# Realistic Water Caustics in pbrt

Fall 2022 CMPT 985 Project Final Report

Team: Water Bending - Kai Chu & Hou In Tam (Ivan)

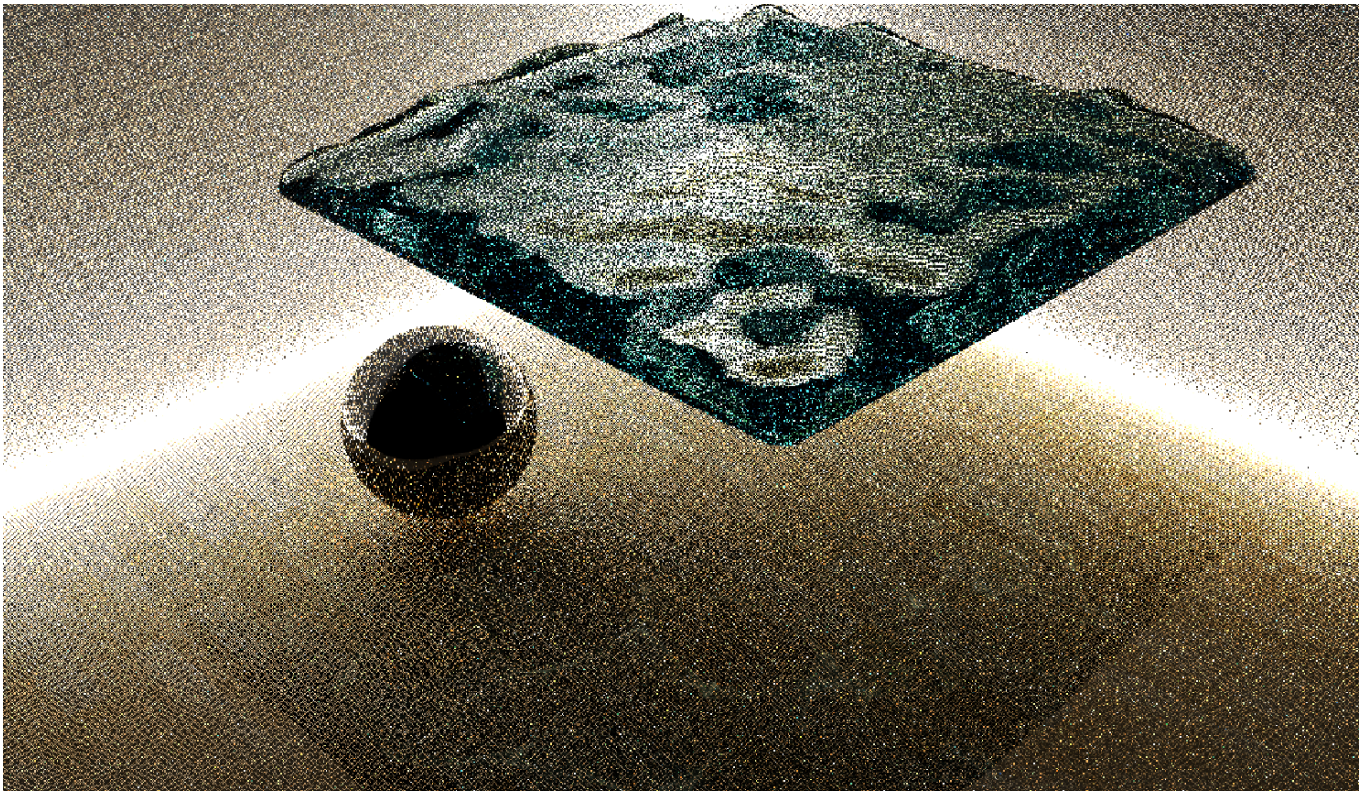

Fig. 1. This image is rendered using our VCM implementation in pbrt with 4 samples per pixel and a max depth of 20. It does not look right because the MIS weight for vertex merging is erroneous. We created this scene in Blender. The water's material has no colour. The bluish colour is created by specifying the water's absorption cross section values.

## 1 INTRODUCTION

Water caustics is a physical phenomenon caused by light interacting with water surfaces. Light energy heading toward a body of water can get reflected, refracted, scattered, and absorbed by the water, resulting in mesmerizing visual patterns on nearby surfaces (e.g., Figure 2). These visual effects add an extra layer of depth to the rendered images of 3D scenes and make them more realistic to the audience. However, since water can interact with other objects in a scene and take on any shapes, many complex calculations are needed to simulate how light energy disperses in a scene accurately. Consequently, it is difficult and time-consuming to recreate this effect in rendering pipelines.

For years, it has been common practice to utilize noise generators and texture maps to fake water caustics in scenes (2.1). However, while they can create some stunning effects, they are by no means physically accurate. The two most commonly used algorithms for physically accurate light simulations are bidirectional path tracing (2.3) and photon mapping (2.4). They can capture lighting in

Fig. 2. This image captures the visual effect we tried to recreate in this project. A pool with caustic effects at the bottom of the pool and on the nearby wall. Scene by 3Darcspace studio. Rendered in Corona Renderer.[8]

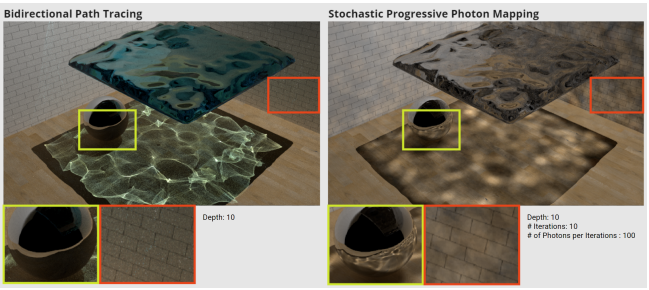


Fig. 3. We rendered a scene we built using pbrt's stock BDPT and PM. It is clear that BDPT and PM each has its own strengths and weaknesses.

scenes better than classical path tracing, but they also have their weaknesses. By combining both these methods, vertex connection and merging is an algorithm robust against highly complex light environments, making it an excellent method for capturing caustic effects (2.5).

In this project, we implemented the vertex connection and merging algorithm in the pbrt renderer with the goal of generating images with physically accurate water caustics. In particular, we aimed to take scenes with a body of water (with real surface geometry) as the input to pbrt and render water caustics at two locations in the scenes: at the bottom of pools and on nearby walls. Along with that, we also sought to recreate water's turquoise blue colour by simulating how water absorbs light in the visible spectrum.

## 2 RELATED WORK

### 2.1 Texture Maps and Noise Generators

Rendering realistic water caustics is not a new problem. However, many early methods are resource intensive, making them impractical to use in real settings. Besides, they often cannot fully capture all light interactions in a scene. As a result, renderers have been faking water caustics using texture maps, noise generators, or even outright ignoring them during image generation. These approaches cause the rendered images to look unrealistic.

### 2.2 Unidirectional Path Tracing

Unidirectional path tracing (or simply path tracing) is a rendering algorithm that excels at capturing light interactions between objects in a scene. It works by shooting rays from the camera, intersecting and bouncing off surfaces in the scene, and keeping track of how much light energy arrives at the camera along the rays. While expensive to compute, this method can simulate complex inter-object light interactions in scenes much better than rasterization. Hence, it is the superior approach for generating physically accurate images.

However, since path tracing only shoots rays from the camera, it has difficulties handling hard-to-reach light sources that are obscured by other objects. For a camera ray to reach these light sources, it has to go through multiple bounces throughout the scene, but such a path has a very low probability of being sampled randomly. For this reason, unidirectional path tracing is inefficient at handling complex scenes.

### 2.3 Bidirectional Path Tracing (BDPT)

Bidirectional path tracing (BDPT) [11][6] extends from the original path tracing algorithm's idea and introduces a new approach to handle more complex lighting environments. Instead of only generating rays from the camera, it also shoots rays from every light source in the scene. By connecting such camera subpaths and light subpaths at some intermediate vertices, the resulting full path has a much higher chance of getting sampled than only shooting rays in any single direction. This improvement makes BDPT much better at handling obscured light sources than classical path tracing.

Although BDPT can handle hard-to-reach lights, it struggles with a specific family of paths: specular–diffuse–specular (SDS) paths. Because of the two specular surfaces, even if BDPT generates subpaths from two directions, the probability of sampling these paths is still extremely low. An example of this would be looking at caustics deposited on a diffuse surface through a mirror. Since it is unlikely to sample such paths, caustics are often missing in mirrors from images rendered using BDPT.

### 2.4 Photon Mapping (PM)

Similar to BDPT, photon mapping (PM) [5] also captures light energy in scenes from both the camera and light sources. However, instead of connecting rays from both ends, PM works by performing density estimation at each vertex on the camera subpaths. The whole algorithm works in two stages. First, photons are shot from each of the light sources. Their energy is stored in a range search structure (e.g., a k-d tree) whenever they hit a surface of an object. Once this stage finishes, a new set of rays are generated from the camera and shot into the scene. For each vertex in the camera ray, the algorithm checks for nearby photons and accumulates their contributions to the camera ray.

This approach is excellent at handling SDS setups as it does not require an explicit connection between the two subpaths. The photons do not need to know whether they can be connected to a camera ray, and camera rays do not care about where the light sources are. Through density estimation, caustic effects can easily be captured as photons naturally concentrate at a specific area after passing through a refractive medium like water.

Still, PM has its weaknesses as well. Due to its use of density estimation, it requires a large number of photons to render an image with a high convergence. This limitation makes it prohibitive to use PM for very large scenes or scenes with many diffuse surfaces.

### 2.5 Vertex Connection and Merging (VCM)

Vertex connection and merging (VCM) [3] is an algorithm that combines the approaches used by BDPT and PM, as it is clear that these two methods complement each other (See Figure 3). VCM works by first shooting light subpaths from light sources in a scene. It then keeps the light subpaths' structures with their individual vertices stored in a range search structure, just like in PM. The algorithm then repeatedly performs what BDPT and PM do: connecting camera subpaths to light subpaths and accumulating light energy to camera vertices from nearby light vertices. With the use of multiple importance sampling (MIS) to scale the contributions from each method, VCM's output images can benefit from the strength of both

BDPT's and PM's approaches. For this reason, we chose to implement VCM in this project, as caustic effects require a well-rounded algorithm to capture all possible light paths.

## 3 APPROACH

We will now discuss how we implemented VCM into pbrt in our project. Instead of going through the formal derivation of VCM here, we will focus on explaining the design choices we made in our implementation. We encourage people who are interested in VCM's formal derivation to read the original paper.

As shown in Figure 4, VCM's approach can be divided into two stages, with three steps in each stage. In the following sections, we will discuss the general goal of each step and explain how we implemented it.

### 3.1 Stage 1a - Trace Light Subpaths

The goal of this stage is to generate all light subpaths to prepare for doing vertex connection and vertex merging in stage 2. While subpath generation is basically the same for all path tracing algorithms, there are two constraints specific to VCM. First, the generated subpaths' vertices must be easily accessible such that they can be put into a range search structure for performing vertex merging. On the other hand, we could not simply extract each of the generated vertices from their paths, as their path relationship is crucial for calculating the MIS for vertex connection.

To this end, we started implementing this stage from pbrt's existing PM implementation. We thought that since pbrt's PM already has a usable range search structure, we could reuse it directly to reduce the work we need to do. However, this approach turned out to be infeasible. There are two reasons for this. Firstly, we found that to enable MIS calculation for vertex connection, each vertex has to maintain two pdf values that are calculated at the time of tracing the path. However, the existing PM implementation does not need MIS, so it lacks the features needed to do the calculations. Secondly, we found that pbrt's PM implementation works oppositely to what we need. Instead of light vertices, it stores camera vertices in the first pass and adds light energy to them later. This means that to reuse the existing code from pbrt's PM, we would have to swap out a large portion of the code. Together, these two issues made us give up on continuing this approach and switch to starting from pbrt's BDPT implementation.

As it turned out, the existing BDPT implementation is more suited for our purpose. Its path-generating mechanism already returns the full path with all pdf values pre-computed. So, it is a low-hanging fruit to reuse its implementation. However, the existing code generates one path at a time and discards it after an iteration, which is not what we need. Therefore, we modified the code to generate one path per pixel in each iteration and store all of them in an array of vectors for later use. Note that we also parallelized this process to reduce the time needed to generate such a large number of paths simultaneously.

### 3.2 Stage 1b - Connect Light Vertices to Eye

The goal of this stage is to account for direct light energy accumulations coming from light vertices that are visible to the camera. This step is needed because, in most implementations, a camera's lens size is tiny (or even zero for a pinhole camera). So, the probability of tracing a light path that hits the camera lens at its end is close to zero. Consequently, we have to explicitly connect such paths to account for possible illuminations in them.

Despite its importance, we did not handle this step explicitly in our implementation. The reasoning is that this manual connection step is, in a sense, already included in the generalized vertex connection process. When we reviewed the existing BDPT's vertex connection step, this special case was already accounted for over there. Hence, to maximize code reuse, we deviated from VCM's approach and deferred this step until Stage 2a, where we handle all cases of vertex connection.

### 3.3 Stage 1c - Build Range Search Structure

This stage aims to set up a range search structure for all light vertices generated in stage 1a to prepare for vertex merging. It is the last step in the preparation phase before we can start doing vertex connection and vertex merging to accumulate light energy. There are two choices for a range search structure: a k-d tree or a voxel grid. We chose to implement the voxel grid as it is the easier one, and parts of its logic already exist in pbrt's PM implementation.

Following pbrt's PM implementation, we decided to implement the voxel grid as a hash grid using a vector of linked lists. Each linked list acts as a hash bucket that stores every vertex that falls into the same grid cell. To initialize the grid, we first iterate through all vertices we stored in stage 1a to get a bound for the grid (with consideration for the user-defined search radius). This step guarantees that the grid is large enough to contain all vertices but not oversized to have too much empty space. Once the grid is ready, putting the light vertices into the grid is just a matter of choosing the hash function. Again, we followed pbrt's PM implementation and used Teschner et al.'s Optimized Spatial Hashing function [10], as it distributes the vertices relatively well. Note that instead of storing the whole vertex objects again in the grid, we opted to store a pointer to the vertices we generated in stage 1a to reduce the memory requirement.

This concludes the preparation phase of VCM, and we are now ready to start tracing camera rays and accumulating light energy in them.

### 3.4 Stage 2a - Vertex Connection

In this stage, the goal is to trace camera rays into the scene and connect with light subpaths to gather light energy for each pixel. We followed the original VCM's approach and only connected each camera subpath to exactly one light subpath such that we could directly use their MIS formulas. This approach is also similar to how pbrt's BDPT implementation works, so it is straightforward to reuse some of its code.

For each pixel, we first use a sampler to get a 2D position on the camera's film. From there, we trace a ray into the scene using typical path tracing techniques until it reaches the max depth or cannot generate a new ray. Once we have the whole camera path, we pick a light subpath specific to this camera subpath and perform vertex connection with them. Here, we reused the existing BDPT

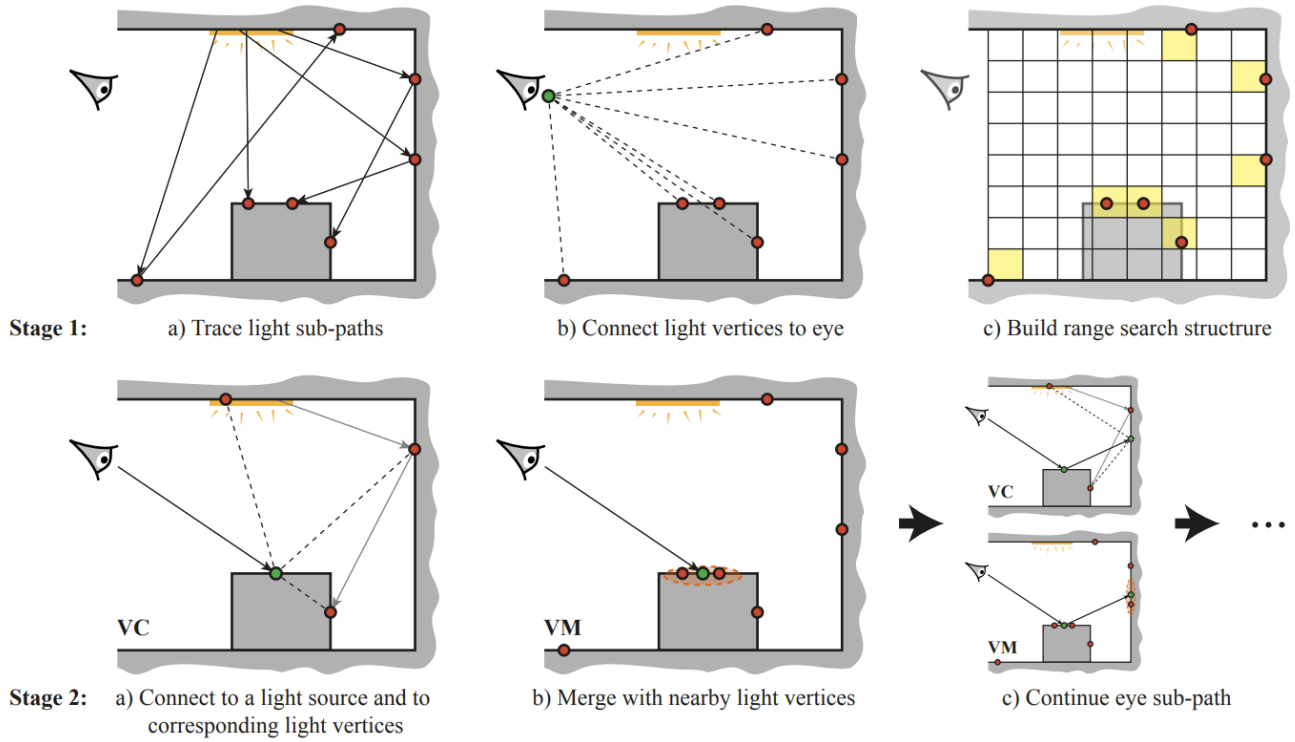


Fig. 4. The six steps approach of the vertex connection and merging algorithm.[2]

implementation as it already considers special cases like a direct connection to a light source or to the camera (the special case in stage 1b). We modified the MIS weight computation to account for possible weights from vertex merging properly.

### 3.5 Stage 2b - Vertex Merging

After finishing the vertex connection step for a camera ray, the next step is accumulating the contribution of nearby light vertices for each camera vertices in the subpath. Since we created a range search structure in stage 1c, this step is straightforward. We take the world coordinates for each camera vertices and retrieve the corresponding hash bucket for the grid. We then compare the distance between the camera vertex and each light vertices in the bucket (to account for hash collision). If they are within the search radius, we then accumulate the light vertex's contribution to the camera subpath with a corresponding MIS weight.

The vertex connection step in stage 2a and the vertex merging step here are repeated for every camera vertices in every camera subpath generated. This whole process is repeated for a user-specified number of iterations. Note that both stage 2a and stage 2b are parallelized to increase the algorithm's efficiency.

### 3.6 Multiple Importance Sampling (MIS)

The most difficult challenge we faced when implementing VCM was its multiple importance sampling (MIS). Since every complete path needs to be weighted according to all other alternate strategies that can generate the same path, there is a lot to consider.

$$w_{v,s,t} = \frac{1}{\frac{n_{\text{VC}}}{n_v}\sum_{j=0}^{k+1}\frac{p_{\text{VC},j}}{p_{v,s}} + \frac{n_{\text{VM}}}{n_v}\sum_{j=2}^{k}\frac{p_{\text{VM},j}}{p_{v,s}}}$$

Fig. 5. The formula for a complete path's MIS weight. [2]

Using the VCM paper as a reference, the MIS weight for a complete path generated by either vertex connection or vertex merging has the form of equation (Figure 5). Depending on the strategy used to sample the path, this equation can be separated into a number of equations (Figure 6), and further into equation like Figure 7. Such summation equations can then be implemented using a series of for-loops.

Apart from directly implementing these summation formulas, VCM's authors also introduced a more efficient approach to implementing MIS. By storing and updating partial values of the weight at each vertex at path tracing time, no summation is needed at the evaluation time. Although it is more efficient than using the summation formulas directly, we did not follow this approach. We tried to implement it but soon found out it was not a good fit with how

**Vertex connection.** For paths sampled with $v = \text{VC}$ we have:

$$w^{\text{light}}_{\text{VC},s} = \sum_{j=0}^{s-1} \frac{p_{\text{VC},j}}{p_{\text{VC},s}} + \frac{n_{\text{VM}}}{n_{\text{VC}}} \sum_{j=2}^{s} \frac{p_{\text{VM},j}}{p_{\text{VC},s}}$$

$$w^{\text{eye}}_{\text{VC},s} = \sum_{j=s+1}^{k+1} \frac{p_{\text{VC},j}}{p_{\text{VC},s}} + \frac{n_{\text{VM}}}{n_{\text{VC}}} \sum_{j=s+1}^{k} \frac{p_{\text{VM},j}}{p_{\text{VC},s}}.$$

**Vertex merging.** For paths sampled with $v = \text{VM}$ we have:

$$w^{\text{light}}_{\text{VM},s} = \frac{n_{\text{VC}}}{n_{\text{VM}}} \sum_{j=0}^{s-1} \frac{p_{\text{VC},j}}{p_{\text{VM},s}} + \sum_{j=2}^{s-1} \frac{p_{\text{VM},j}}{p_{\text{VM},s}}$$

$$w^{\text{eye}}_{\text{VM},s} = \frac{n_{\text{VC}}}{n_{\text{VM}}} \sum_{j=s}^{k+1} \frac{p_{\text{VC},j}}{p_{\text{VM},s}} + \sum_{j=s+1}^{k} \frac{p_{\text{VM},j}}{p_{\text{VM},s}}.$$

Fig. 6. Separated formulas for specific path sampling strategies. [2]

$$w^{\text{light}}_{\text{VC},s} = \sum_{j=0}^{s-1} \prod_{i=j}^{s-1} \frac{\overleftarrow{p}_i(\overline{\mathbf{y}})}{\overrightarrow{p}_i(\overline{\mathbf{y}})} + \eta_{\text{VCM}} \sum_{j=2}^{s} \overleftarrow{p}_{j-1}(\overline{\mathbf{y}}) \prod_{i=j}^{s-1} \frac{\overleftarrow{p}_i(\overline{\mathbf{y}})}{\overrightarrow{p}_i(\overline{\mathbf{y}})}$$

Fig. 7. The summation formula for vertex connection's MIS weight that can be implemented using for-loops. See the original paper for the equation for vertex merging. [2]

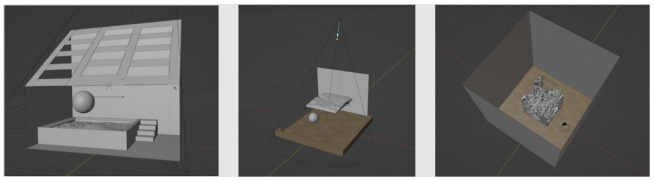

Fig. 8. We created three scenes to test our implementation.

pbrt's existing systems work. So, we implemented the summation formulas instead.

Note that we did not go into the full derivation of the formulas here, as we want to focus on explaining our design choices. Please refer to the original VCM paper for the formal derivations.

### 3.7 Scene Creation

Apart from the VCM implementation, we also worked on creating some scenes in Blender to serve as inputs into pbrt. During our project's planning phase, we thought we could simply reuse some of the scenes that came with pbrt or download pre-created scenes online to test our code. However, we soon found out that this is not feasible, as scenes online either take way too long to render or do not have geometry for their water surfaces (e.g., faked using bump maps). As a result, we spent time learning to work with Blender to create scenes with textures and materials that can be used with pbrt. See Figure 8 for the scenes we created.

### 3.8 Recreate Water's Bluish Colour

The remaining task was to find formulas to model water's physical properties. In particular, we want to specify water's index of refraction, absorption cross section, and scattering cross section values to recreate realistic light-water interaction. We discovered that pbrt already has built-in ways to model those properties. They can be modelled using material parameters when we define the water material in a scene file. So, instead of finding the formulas, our task shifted to finding the correct parameters to feed into pbrt.

We found that water's index of refraction is 1.33 [4]. We also found in the literature that pure water has absorption cross section values of $[0.624, 0.0511, 0.00530]m^{-1}$[9] and scattering cross section values of $[0.0005, 0.0015, 0.00375]m^{-1}$[1] (each corresponds to wavelengths of 700 nm (red), 546 nm (green), and 435 nm (blue) [7] respectively).

## 4 RESULTS & ANALYSIS

First of all, We were able to recreate water's bluish colour using the parameters we found in 3.8. See Figure 1.

However, unfortunately, we were not able to fully implement VCM. In particular, the multiple importance sampling code in our implementation has errors and does not weigh the light contribution coming from vertex merging correctly. This results in bright and dark spots showing up incorrectly in our rendered images.

In the following subsections, we will demonstrate why the MIS component for vertex merging is faulty by showing that all other crucial components in our implementation are working properly. We will do this by comparing our rendered images with reference images rendered with pbrt's stock algorithms, using both a scene that came with pbrt and the scenes we created using Blender.

### 4.1 Error Analysis 1 - Voxel Grid and Radius Measurement

Here, we show that our voxel grid, hash function, and radius measurement features are working as expected. Figure 9 shows a pbrt scene rendered multiple times with reducing search radius r. The rendered images show that as the search radius for vertex merging decreases, there are fewer bright spots in the images. This result suffices to prove that our voxel grid, hash functions, and radius measurement implementation are not faulty.

First, if either our voxel grid or hash function is malfunctioning, we should expect the bright spots to be missing or overly concentrated in certain regions in the images. However, it is clear that the bright spots are distributed throughout the scene, with more in regions close to the spotlight. This indicates that both our voxel grid and hash function work properly to separate the light vertices into their corresponding hash buckets.

Second, if our radius measurement function is broken, we should not expect to see a steady trend in the number of bright spots in the rendered images as we reduce the search radius. However, there is clearly a trend in the images. As we lower the radius, there are fewer bright spots in them. This is as expected as each camera vertex will accumulate fewer light vertices with a smaller radius, leading to a less bright pixel. This behaviour indicates that our radius measurement is functioning as expected.


Fig. 9. We rendered the same scene with reducing search radius in our VCM implementation. As the radius reduces, there are fewer bright spots in the image.

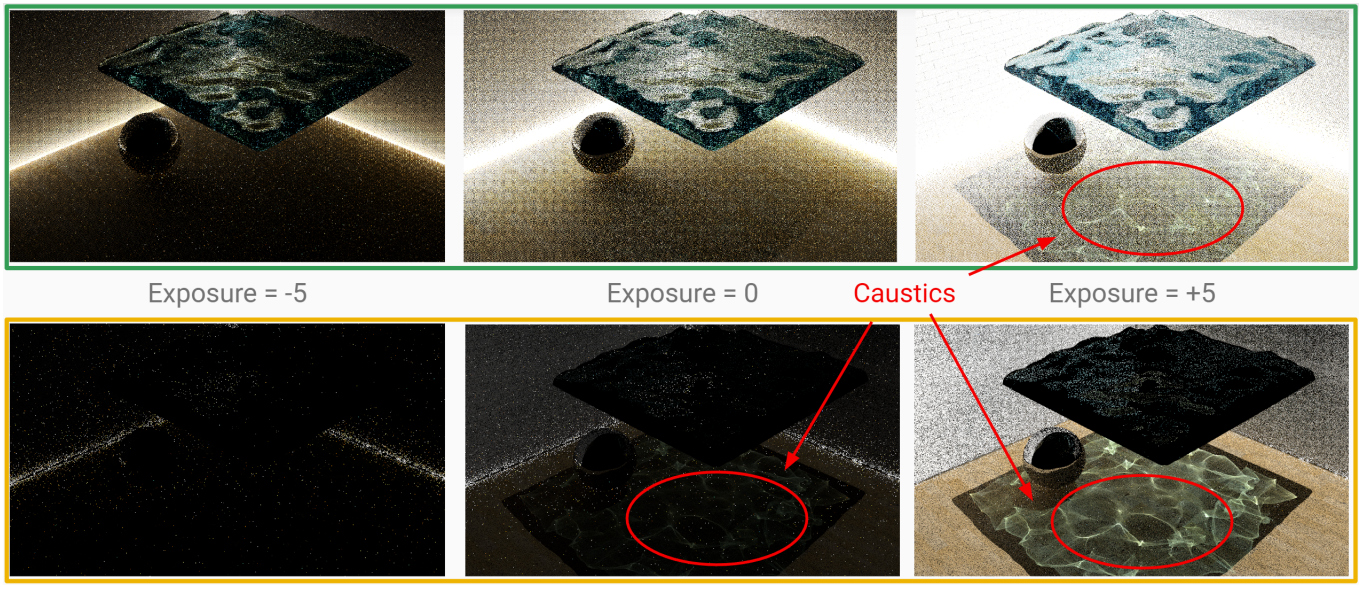


Fig. 10. Images rendered with our VCM implementation shows clear caustic effects under exposure adjustment.

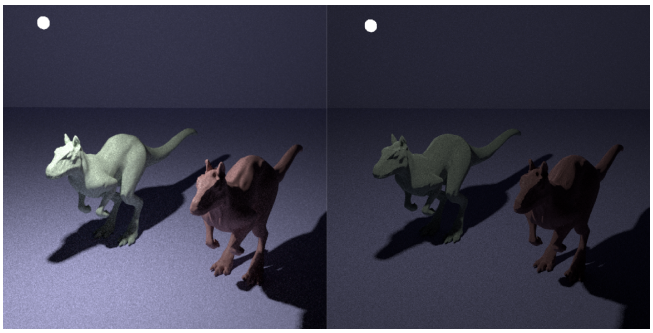

Fig. 11. We rendered the same scene pbrt's stock BDPT and our VCM with vertex merging's contribution set to zero. As expected, the image rendered using VCM is darker.

### 4.2 Error Analysis 2 - Camera and Eye Subpath Tracing Through Scattering Medium

Next, we show that the path tracing component for both camera and light subpaths are working correctly. In Figure 10, the two images in the middle are the unedited versions of images rendered with our VCM implementation. They look weird because the MIS weights are scaling the light contributions incorrectly. However, by adjusting the images' exposure manually, we can see that there are caustic effects on the floor. This indicates that the path tracing components in our implementation are handling ray bounces and medium interactions correctly. If they are not functioning properly, we should not be able to see such clear caustic patterns. In fact, we may not be able to see any interpretable image at all.

### 4.3 Error Analysis 3 - MIS weight for Vertex Connection

Lastly, we show that the MIS weight for the vertex connection component is working properly. Figure 11 a shows the default pbrt scene rendered using pbrt's BDPT (left) and our VCM (right) with vertex merging's contribution explicitly set to zero. This means that the bright spots coming from vertex merging would disappear in the image. As expected, the image rendered using VCM is darker than the reference. This is because the MIS weight for paths generated using vertex connection is working properly to scale down their contributions.

The results from all analyses indicate that all other components in our VCM implementation are working as expected. So, we believe that the MIS weight for vertex merging is the culprit in making our rendered images look weird.

## 5 LIMITATIONS

Our VCM implementation has several limitations. Of course, it is clear that our MIS weight for vertex merging is not working correctly. To produce images that look normal, we have to go back and investigate where things went wrong in our code and fix it.

Apart from the incorrect MIS implementation, we also decided not to follow the more efficient way to implement the MIS computation function. While this design choice made implementing MIS into pbrt simpler, it considerably lowered the algorithm's efficiency. We believe this is why our implementation took up to four hours to render a relatively simple scene during testing. This extended rendering time made debugging much harder than it should be and made our implementation less practical to use. Therefore, future revisits to this project should reimplement the MIS using the better formulation proposed by VCM's authors in their paper.

Finally, we should also add an option to our implementation to output unfinished images during rendering. For example, we should be able to set up the rendering loop such that it outputs an image when it finishes 50% of all iterations. This feature would make debugging much easier in the long run.

## 6 CONCLUSION

In this project, we tried to implement the vertex connection and merging algorithm in pbrt to capture water caustics in 3D scenes. While our final implementation is not 100% correct, we believe we are not far away from fully implementing it. Although we failed to achieve the goal we set up for ourselves at the beginning, we nonetheless learned a lot about different rendering techniques while working on this project. One key insight we got from this project is that there are many approaches to the same problem, and there is not necessarily a single best solution. Different approaches may have different pros and cons, and it is up to us to decide what is the best solution for our specific situation. So, when choosing a solution for a problem, it is important not to make a decision too early without exploring all possible options first.

## 7 TEAM CONTRIBUTIONS

Our contribution in each of the task and the timeline are as follow:

| Task | Kai | Ivan |
|---|---|---|
| Scene Creation | 95% | 5% |
| Implement VCM | 20% | 80% |
| Recreate Water's Colour | 5% | 95% |
| Render Scenes for Testing | 70% | 30% |
| Report and Presentation | 40% | 60% |

October 2, 2022 [Project Proposal Submission] to November 1, 2022 [Project Milestones]

(1) Get familiar with pbrt via reading the book [~4 weeks]
(2) Find and set up parameters for water's electromagnetic absorption [~2 days]
(3) Create a simple custom scene with Blender [~1 weeks]

November 1, 2022 [Project Milestones] to December 6, 2022 [Project Presentation & Demo]

(1) Implement VCM into pbrt (does not include debug time) [~3 weeks]
(2) Create a more complex custom scene with Blender [~1 weeks]
(3) Rendering scenes and Debugging [~2 weeks]

## 8 CODE

Please follow this link to download our code: https://drive.google.com/file/d/1ALlyMTIFjzMcsQiVB6VmKBQHXnJNCtKM/view?usp=share_link. This folder contains the images we generated during testing and debugging (in .exr): https://drive.google.com/drive/folders/1IqqFgUnv8xMwXnBLzOytiJwQ7CHoiD06?usp=share_link.

## REFERENCES


[1] Hendrik Buiteveld, J. H. M. Hakvoort, and M. Donze. 1994. Optical properties of pure water. In *Ocean Optics XII*, Jules S. Jaffe (Ed.), Vol. 2258. International Society for Optics and Photonics, SPIE, 174 – 183. https://doi.org/10.1117/12.190060

[2] Iliyan Georgiev. 2012. *Implementing Vertex Connection and Merging*. Technical Report. Saarland University. http://www.iliyan.com/publications/ImplementingVCM

[3] Iliyan Georgiev, Jaroslav Křivánek, Tomáš Davidovič, and Philipp Slusallek. 2012. Light transport simulation with vertex connection and merging. *ACM Trans. Graph.* 31, 6, Article 192 (Nov. 2012), 10 pages. https://doi.org/10.1145/2366145.2366211

[4] Amy Ho. 2005. Index of refraction of water. https://hypertextbook.com/facts/2005/AmyHo.shtml

[5] Henrik Wann Jensen. 2001. *Realistic image synthesis using photon mapping*. Vol. 364. Ak Peters Natick.

[6] Eric P Lafortune and Yves D Willems. 1993. Bi-directional path tracing. (1993).

[7] D. Mihai and E. Străjescu. 2007. From wavelength to R G B filter. 69 (01 2007), 77–84.

[8] Christopher Nichols. 2019. *What are caustics and how to render them the right way.* Retrieved October 3, 2022 from https://www.chaos.com/blog/what-are-caustics-and-how-to-render-them-the-right-way

[9] Robin M. Pope and Edward S. Fry. 1997. Absorption spectrum (380–700 nm) of pure water. II. Integrating cavity measurements. *Appl. Opt.* 36, 33 (Nov 1997), 8710–8723. https://doi.org/10.1364/AO.36.008710

[10] Matthias Teschner, Bruno Heidelberger, Matthias Müller, Danat Pomerantes, and Markus H Gross. 2003. Optimized spatial hashing for collision detection of deformable objects.. In *Vmv*, Vol. 3. 47–54.

[11] Eric Veach and Leonidas Guibas. 1995. Bidirectional estimators for light transport. In *Photorealistic Rendering Techniques*. Springer, 145–167.