# Realistic Water Caustics in pbrt
(Through implementing VCM)

Team - Water Bending

**Hou In Tam (Ivan) & Kai Chu**

## Project Goals

## Project Goal

Our goal was to render physically accurate caustic effects caused by light passing through and bouncing off bodies of water in pbrt

We aimed to take in scenes with a body of water and recreate two phenomena:

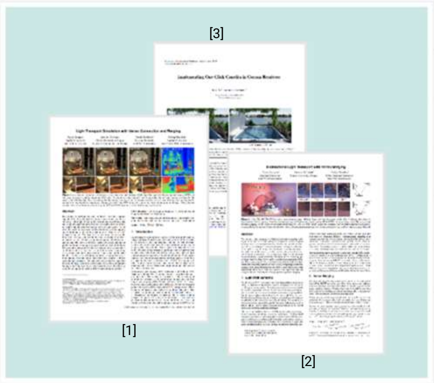
1. Caustic effects
    - At the bottom of the water
    - On nearby walls
2. Water's bluish colour caused by electromagnetic absorption

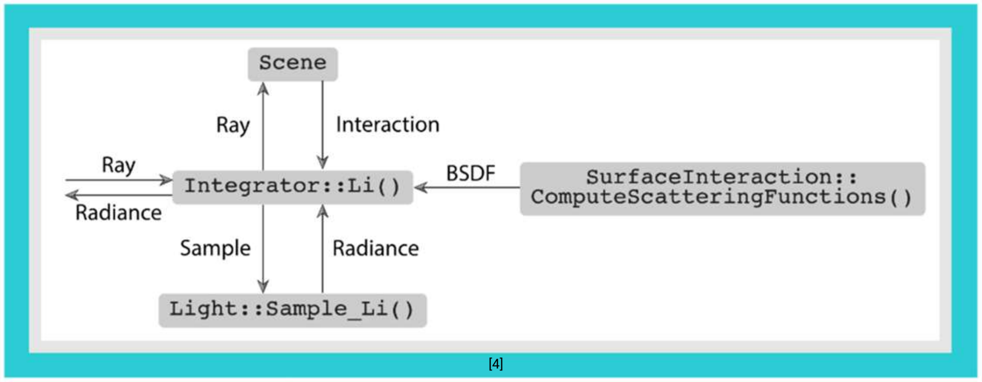
To achieve this goal, we decided to implement the Vertex Connection and Merging algorithm proposed by Georgiev et al. in 2012

# Why work on this?

- Unique real world phenomenon to capture and replicate in a rendering pipeline
- Excellent opportunity for us to learn more about rendering pipelines and how the pbrt renderer works
- Compare different rendering algorithms and learn how different approaches affect the rendering outcome

[5]


[3] [1] [2]



[4]

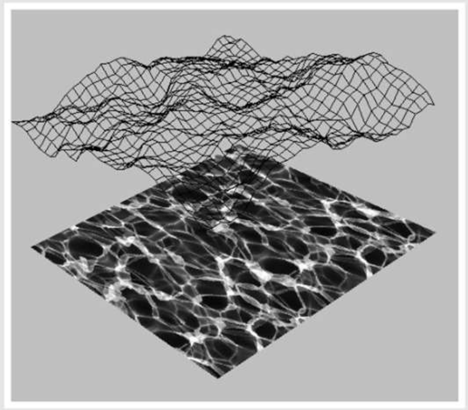
[1] Light Transport Simulation with Vertex Connection and Merging, [2] Bidirectional Light Transport with Vertex Merging, [3] Implementing One-Click Caustics in Corona Renderer, [4] pbrt 3rd edition 2018, 1.3 pbrt: System Overview Figure 1.19, [5] https://www.chaos.com/blog/what-are-caustics-and-how-to-render-them-the-right-way

## Related Work

## Related Work

**Textures**

For many years, caustics were faked using textures and directly added to scene objects


https://www.dgp.toronto.edu/~stam/reality/Research/PeriodicCaustics/index.html

https://alexanderameye.github.io/notes/realtime-caustics/

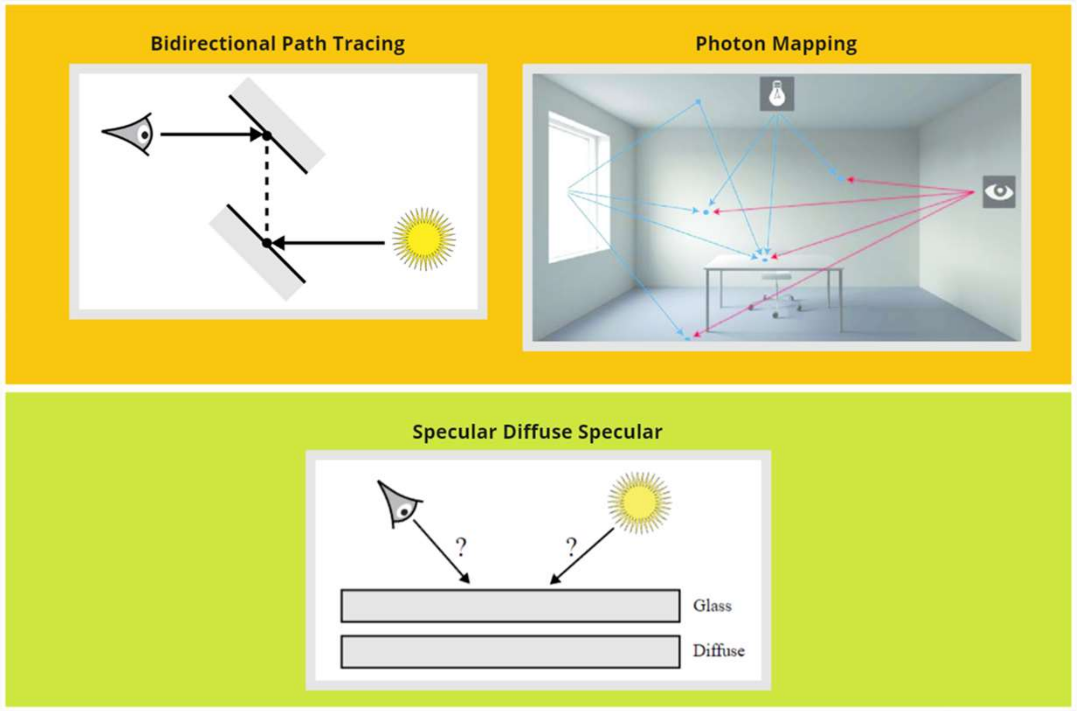
## Related Work

**Bidirectional Path Tracing (BDPT)**

- Good at rendering scene with hard-to-reach light source
- Has difficulty in specular-diffuse-specular (SDS) configurations

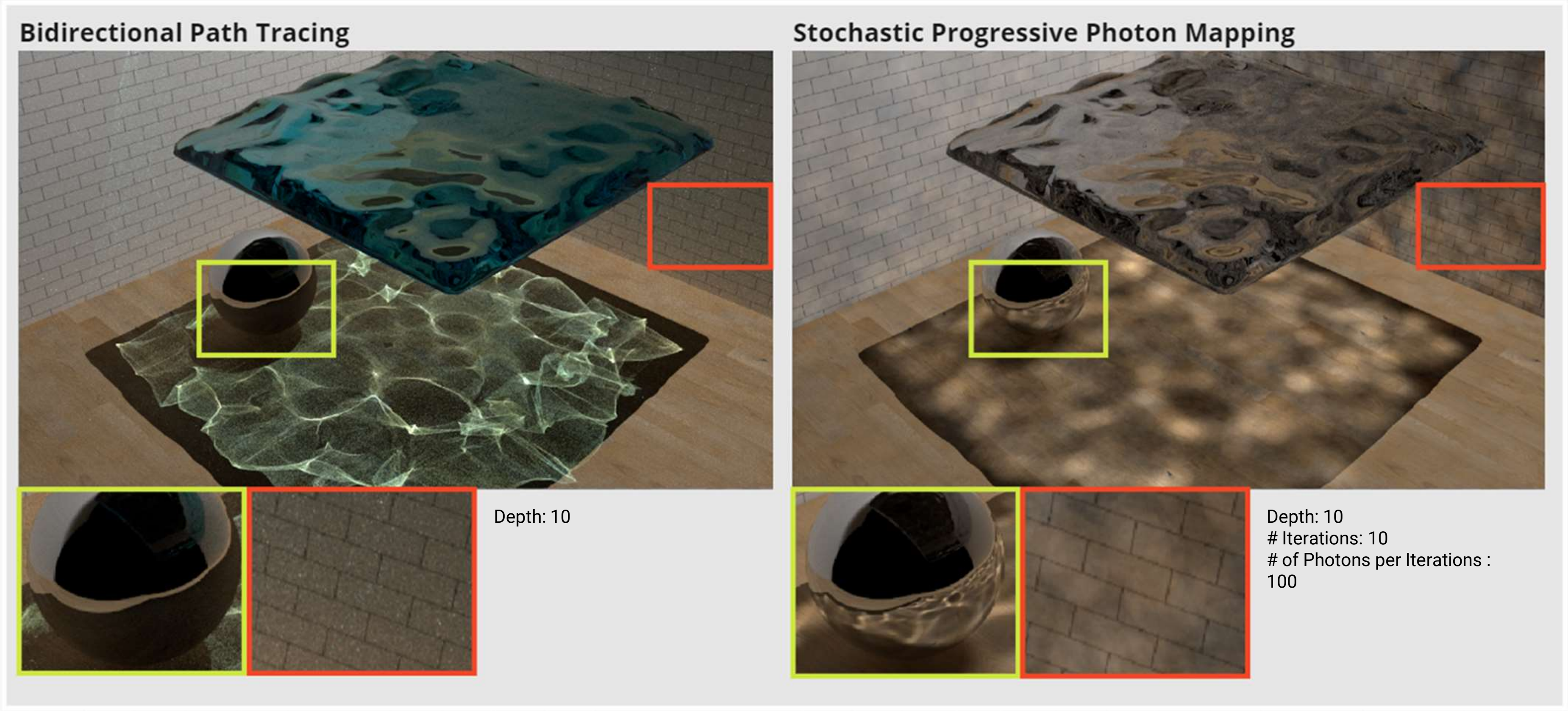
**Photon Mapping (PM)**

- Cache "photons" at surfaces
- Eye paths retrieve them with density estimation
- Good at rendering SDS surfaces
- Inefficient under diffuse lighting



pbrt 3rd edition 2018, SPPM Figure 16.6, pbrt 3rd edition 2018, 16.3 BDPT Figure 16.13, https://www.researchgate.net/figure/Illustration-of-the-photon-mapping-calculation-method_fig5_283357163

# Related Work


Bidirectional Path Tracing
Stochastic Progressive Photon Mapping

Bidirectional Path Tracing

Depth: 10

Stochastic Progressive Photon Mapping

Depth: 10
# Iterations: 10
# of Photons per Iterations : 100

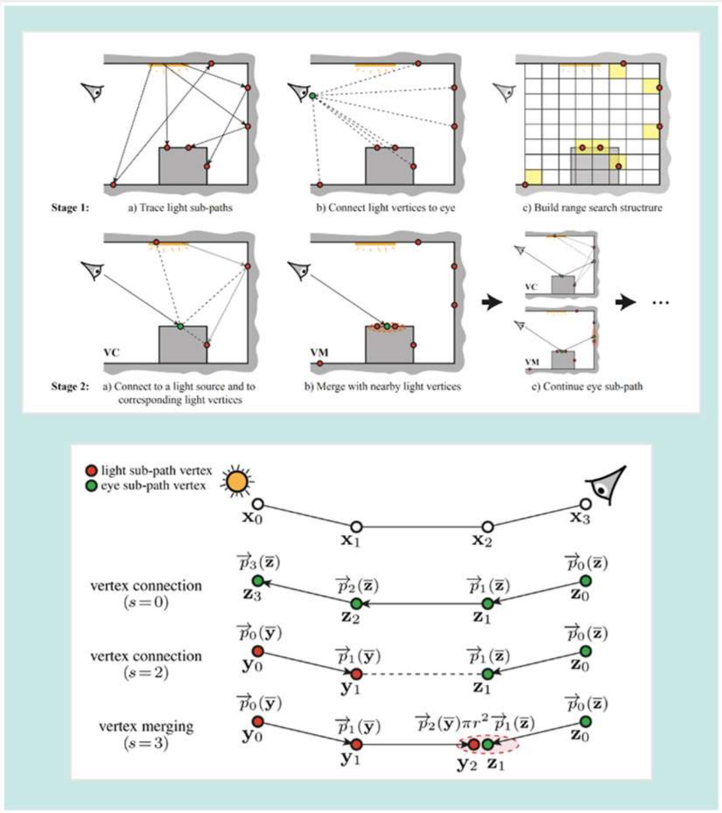
## Related Work

**Vertex Connection and Merging (VCM)**

- Combination of BDPT and PM
- Uses both Vertex Connection in BDPT and Vertex Merging in PM
- Can handle complex lighting situations

Useful for...? Water caustics!

This is why we chose to implement VCM



http://iliyan.com/publications/ImplementingVCM

## Related Work

What about methods that are based on VCM?

For example:

- Efficient Caustic Rendering with Lightweight Photon Mapping (Grittmann et al., 2018)
- Implementing One-Click Caustics in Corona Renderer (Šik & Křivánek, 2019)

They are better and more efficient than VCM.

But we didn't choose them because implementing them requires us to first implementing VCM

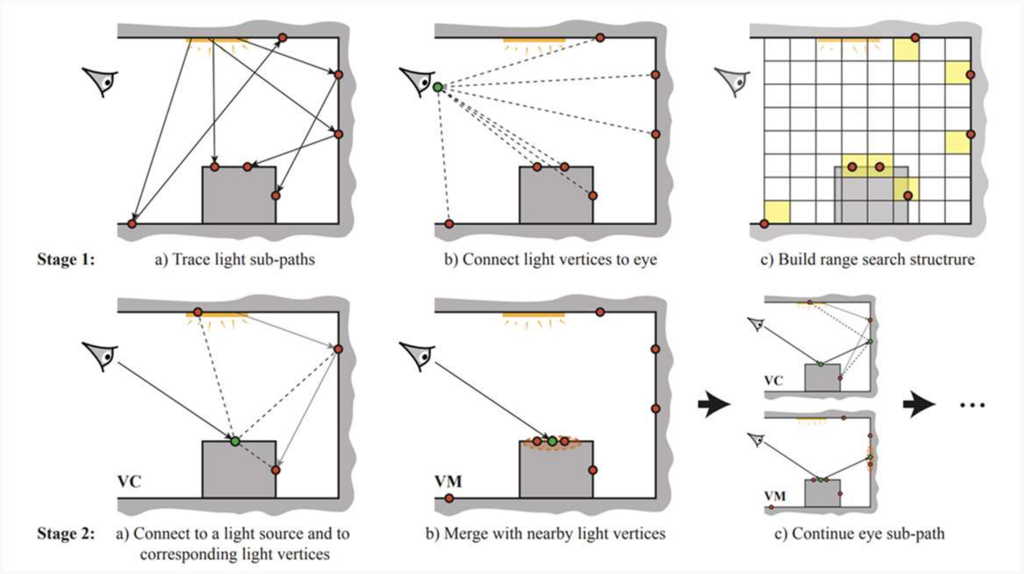
→ Implementing multiple methods at once (Not a good idea!)

## Approach

## Approach

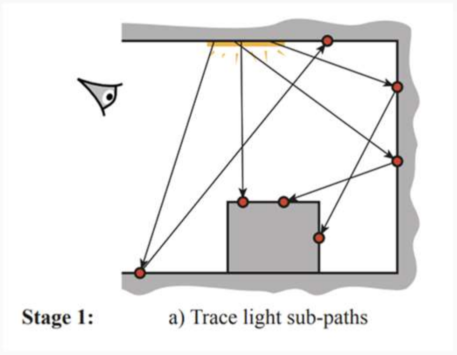
We implemented VCM as a module that can be put into any pbrt builds and used with any input files

Here is a procedure diagram from the VCM paper. We followed it closely in our implementation

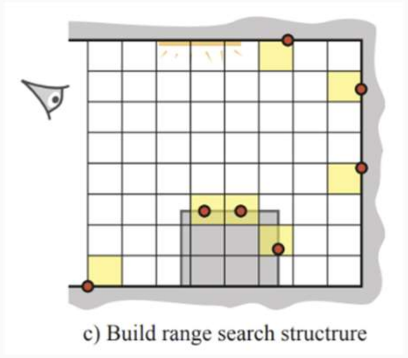

Stage 1:
a) Trace light sub-paths
b) Connect light vertices to eye
c) Build range search structrure
Stage 2:
VC
a) Connect to a light source and to corresponding light vertices
VM
b) Merge with nearby light vertices
VC
VM
c) Continue eye sub-path
...

http://iliyan.com/publications/ImplementingVCM

## Stage 1a - Tracing light sub-paths



Later:



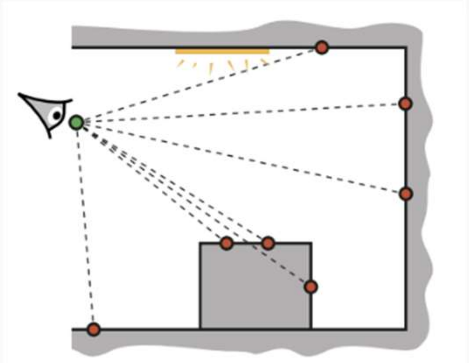
Constraints:

- Need to maintain the path structure for doing vertex connection
- Need to store individual vertices in a range search structure for doing vertex merging

Implementation:

- First try
  - Tried to adopt pbrt's SPPM as it already has a grid structure
  - Difficult to retrofit the vertex data needed (e.g., forward/backward pdfs) for vertex connection
- Second try
  - Start from pbrt's BDPT implementation
  - Store whole light paths in an array of vector

## Stage 1b - Connect light vertices to eye


b) Connect light vertices to eye

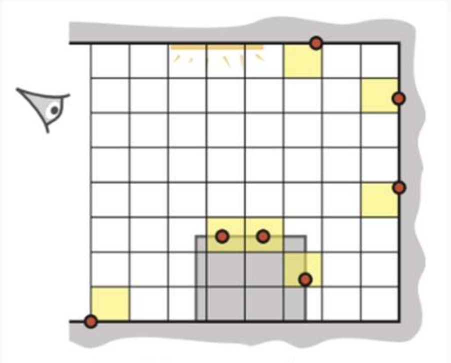
Why?

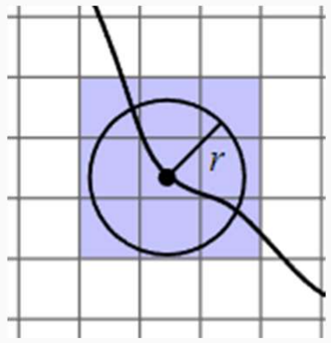
- Camera lens very small (or infinitely small for a pinhole camera)
  - Probability of tracing a light path that ends at the camera is 0
- So, connect light vertices directly to camera

We did not handle this step explicitly

- In a sense: a "vertex connection" step
- pbrt's BDPT handles this special case together with other vertex connection cases
- We follow the same approach and defer this step

## Stage 1c - Build range search structure


c) Build range search structure

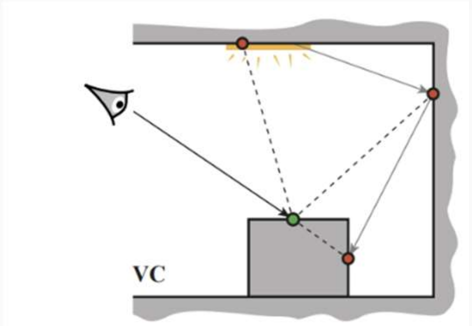


Pbrt Ch16.2.5 Fig 16.10

Store the individual vertices for vertex merging

- Kd-tree or voxel grid
  - We chose voxel grid: easier to implement
  - This is also what the authors used in the VCM paper
- Implementation
  - Remember we already have all the light paths stored
  - Loop through all light vertices to get the bound for the grid
  - Divide the grid into cells with size roughly = merge radius
  - Implement the grid as a vector with a hash [(x, y, z) ➔ index]
  - Hash each vertices into the vector, store as a linked list
  - List nodes use pointers to point to vertices in the light path vectors
    - Memory efficiency

## Stage 2a - Vertex Connection (VC)



**Stage 2:** a) Connect to a light source and to corresponding light vertices

I.e., what traditional BDPT does

Remember the special case in Stage 1a (light path to camera)?

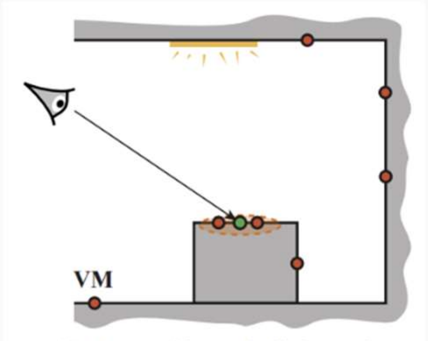
Another special case:

- Low probability for a camera ray to hit a light source
- Connect to light source directly

Implementation:

- pbrt's BDPT implementation handles both special cases and normal vertex connection
- We adopt its approach
- With modification to how each connection's Multiple Importance Sampling (MIS) weight is calculated (more later)

## Stage 2b - Vertex Merging (VM)



b) Merge with nearby light vertices

For each vertex in a camera ray, accumulate the contribution of light vertices nearby

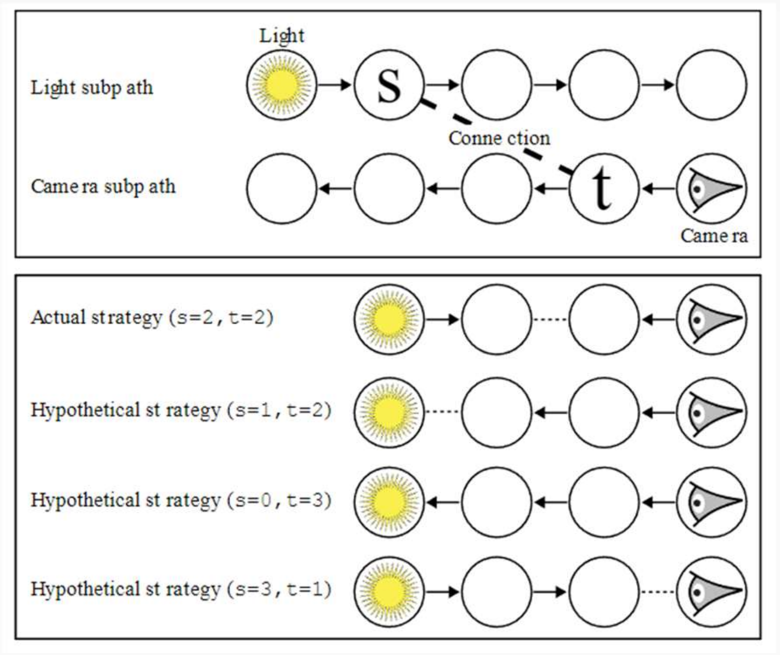
Use the grid we built in Stage 1c

- Get the index into the grid vector using the hash function
  - [(x, y, z) ➔ index]
- Loop through all light vertices stored at that bucket
  - Check if within radius
    - Possible out-of-range vertices due to hash collision
  - Accumulate each light vertex's radiance
    - Weighted by the camera ray's throughput so far
    - Weighted by MIS weight

## Multiple Importance Sampling (MIS)

Every complete path created by either VC or VM needs to be weighted by a factor

- There can be multiple ways to sample for the same path
  - Various connection points for a VC path
  - Various merging points for a VM path



Light
Light subp ath
S
Conne ction
Came ra subp ath
t
Came ra
Actual strategy (s=2, t=2)
Hypothetical st rategy (s=1, t=2)
Hypothetical st rategy (s=0, t=3)
Hypothetical st rategy (s=3, t=1)

# Multiple Importance Sampling (MIS)

The MIS weight has the form of:

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

Depending on the sampled strategy, can be separated into:

VC

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

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

VM

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

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

And further into:

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

⋮

## Multiple Importance Sampling (MIS)

A more efficient way to implement it is to store partial values at each vertex

- No summation needed at evaluation time
- This is a new approach proposed by the VCM authors

$$\mathbf{y}_1 : d_1^{\text{VCM}} = \frac{p_0^{\text{connect}}}{p_0^{\text{trace}}} \frac{1}{\overrightarrow{p}_1} \qquad \mathbf{z}_1 : d_1^{\text{VCM}} = \frac{p_0^{\text{connect}}}{p_0^{\text{trace}}} \frac{n_{\text{light}}}{\overrightarrow{p}_1}$$

$$d_1^{\text{VC}} = \frac{\overleftarrow{g}_0}{p_0^{\text{trace}} \overrightarrow{p}_1} \qquad d_1^{\text{VC}} = 0$$

$$d_1^{\text{VM}} = \frac{\overleftarrow{g}_0}{p_0^{\text{trace}} \overrightarrow{p}_1 \eta_{\text{VCM}}} \qquad d_1^{\text{VM}} = 0$$

$$\mathbf{y}_i, \mathbf{z}_i : d_i^{\text{VCM}} = \frac{1}{\overrightarrow{p}_i}$$

$$d_i^{\text{VC}} = \frac{\overleftarrow{g}_{i-1}}{\overrightarrow{p}_i} \left( \eta_{\text{VCM}} + d_{i-1}^{\text{VCM}} + \overleftarrow{p}_{\sigma,i-2}\, d_{i-1}^{\text{VC}} \right)$$

$$d_i^{\text{VM}} = \frac{\overleftarrow{g}_{i-1}}{\overrightarrow{p}_i} \left( 1 + \frac{d_{i-1}^{\text{VCM}}}{\eta_{\text{VCM}}} + \overleftarrow{p}_{\sigma,i-2}\, d_{i-1}^{\text{VM}} \right)$$

We did not follow this approach

- We tried, but found out it's not a good fit with how pbrt's existing functions work
- Instead, we implemented the summation formulas

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

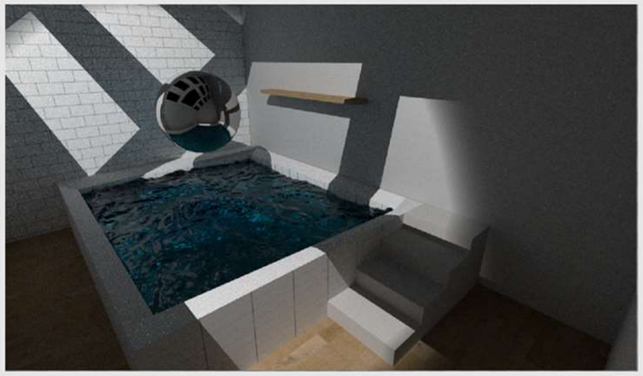
$$\vdots$$

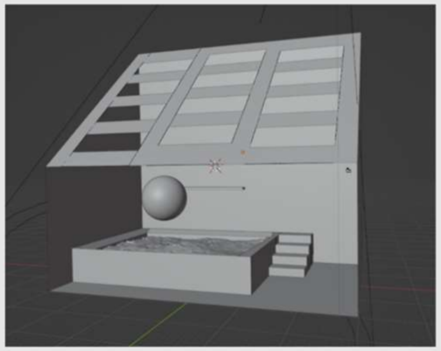
## Scene Creation

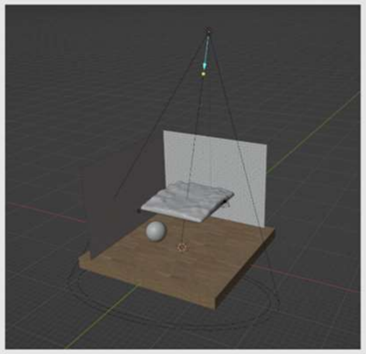
To Capture different caustic scenarios occurring

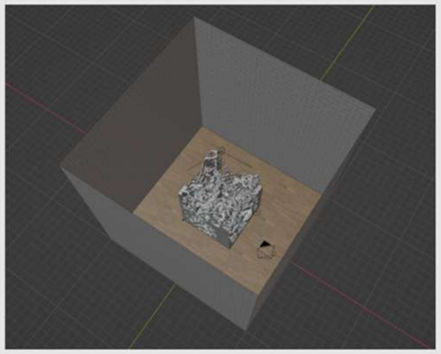
- below water surfaces
- on nearby walls

Testing our VCM methods against BDPT and SPPM.

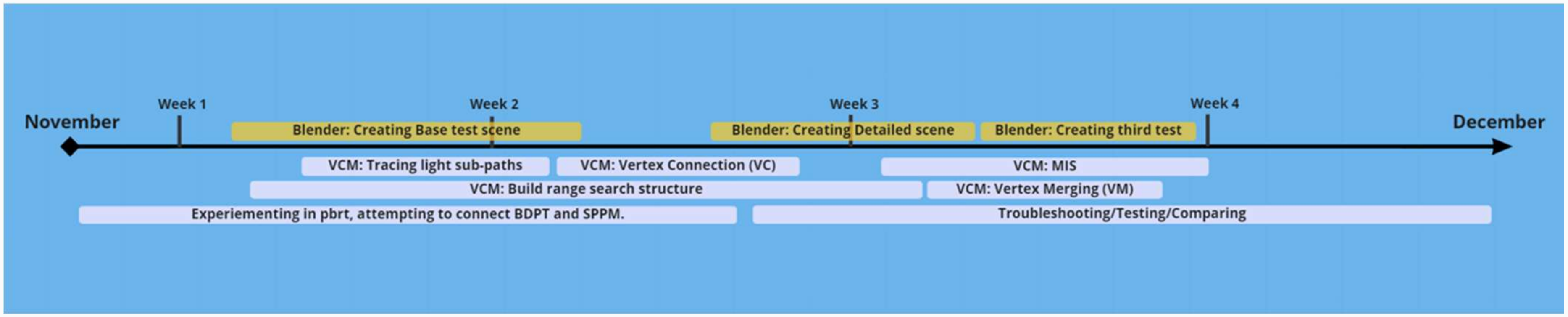








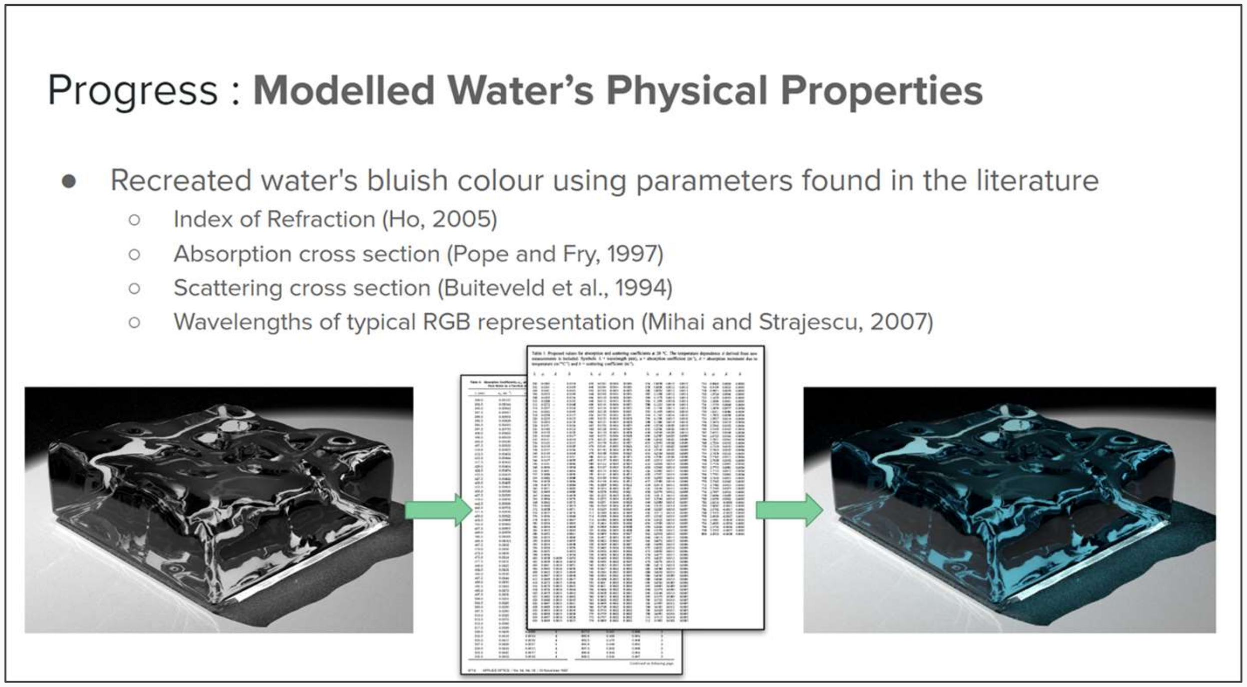
## Timeline


November
Week 1
Week 2
Week 3
Week 4
December
Blender: Creating Base test scene
Blender: Creating Detailed scene
Blender: Creating third test
VCM: Tracing light sub-paths
VCM: Vertex Connection (VC)
VCM: MIS
VCM: Build range search structure
VCM: Vertex Merging (VM)
Experiementing in pbrt, attempting to connect BDPT and SPPM.
Troubleshooting/Testing/Comparing

## Results & Analysis

## Water's bluish colour

Already done at the time of the project milestone!



Progress : **Modelled Water's Physical Properties**

- Recreated water's bluish colour using parameters found in the literature
  - Index of Refraction (Ho, 2005)
  - Absorption cross section (Pope and Fry, 1997)
  - Scattering cross section (Buiteveld et al., 1994)
  - Wavelengths of typical RGB representation (Mihai and Strajescu, 2007)

## Results & Analysis

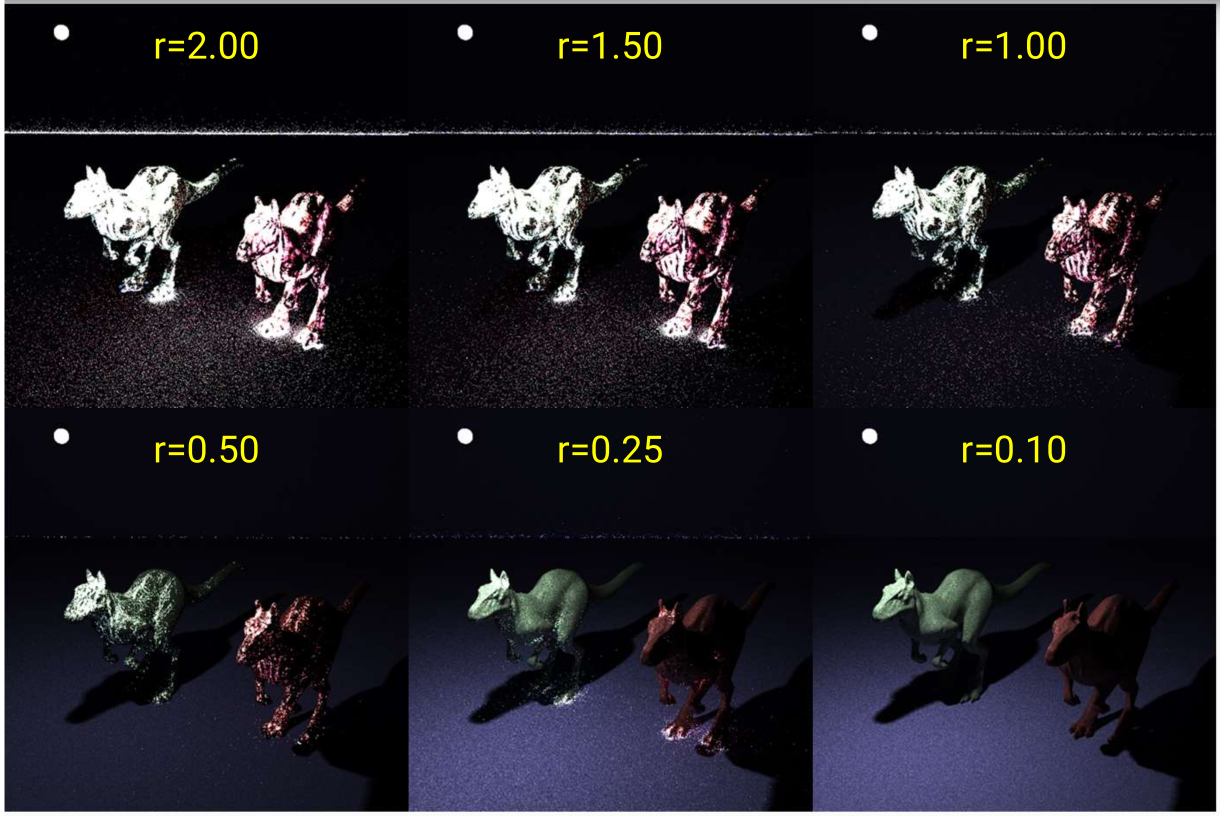
Unfortunately, we weren't able to fully implement VCM.

- In particular, the MIS weight for Vertex Merging in our implementation has errors
- We were still trying to fix it

In this section, we will demonstrate how we narrowed down the error to this exact component by:

- Comparing our rendered images with reference images rendered with pbrt's stock algorithms
- Using scenes downloaded from pbrt's website and scenes we created
- And showing that each of the other components are working properly

## Analysis 1 voxel grid and radius measurement are working properly



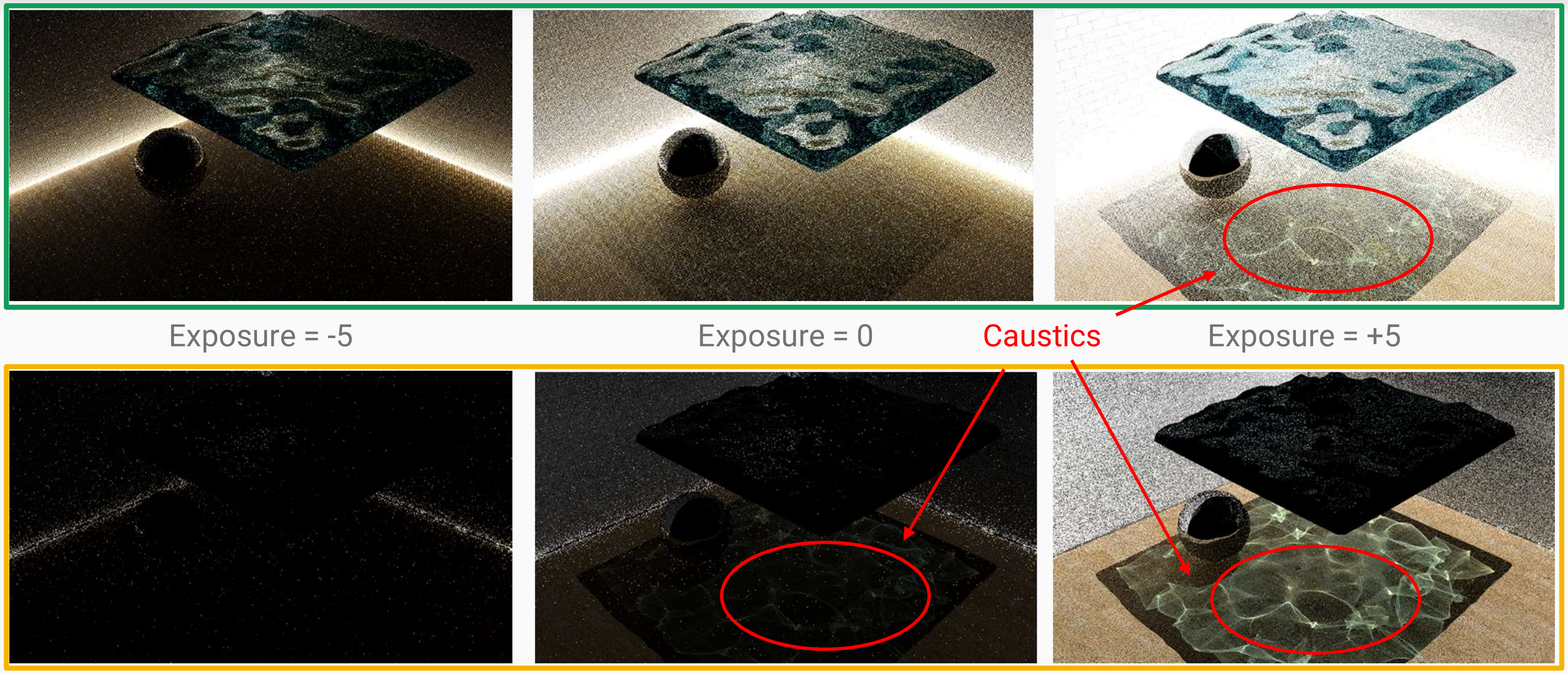
Scene: pbrt's default scene

Same rendering parameters except r
- Sobol sampler
- Samples per pixel = 16

As we reduce the merging radius:
- Fewer bright spots
- I.e., fewer light vertices accumulated at each camera vertices

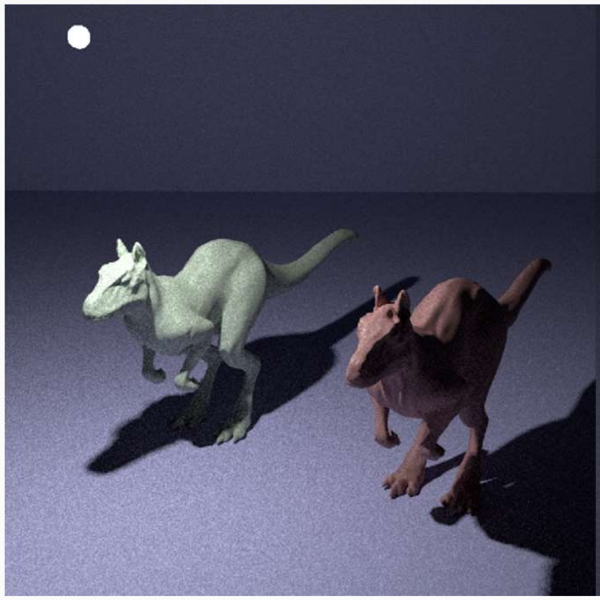
- The voxel grid and radius measurement are working as expected

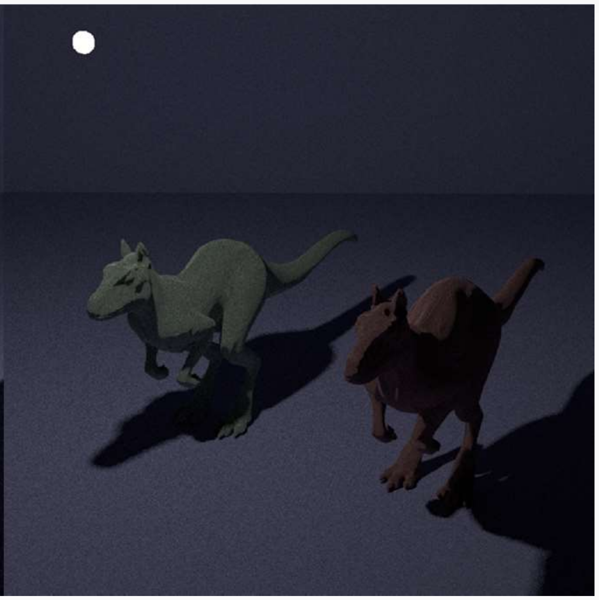
## Analysis 2 camera and eye sub-path tracing through scattering medium are working properly



Our custom scene rendered using our VCM. Top: 4 samples per pixel, depth 20; Bottom: 8 samples per pixel, depth 15

## Analysis 3 MIS weight for Vertex Connection is working properly

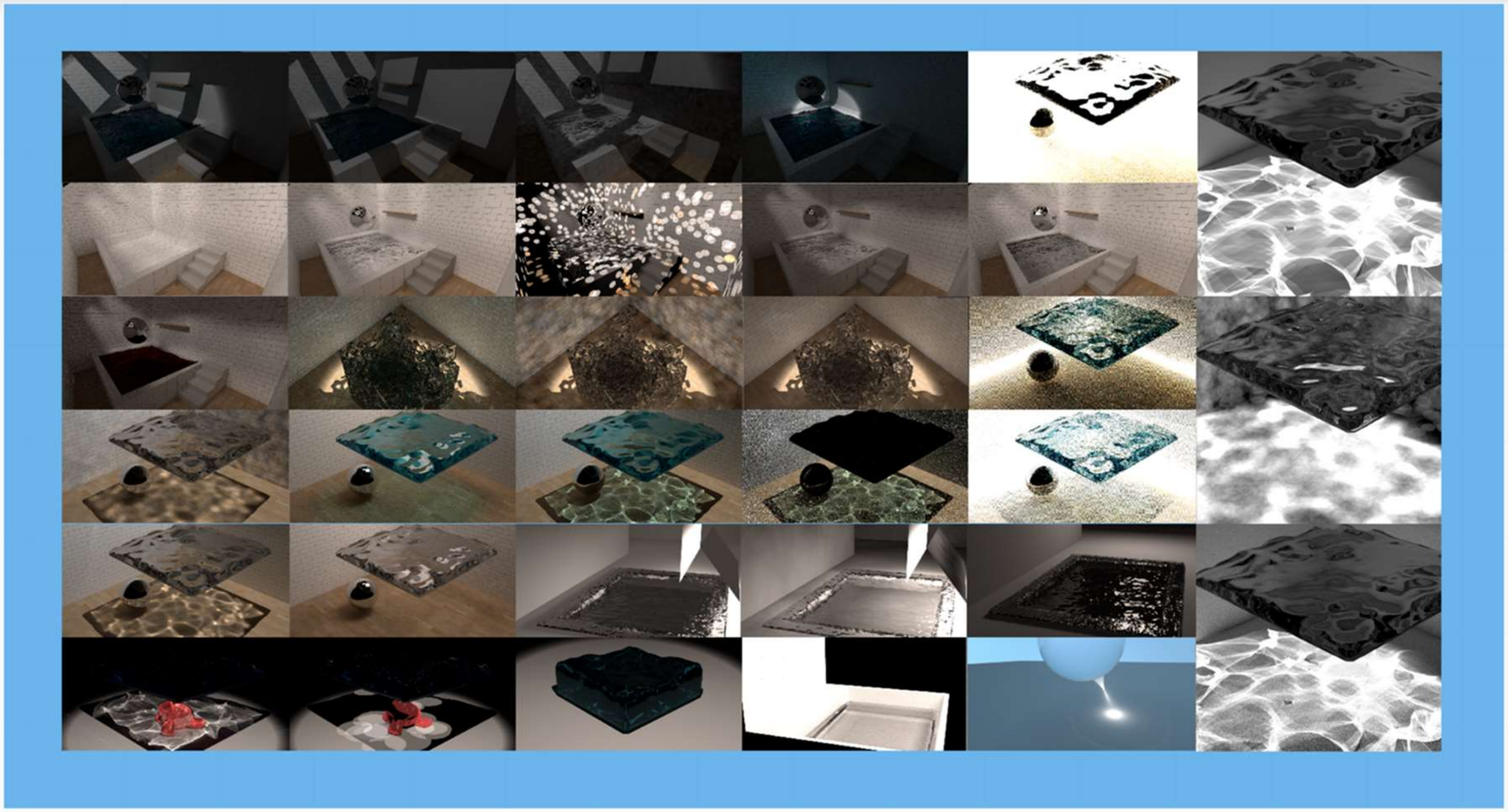

Reference rendered with pbrt's BDPT


Rendered using our VCM with Vertex Merging's contribution explicitly clamped to zero

We rendered this scene with Vertex Merging's contribution (the problematic component) set to zero.

- The rendered scene looks darker than reference
- Indicates that the MIS weight for Vertex Connection is working as expected to scale down VC's contribution

## Other scenes we tried to render



## Conclusion

## Encountered Challenges

Figuring out the Maths behind the MIS

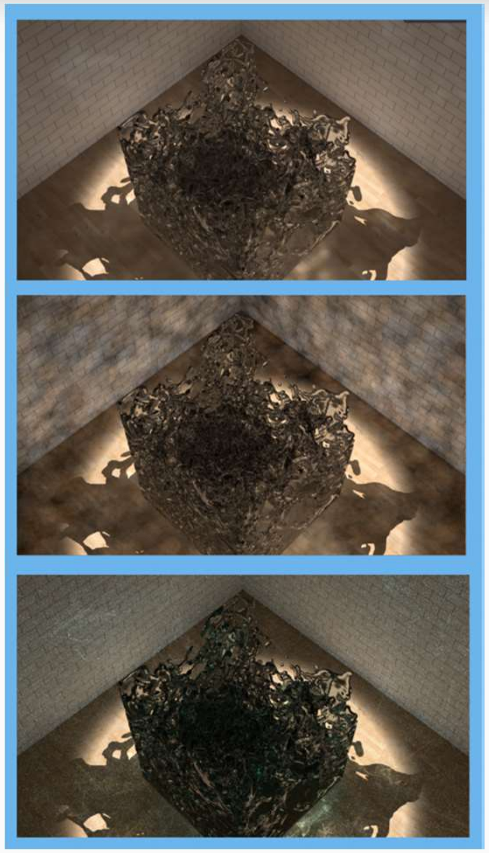
- Especially difficult to adapt it to pbrt's existing systems

Producing Caustic Effects on nearby walls

- Very hard to differentiate between caustic effects and renderings not completely converging
- Hard to see caustic effects occurring on walls even for the stock pbrt algorithms (BDPT, SPPM, ...)

Images take too long to render

- Only to find out that it's not working
- Should have implemented a way to save images half way through rendering



## Take away & What we learned

What we learned

- Multiple Importance Sampling's underlying rationale, balance/power heuristics
- BDPT, PM, and VCM's rendering approach and their mathematical frameworks
- Sampling techniques: Halton, Sobol, etc.
- How pbrt (the whole thing) works
- Blender scene creation

Take away

- There are many approaches to the same problem
    - There is not necessarily a single best one
- Things can go wrong
    - Allocate extra, extra, extra time to account for anything unexpected
    - Even if we couldn't 100% complete the project, we still learned a lot on the way
        - Very useful for the future

## What still needs to be done?

- Fix the MIS weighting for Vertex Merging
- Maybe reimplement the MIS using the more efficient approach
- Implement a debug option to save images half way through rendering
- Look into rendering caustics on nearby walls