Rendering Realistic Whiskey and Vodka: Xianzhe Zhang, Wen Zhou

xianzhez, zhouwen @ stanford.edu

1. Introduction

In this project, we aim to render a scene inspired by the images below, with some glass that contains ice cubes and tempting alcohol drink. This is interesting to us because we find such images very visually appealing and may be useful in advertisement industry. Liquid, ice and light together can always produce captivating effects.

Figure 1: Inspiration images.

2. Technical details

Our code is in a github repo: https://github.com/wendyzzzw/cs348b-2019-wenzhou/tree/project. Note that this is a private repo while cs348b19TA is a collaborator and thus can see our project source code.

2.1 Volumetric photon mapping

We first implemented volumetric photon mapping to synthesize the realistic lighting effect of light scattering in the liquid, ice and glass as described in [1, 2].

Our implementation is based on the source code of the stochastic progressive photon mapping integrator for surfaces (sppm.c / sppm.h) that is already provided by PBRT. It includes codes for generating visible points using a SPPMpixel structure and storing them in a SPPMgrid structure, as well as for shooting photons and estimating surface radiance. Fig.2 (a) shows a teapot scene [3] rendered with the surface sppm integrator, in which the tea inside the teapot only has color at the outmost surface as the photons are only intersecting with the surface instead of the medium volume.

With this PBRT's implementation of sppm, the remaining task we need to do is to extend this surface photon map to the volumetric participating media. In order to fulfill this, we followed section 2 and 4 in [2] and referred to the implementation of volumetric path tracing contained in the source code of PBRT (volpath.c / volpath.h).

The first step is to handle photon-medium interaction during the phase when we are following camera ray path to find the visible points. The volpath integrator volpath.c / volpath.h) in PBRT provides codes for propagating the ray through the volume and handling scattering at point in medium. We used these codes to estimate in-scattered radiance at each point sample and save them in the SPPMpixel structure. In fig.2 (b) the image is rendered with the volpath integrator, while in fig.2(c) the image is rendered with surface sppm plus our camera ray marching implementation. We can see that the tea now has color since the camera ray can “scatter through” the volume media.

The second step is to intersecting the photon with the volume medium during the phase when we are shooting photon rays and tracing them to accumulate contributions. To do this, as SPPM, we use a halton sampler to sample the photons, then follow the photon path through the scene, and when it interact with a volume, we calculate the scattering at the intersection point and sample a new photon ray direction with this information. Russian roulette is also used to early terminate photon path. As shown in fig.2(d), the image rendered with our volumetric sppm now has the correct lighting effect in tea.

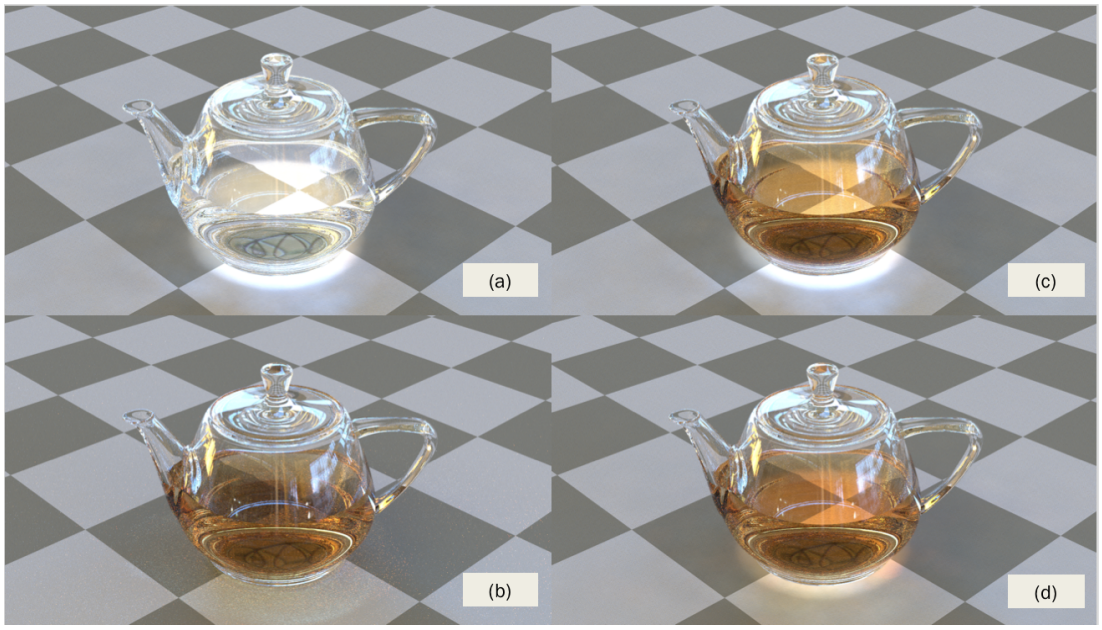

Figure 2: Teapot scene rendered with different integrator: (a) surface sppm; (b) volpath; (c) surface sppm + camera ray marching through volume; (d) our full implementation of volumetric sppm.

Then we use our volumetric sppm integrator to render one whiskey glass that contains liquid and ice cubes as shown in fig.3. We are satisfied with the caustic and color bleeding effect. The interactive surface of ice/liquid, liquid/glass looks realistic.

Figure 3: Whiskey glass rendered with our full implementation of volumetric sppm.

2.2 Layered materials

To make the scene look more realistic, we decide to apply layered materials on some of the objects in the scene. After read a wide range of related papers and investigated many projects about layered materials, we decide to use LayerLab [4] to generate BSDFs of layered materials. Even though PBRT-scenes have already prepared some pre-computed BSDFs of layered materials for us, we want to try different materials easily and change the properties of the layered materials as we want. LayerLab is an ideal choice for us.

LayerLab is a great system which supports arbitrary layer structures and remains efficient and accurate. It uses tabulating reflectance functions in a Fourier basis, but stores models in a compact form. Therefore, we could easily use the BSDFs generated by LayerLab as Fourier materials in PBRT. We followed the tutorial of LayerLab and the source code to learn how to write our code to generate new materials.

There are two main categories of materials in LayerLab: conductors and dielectrics. And we can easily adjust the isotropic and anisotropic property, roughness of the boundaries, albedo, absorption. We want to use conductor materials for the shelf and the stick in our scene and use dielectric materials for the bottles on the shelf.

For the shelf and stick in the scene, we are using coated metal as their materials. The top layer is a coating layer, the bottom layer is a metal layer. We used chrome as the materials for both shelf and stick. The roughness of both layers of shelf is 0.02, and the roughness of both layers both layers of stick is 0.1. As you can see in the following image, the roughness would affect their reflection greatly. And the layered material would also make both object more realistic.

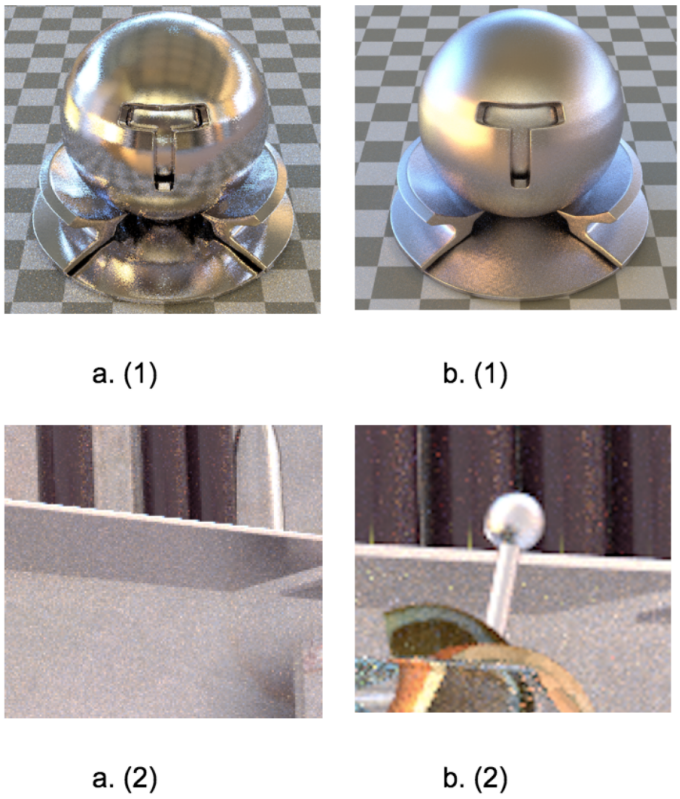

Figure 4: Metal materials with different materials.

For the bottles on the shelf, we are using layered dielectric materials to get the final ceramic material. We tried different parameters to get different colors and reflective effect. We use different albedos to get different different colors, fig. 5 shows the color and the albedos we used with eta=1.5 (i.e. absorption), alpha=0.02 (i.e. roughness).

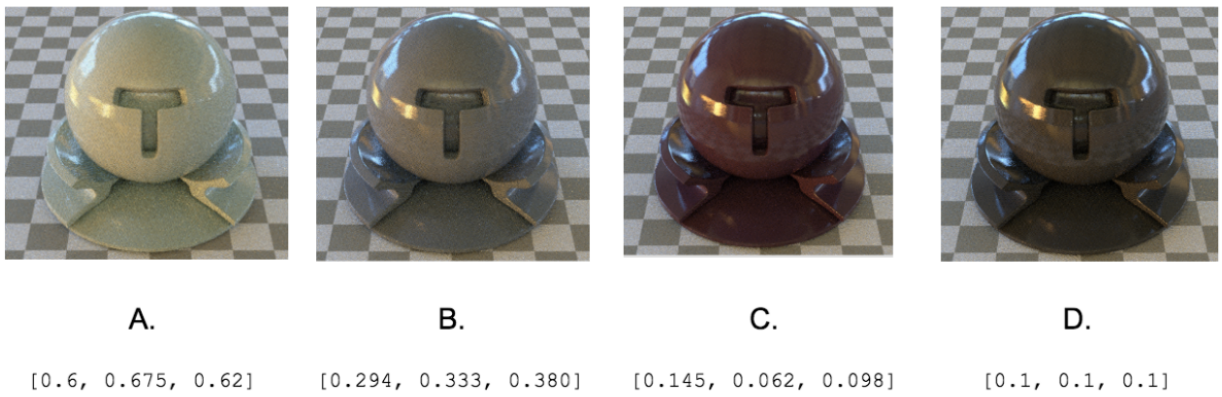

Figure 5: Dielectirc materials with different albedos.

Finally, we chose the layered material with albedo [0.145, 0.062, 0.098] as the ceramic material of the bottles. The effect is shown as below:

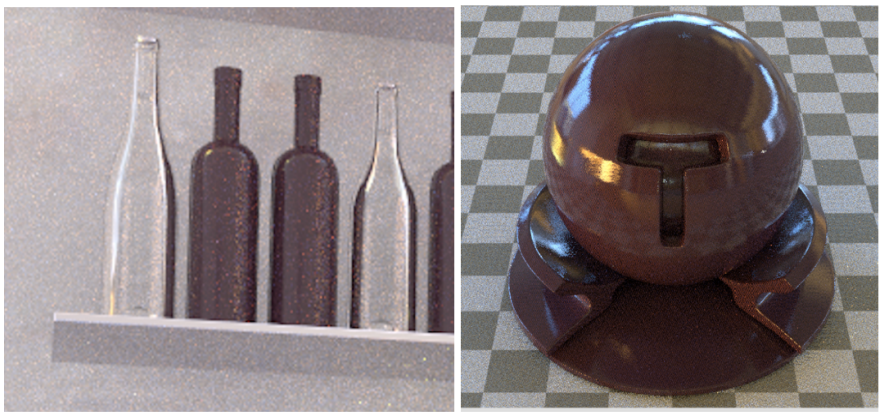

Figure 6: Final effect on bottles.

2.3 Modeling

We downloaded several 3D models (whiskey glass, vodka glass, bar table, bottles and shelf) from TurboSquid.com and aligned them together to build our own scene in Blender. The entire 3D scene is shown in fig.7.

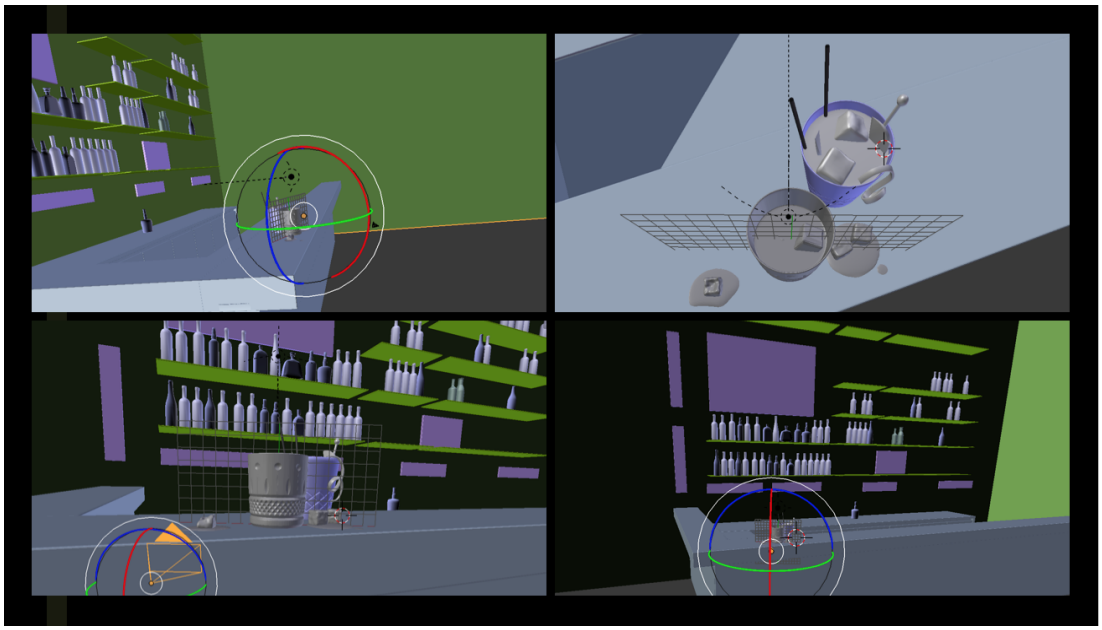

Figure 7: 3D scene in Blender from different views.

To export the mesh to be used in PBRT, we used the pbrt-v3-blender-exporter[5] to export our model as the PBRT format.

3. Problems encountered

When implementing our volumetric photon mapping integrator, we need to use a Halton sampler to sample photons. However, we found that the standard Halton sampler provided by PBRT can only has a sample dimension less than 1000. To tackle this problem, we implemented our own sampler (MySampler in volsppm.cpp), which returns rng.UniformFloat() when encountering a higher than 1000 sample dimension.

Another problem is that after we export the Blender scene to PBRT, we found that the coordinates in PBRT is slightly different from that in Blender so we need to carefully re-adjust the camera position and fov to get our desired camera view. Also, the plug-in we were using to export Blender scene to PBRT does not support texture and materials. We need to include the imagemap and bumpmap by hand, as well as pick our desired materials and adjust the corresponding parameters after carefully reading the PBRT documeng and source codes.

Besides, we found that the lighting could significantly influence the effect of rendered image. Therefore, we tried different light sources to achieve a better artistic effect. The distance between light source and the objects in the scene could also impact the rendered effects. So we made many trials on these parameters. Finally, we used 2 environmental light source (one is an environmental map, the other one is a weak uniform infinite light), 3 spot-light sources (one orange, one redish and one white).

Moreover, there is a crutial problem regarding with the rendering process. In our project, we used complicated algorithm and multiple different layered materials. It will take tons of time and be infeasible to render our final image in a high resolution and quality on our local machines. So to overcome this problem, we set up Google Cloud Platform to use multiple CPUs to render our final image. We used 24 cores with 48 hyper threads to render our final image with resolution of 2400x1350. It took around 4 hours to get our final result.

4. Results

4.1 Intermediate Results

Figure 8: Intermediate image with a simpler scene.

Figure 9: Rendered image without camera depth-of-field. The resolution is 980 x 540, with 512 samples per pixel and 512 volumetric sppm integrater numiterations.

4.2 Final Image

Figure 10: Our final image with camera depth-of-field. The full resolution is 2400 x 1350, with 2048 samples per pixel and 2048 volumetric sppm integrater numiterations. The full-resolution image can be accessed **here**.

5. Team Contributions

We work closely with each other and meet in person to discuss and work together.

- Xianzhe Zhang: Explored photon mapping integrator techniques and implemented part of volumetric photon mapping integrator in PBRT, use LayerLab to generate BSDFs of layered materials, set up GCP to render our image.
- Wen Zhou: Implement volumetric photon mapping integrator in PBRT, set up PBRT export plugins in Blender, build the scene with 3D models in Blender and adjust camera view.

Reference

- [1] Henrik Wann Jensen. 2001. Realistic Image Synthesis Using Photon Mapping. A. K. Peters, Ltd., Natick, MA, USA.
- [2] Jensen, Henrik Wann, and Per H. Christensen. "Efficient simulation of light transport in scenes with participating media using photon maps." Proceedings of the 25th annual conference on Computer graphics and interactive techniques. ACM, 1998.
- [3] Benedikt Bitterli, 2016. Rendering resources. https://benedikt-bitterli.me/resources/
- [4] Jakob, Wenzel, et al. "A comprehensive framework for rendering layered materials." ACM Transactions on Graphics (ToG) 33.4 (2014): 118.
- [5] pbrt-v3-blender-exporter. https://github.com/giuliojiang/pbrt-v3-blender-exporter