) also adds depth to the model. See Figure 16.

Apart from the environment map, there is also a warm spotlight shining on the left side of the model and a cool one from high up on the right of the model. This subtly accentuates the contrast between the detailed left side of the cotton candy and the soft, glowing appearance of the indirectly lit side.

3.5 Miscellaneous

We used the `volpath` integrator with a `sobol` sampler. We also used a perspective camera focused on the foreground to blur out the background. Object positioning and scene composition was done mostly in Blender. A custom exporter was written to randomize curve widths, meshes were exported to the PLY format, and a simple script and rig system was written to enable easy transfer between the Blender and PBRT cameras.

4 Distributed rendering

Since we are rendering hair material with multiple bounces, it takes a lot of samples to reduce variance. Moreover, it is very difficult to debug the fine structure of a cotton candy without sufficient number of samples. Thus, we extended pbrt to support multi-node distributed rendering. To achive this with minimal changes to the system structure, we add two optional flags to the pbrt-excecutable: `--dist-master` and `--dist-slave`. The `Render` function of any `SamplerIntegrator` is extended to 1) send tasks to clients if it is a master and 2) connect to master, receive coordinate, render a tile of the image, and send data back to master if otherwise. This allows almost arbitrary horizontal scalability of computation power.

With the updated pbrt-excecutable, we further assume that all machines in the rendering cluster has access to a shared network file seytem. This is easy to achieve in every cloud provider. Then since each process is reading the exact same `.pbrt` file and folder structure, tiles rendered by them can be seamlessly composed.

However, it is infeasible to log into multiple machines and run commands separately on each machine. We additionally developed a python-based scheduler that manages what pbrt processes to run on each node with a single master server and a slave daemon for each node. The master server supports a frontend that communicates with user, who can ask the master server to render a `.pbrt` file on its filesystem. We develop a `pbrt-client` commandline tool to send instructions to the master server. Additionally, `pbrt-client` provides `rsync` based functionality to sync with remote files and retrieving rendered images. The system setup is illustrated by Figure 17.

Finally, to simplify setting up network connection and ease deployment. We use a soon to be released library `Symphony` to automate deploying a render farm onto a kubernetes cluster. We build docker images containing the pbrt excecutable and python schedulers. A `pbrt-cluster` commandline deploys, manages and tears down the render farm. On google cloud, kubernetes supports autoscaling, so machines are turned off after the render farm is turned off. In all, this means that we are only incuring costs from compute power when we use them.

In all, after setup, the usage of the distributed rendering system is very simple.

```
$> pbrt-cluster launch my-cluster 20
# Launches a cluster named my-cluster with 20 worker nodes
# The commandline waits for the cluster to setup
$> pbrt-client render scene.pbrt
# The client syncs the folder containing scene.pbrt to remote
# It creates a job that renders scene.pbrt
$> pbrt-client fetch
# The client periodically fetches render results
$> pbrt-cluster delete my-cluster
# Turns off the cluster. With autoscaler setup
# there would soon be no running instances
```

With this setup, we used 160 cores to render our final scene at 1920×1080 resolution with 4096 samples per pixel using volpath tracer with sobol sampler. It takes 4517s to render.

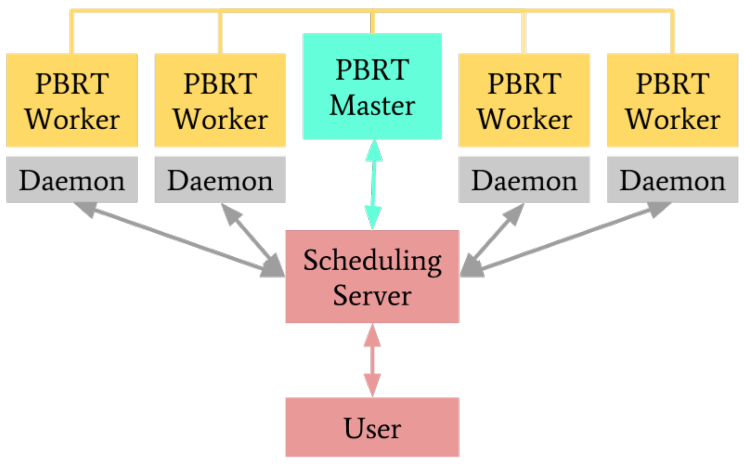


Figure 17: The system setup of distributed pbrt rendering. The user interacts with a python based scheduling server that communicates daemons on worker nodes. The scheduler server and daemons spin up pbrt processes with the proper input. These pbrt processes collectively complete the render.

5 Acknowledgements

This project would not have been possible without the tremendous support we received from Feng Xie, our TA and also from Pat Hanrahan and Matt Pharr, our course instructors.

We are also very grateful to the various free and usually open-source tools used in this project (Blender, GIMP, layerlab, CrazyBump) that made scene manipulation and texturing possible.

6 Contributions

The final product was a collective effort of all three team members. Hubert modeled and shaded the cotton candy and the table. Chenlin modeled and textured the sakura, cup and plate. Jiren developed distributed pbrt rendering. All three members collaborated closely to compose the final scene and tune the final image.

References

[CBTB16] Matt Jen-Yuan Chiang, Benedikt Bitterli, Chuck Tappan, and Brent Burley. A practical and controllable hair and fur model for production path tracing. In *Computer Graphics Forum*, volume 35, pages 275–283. Wiley Online Library, 2016.

[Cla] Ryan Clark. Crazy bump.

[dFH$^+$11] Eugene d'Eon, Guillaume Francois, Martin Hill, Joe Letteri, and Jean-Marie Aubry. An energy-conserving hair reflectance model. In *Computer Graphics Forum*, volume 30, pages 1181–1187. Wiley Online Library, 2011.

[dMH13] Eugene d'Eon, Steve Marschner, and Johannes Hanika. Importance sampling for physically-based hair fiber models. In *SIGGRAPH Asia 2013 Technical Briefs*, page 25. ACM, 2013.

[JdJM14] Wenzel Jakob, Eugene d'Eon, Otto Jakob, and Steve Marschner. A comprehensive framework for rendering layered materials. *ACM Transactions on Graphics (ToG)*, 33(4):118, 2014.

[MJC$^+$03] Stephen R Marschner, Henrik Wann Jensen, Mike Cammarano, Steve Worley, and Pat Hanrahan. Light scattering from human hair fibers. In *ACM Transactions on Graphics (TOG)*, volume 22, pages 780–791. ACM, 2003.

[Zaa] Greg Zaal. Hdri haven.