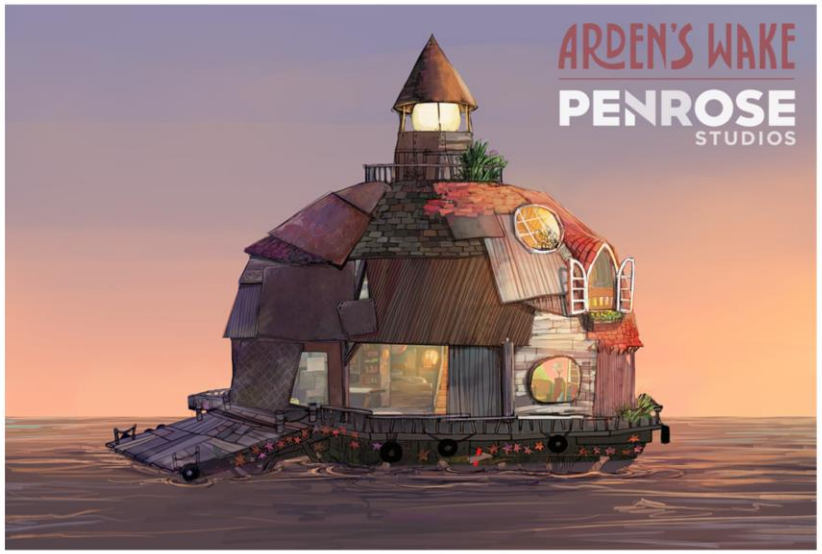
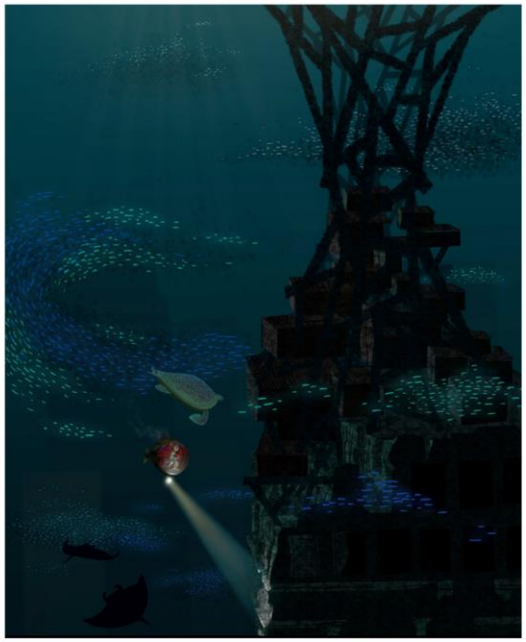
Devon Penney - Head of engineering - Penrose Studios

Twitter: @dmpvfx

This talk is about water simulation and rendering in Arden’s Wake





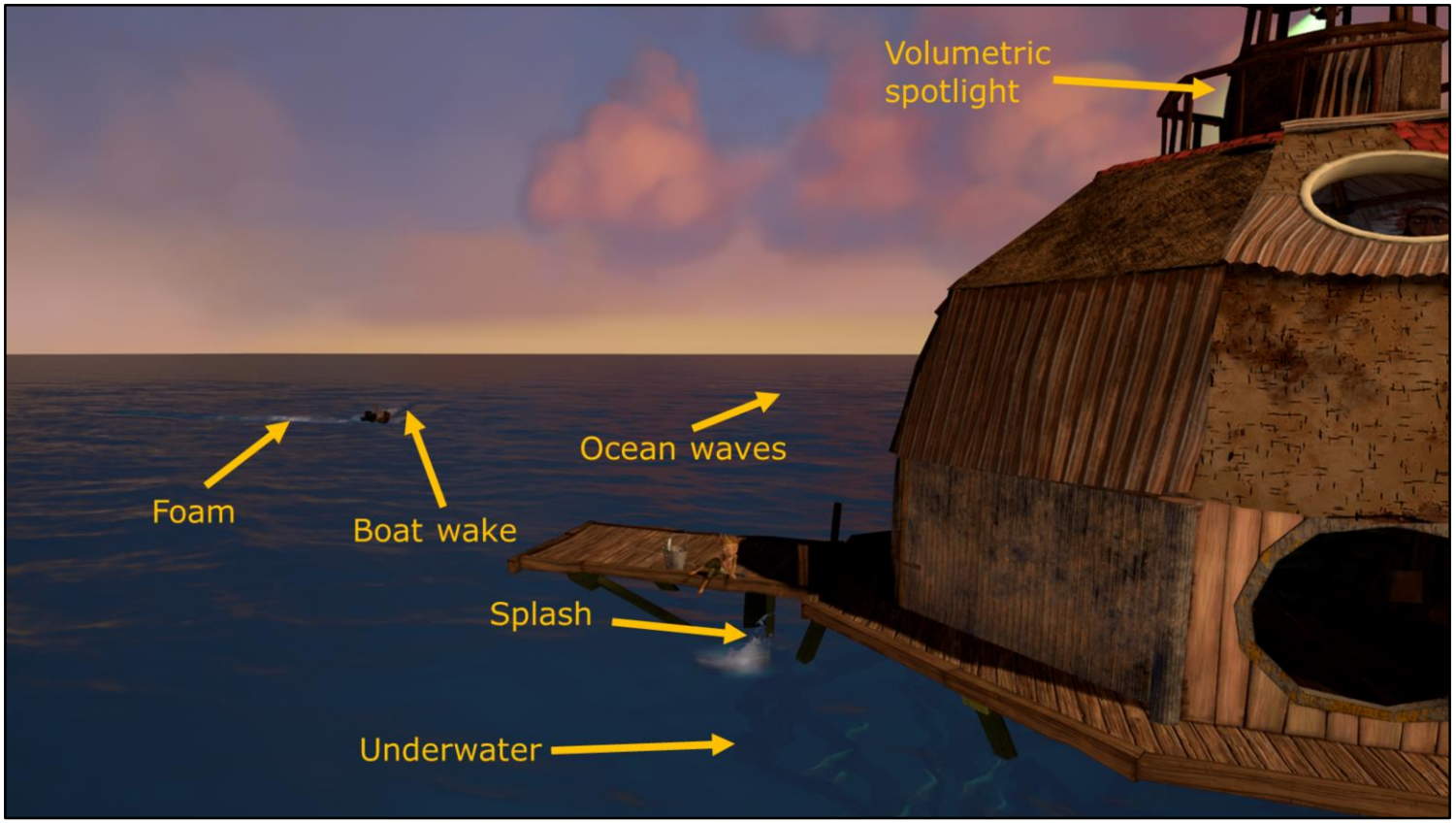
The entire set of Arden's Wake is on or below an infinite world of water.

Things interact with the water surface (boats, splashes, etc).

Underwater scenes have key volumetric lighting elements.

There is a lot of complexity in the non water assets, such as characters, sets, FX, and set lighting, so all solutions need to be high performance.

Here is a screenshot from a bottleneck scene where there is a lot going on.



Here are all the elements this talk will cover, highlighted with yellow arrows.

Foam – dynamic whitewater simulation coming off the boat wake

Boat wake – dynamic fluid simulation using a solution to the shallow water equations

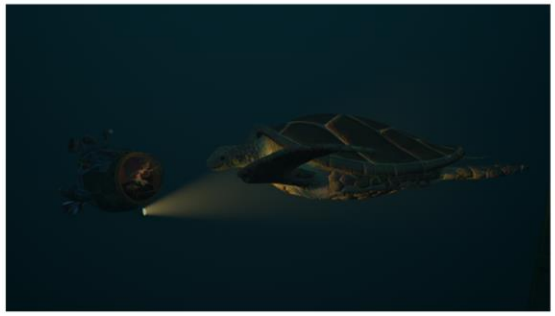
Ocean waves – Ambient ocean waves

Splashes – Baked out and dynamic splash simulations

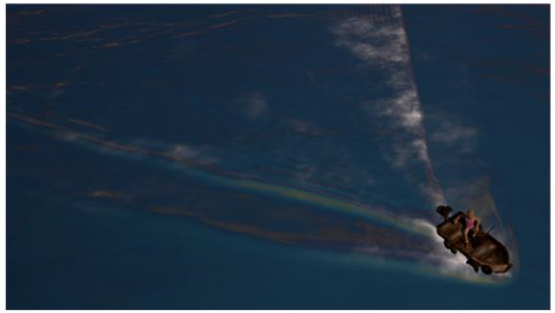
Underwater rendering – Volumetric passes for the murky underwater look

Volumetric spotlights – Seem in green in the upper right hand corner, used in a variety of contexts

Underwater Rendering



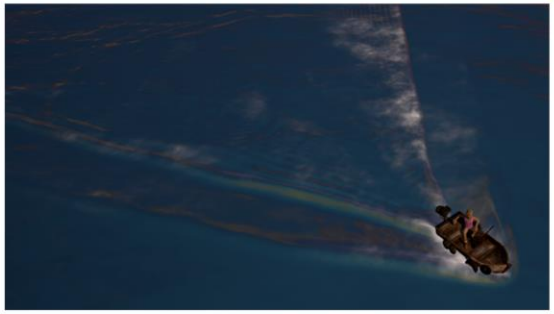
Water Simulation



We will cover both the underwater rendering and water simulation done for Arden's Wake.

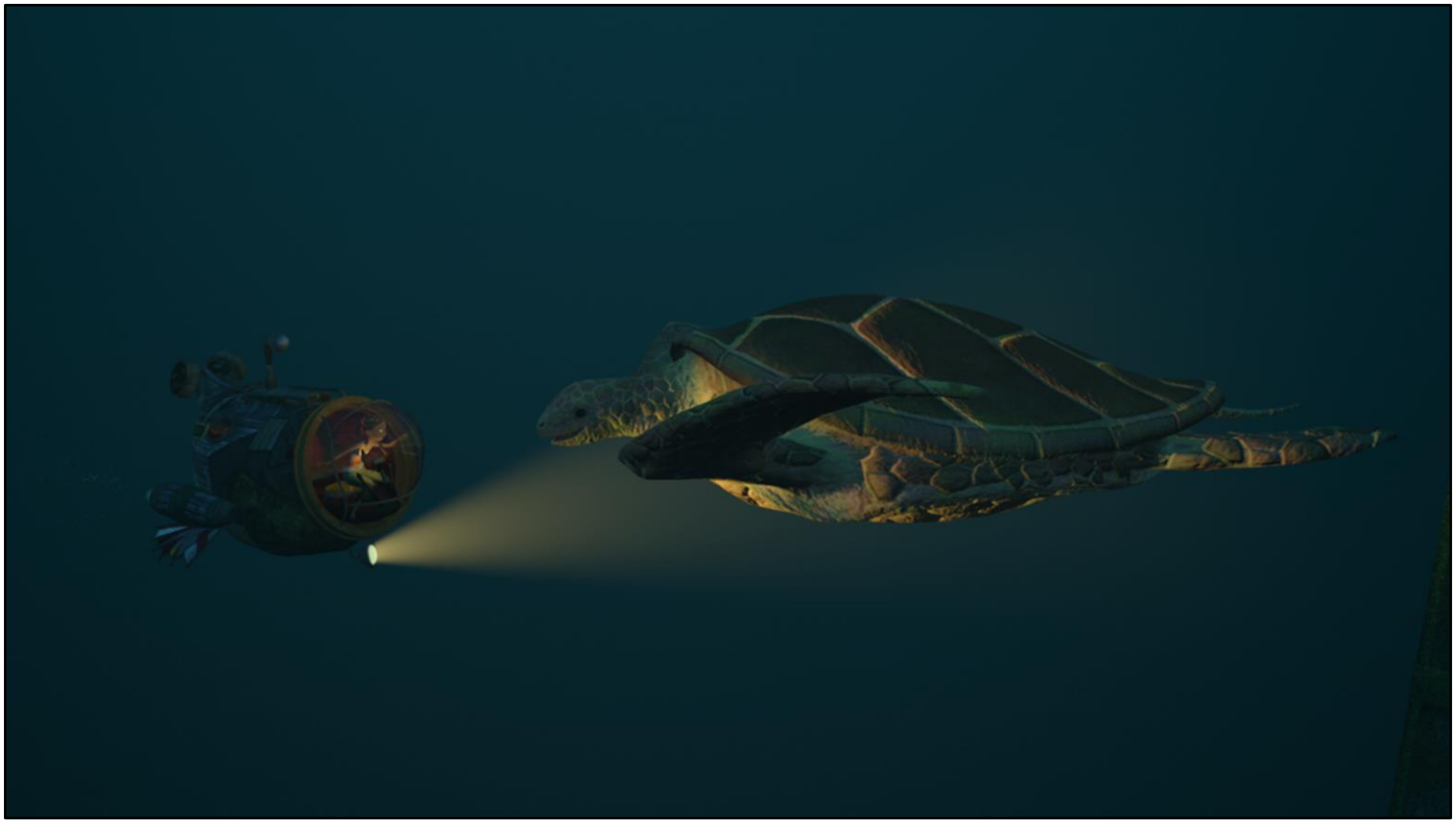
Underwater Rendering

Water Simulation



We'll first talk about the underwater rendering.

This is some artwork from Arden's Wake. We wanted a fully volumetric underwater look with realistic sunlight filtering through the water, fully volumetric spotlights, and compositing with FX and translucent geometry.



Here is an actual render from Arden’s Wake that demonstrates the properties mentioned in the previous slide.

Directional Lights

Inspiration

- [Hoobler 16] – Volumetric rendering at Nvidia
- [Hillaire 15] – Volumetric rendering in Frostbite

Sunlight illuminating water

Colored transmittance

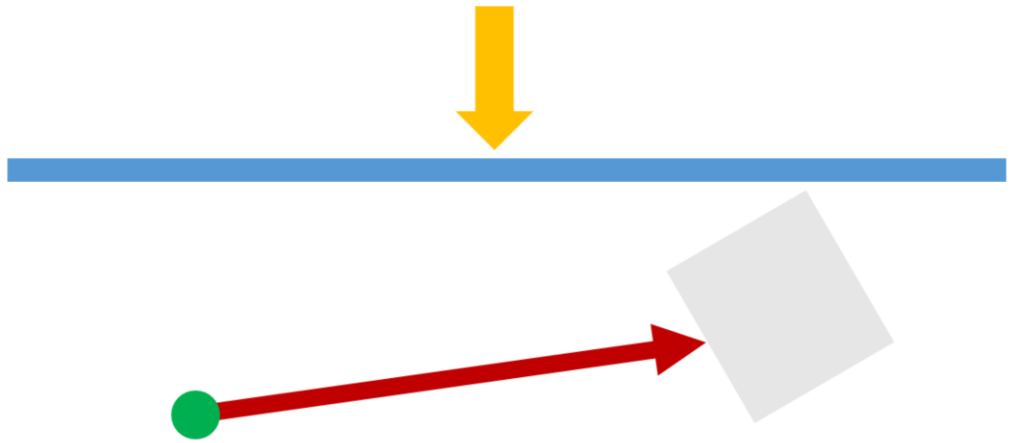
Analytical solution

We needed to have environment lighting from the sun with light that transmits through the water.

Inspiration – Nathan Hoobler and Sebastian Hillaire gave great explanations of volume rendering, and their slides are available online.

We used a directional light formulation, built off of existing work, but extended it with a closed form solution in a constrained setup.

Directional Lights



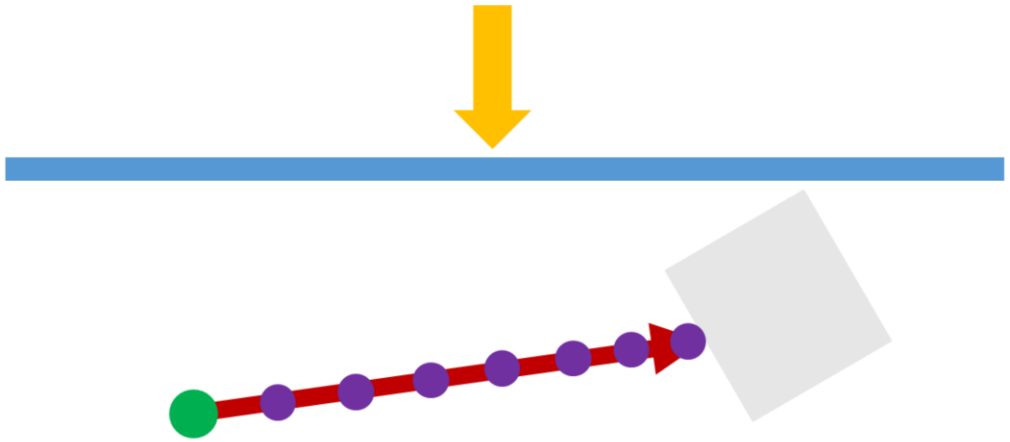
On the top, the yellow arrow represents the direction of the light, which is constrained to be in the –z direction.

The blue line is the water surface height in z.

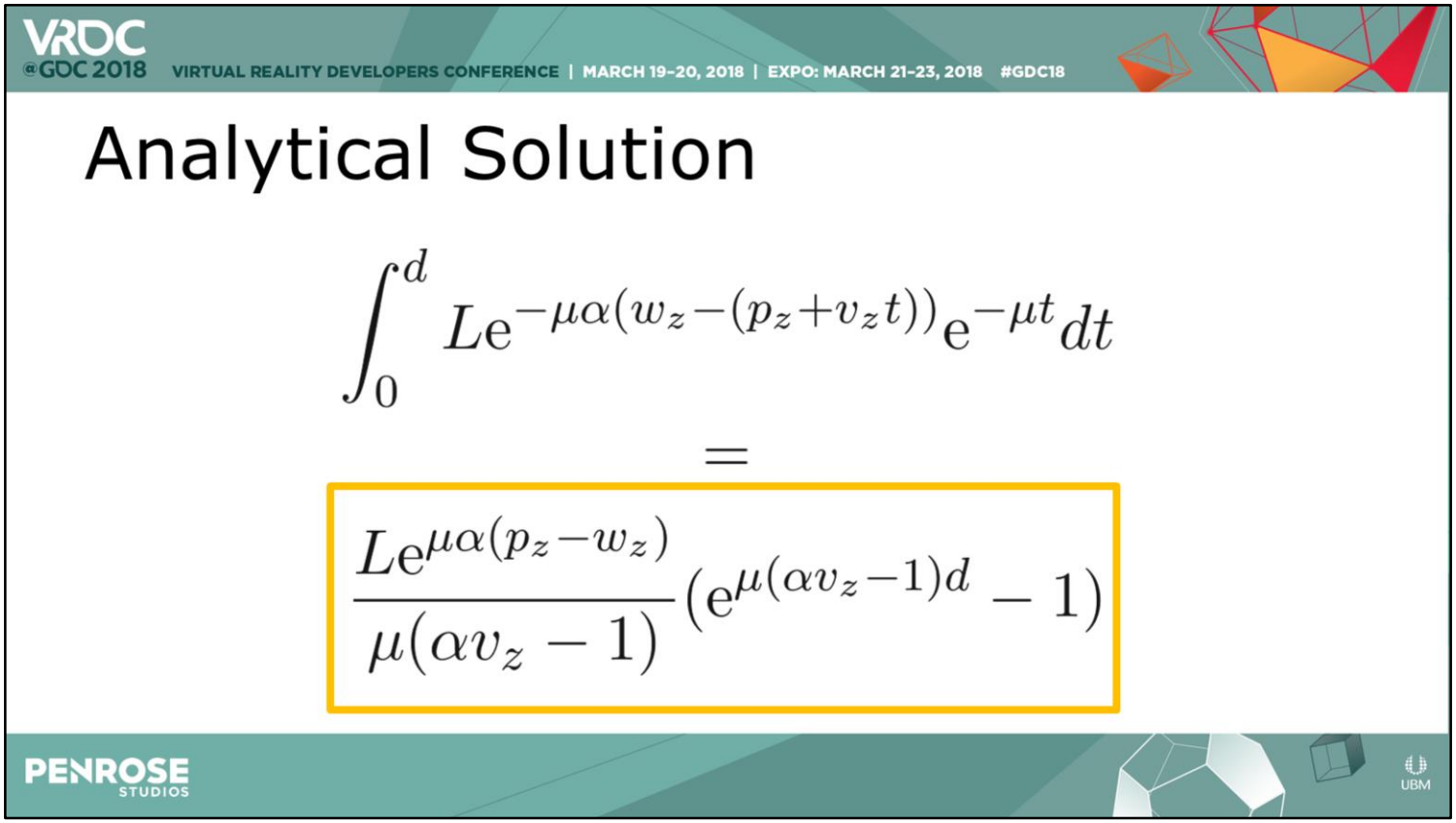
The ray we are computing the lighting for starts at the green circle and follows the red arrow.

The gray object is underwater, and terminates the ray.

Numerical Solution



In many rendering solutions, we'd ray march this, computing the integrand of the integral defining the lighting equation for this setup at each purple dot along the red ray. For underwater environments, this can be prohibitively slow for a real time solution.



If we solve the above integral, we get a closed form solution for the directional light described.

Here are the parameters for the integral:

d – maximum distance along the ray until the ray terminates.
L – Radiance of the directional light
mu – density of the medium
alpha – density multiplier for the attenuation of the light as it travels from the water surface down through the water
w_z – water height in z
p_z – z coordinate of the ray start
v_z – z component of the direction of the ray
t – distance along the ray

The analytical solution can be coded simply in a shader with about 30 instructions rather than 2000+ for a slow ray marching solution with 100 samples that still gives artifacts.

This is an example of how the color shifts as you get deeper underwater. This video (and others described in these slides) can be seen in the recording of the talk.

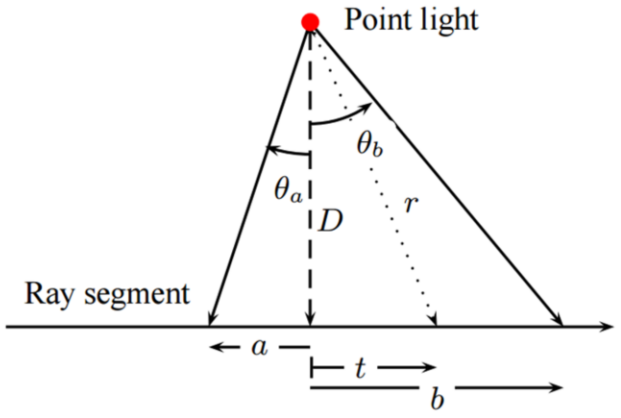
Spotlights

Attempts at sampling

- Brute force marching
- Trapezoidal rule
- Different quadrature

Equiangular sampling

[Kulla 11]



The need to view the spotlights from all angles means we must have good integration schemes.

No proxy geometry or cards will be usable due to artifacts in VR, so must use a fully volumetric solution.

There is no closed form analytical solution like previous slides, but one can use look up tables for some formulations of spotlights.

Brute force ray marching lead to artifacts.

We settled for equiangular sampling, which is a way of importance sampling lights in homogenous mediums. The technique places samples in an intelligent way such that they have a maximal impact on the final integration result. It is also very easy to code and very efficient.

Trapezoidal Rule

This is rendering a spotlight from the perspective of the red arrow on the right using trapezoidal rule quadrature for ray marching.

Note the noticeable edge in the beam with the yellow arrow pointing to it.

This is an artifact due to undersampling, but we can't afford more samples.

Trapezoidal Rule + Equiangular

This example uses the same number of samples in the previous slide, but with equiangular sampling.

Note how the beam now looks smooth and there is no artifact.

There is no noticeable performance degradation due to the simplistic implementation.

Results

16 samples per ray per spotlight

~.1-.2ms per eye per spotlight

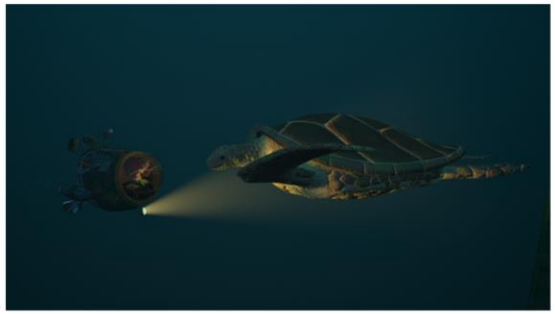
.33ms per eye

- directional light
- 1 spotlight
- compositing
- dithering [Gjoel 16]
- ambient light

Overall, the volume rendering pass takes .33 ms per eye.

Dithering is important due to how dark our sets are, and the talk slides cited about the rendering for the game Inside gives great insight into this topic.

Underwater Rendering



Water Simulation

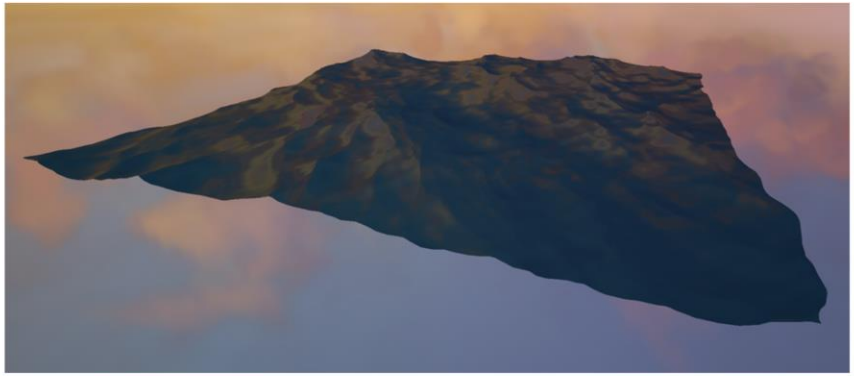
Next we will discuss the water simulation techniques used on Arden’s Wake.

Ocean Waves

Procedural method [Tessendorf 01]

No repeated waves in time

Tiles nicely



Tessendorf's technique for ocean waves is great for this type of setup with a large ocean plane.

It doesn't need to be simulated, and you don't notice tiling and looping if setup correctly.

There are plenty of example implementations that can be found online.

Water Simulation

3d simulation (too slow)

bake out simulations (too much memory)

Various tricks (don't look good)

[Bridson 16] – Shallow water equations

For simulated elements, we needed to choose a solution that was lightweight but gave us nice properties like surface velocities.

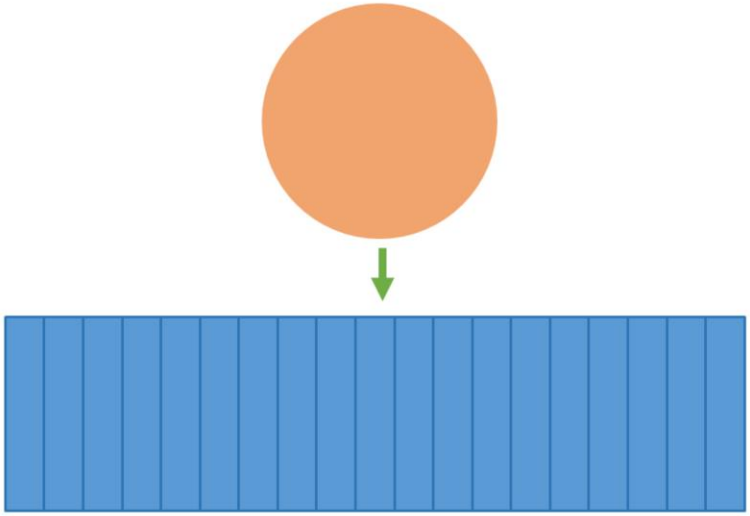
We choose to implement a shallow water simulator, which is a 2d fluid simulation technique.

The simulation domain is defined as a 2d grid of cells.

Cells are either occupied by water or colliders, and 2d velocities flow on the surface of the water.

Robert Bridson's book on fluid simulation for computer graphics has great descriptions of algorithms.

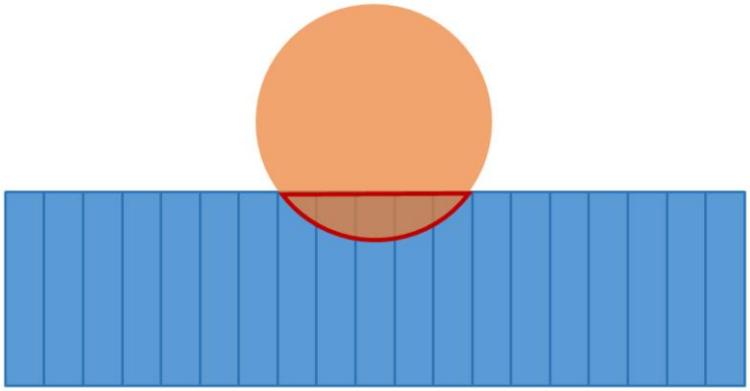
Initial State



We will walk through a simplified simulation step in a slice through the simulation domain with a ball dropping in some undisturbed water.

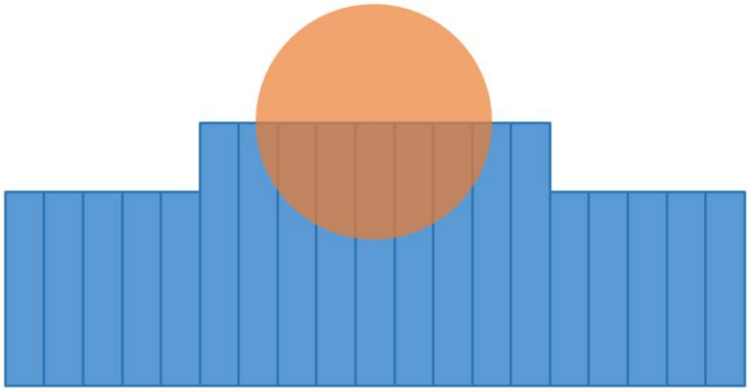
The orange ball is falling straight down into the fluid with a downward velocity in green.

Calculate Overlap Volume



Once the ball is intersecting the fluid, we must determine the overlap volume outlined in red.

Redistribute Overlap Volume

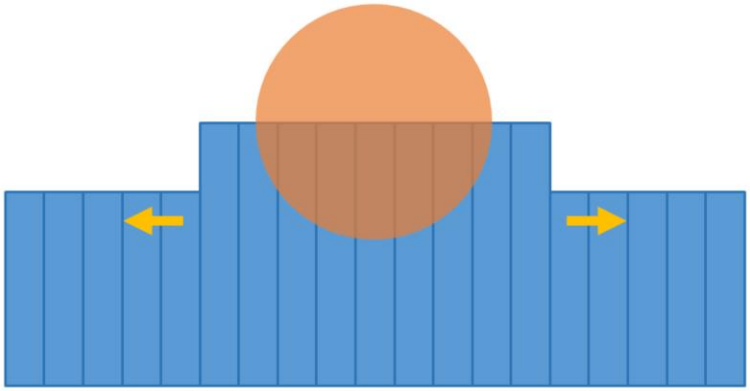


We take the overlap volume and distribute to neighboring cells.

Note that for performance reasons, we limit the distance we distribute.

Also, note cells that overlap the collider get a height value because we extrapolate height into colliders for velocity stability. Please see referenced literature for more information on this.

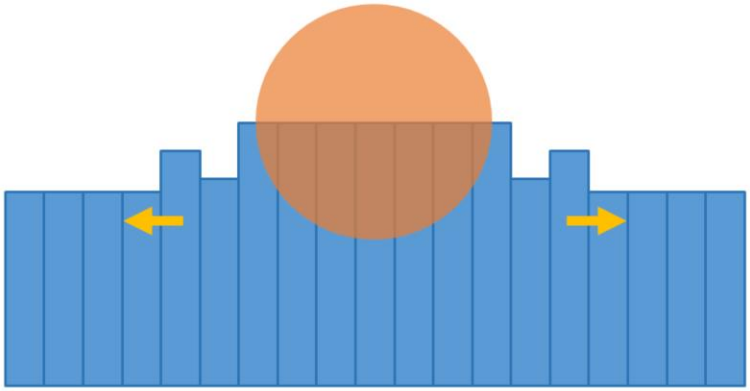
Update Velocity



We update velocity based on a finite difference of the height.

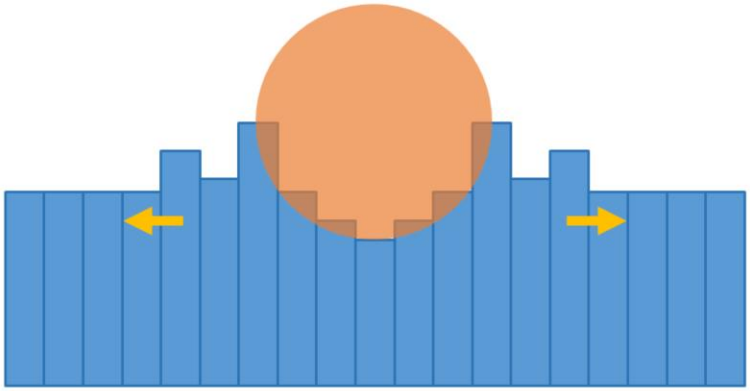
We get velocities propegating outward as expected.

Update Height



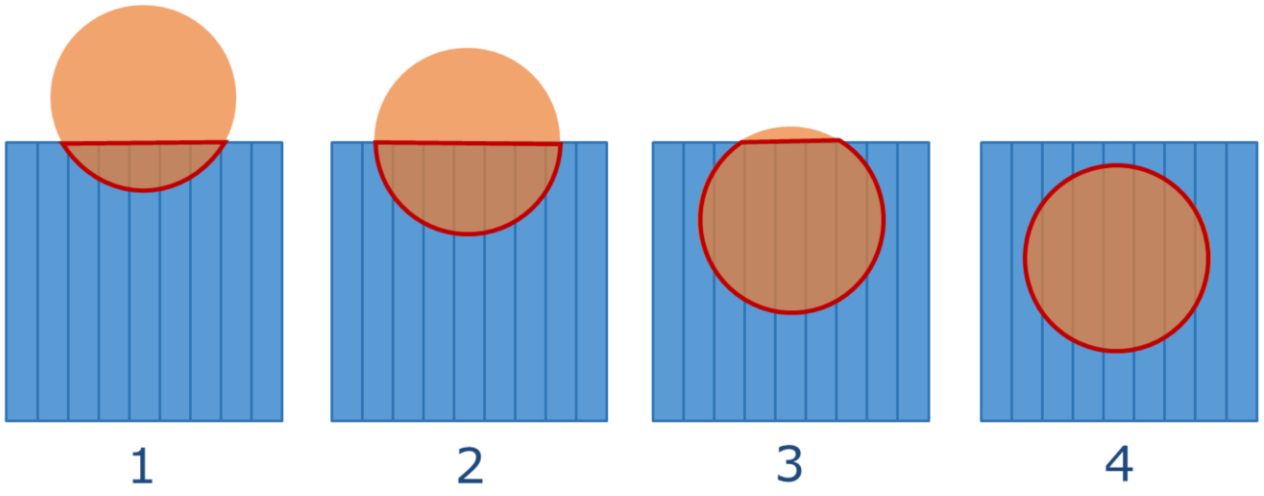
We update water height according to finite differences of the velocity. You can see the formation of waves.

Output



You can push fluid down under the collider only at output time, but you don’t propagate this back to the next solver step.

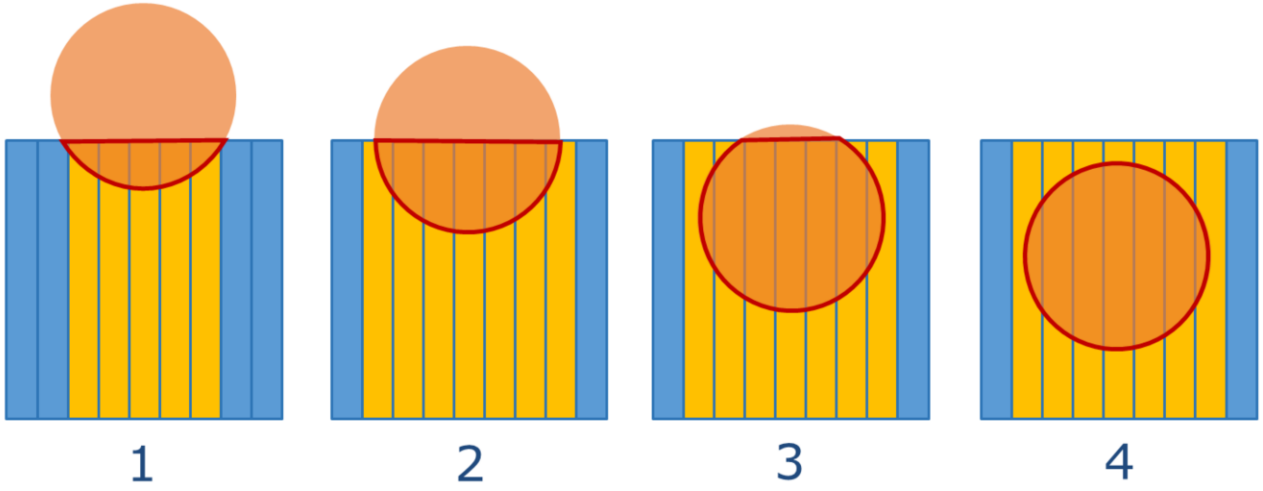
Overlap Volume Problem



Here are 4 examples of a sphere overlapping the water plane at different depths.

Note that from 1 to 4 the overlap volume increases.

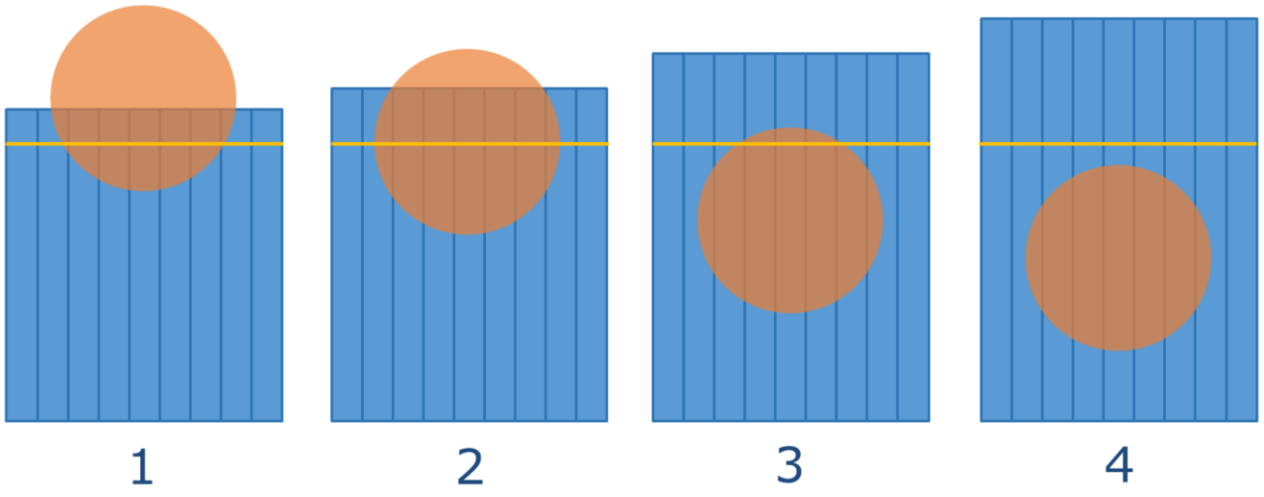
Incorrect Collision Mask



There are a few problems with calculating overlap volume in the fashion previously described.

Yellow show the cells the solver considers as being in a collision state. Note that qualitatively these don't represent what you see intersecting the water surface, especially in #4 where the ball is completely underwater.

Too Much Displaced Volume



There are also issues with what happens when we redistribute the overlap volume.

Note that as the sphere gets buried deeper in the water, more fluid is displaced, which is intuitively wrong.

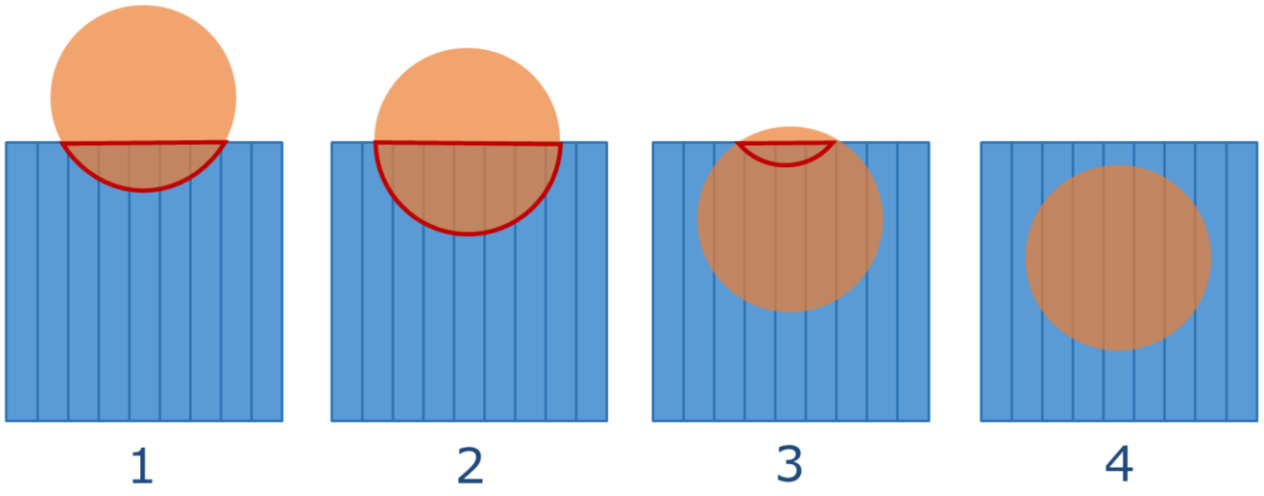
Imagine swimming near the surface of the water, kicking hard. You get a lot of disturbance of the water

Now imagine diving deep under the water and kicking just as hard. You don't see nearly as much disturbance, but with this model, you'd see more.

The goal is to come up with a collision model that qualitatively does what we'd expect while maintaining the simple simulator framework.

This video shows using the incorrect collision model for this sub falling in the water with the red proxy sphere used for collisions. Note that in this case, as the sub descended fully underwater, you notice cells at the surface propagating waves that are too large, and artifacts for the cells that are considered to be in a collision state.

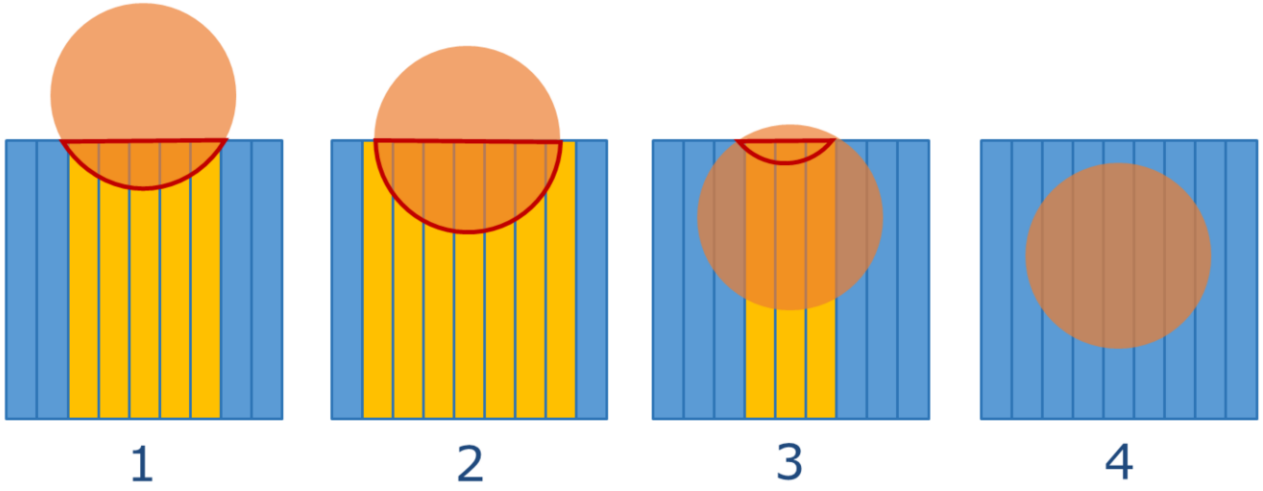
Overlap Volume Fix



To fix this problem, when the sphere is more than half underwater, we take the overlap volume equal to the portion above water.

Look at 3 where the overlap volume is mirror of what is above surface.

Correct Collision Mask

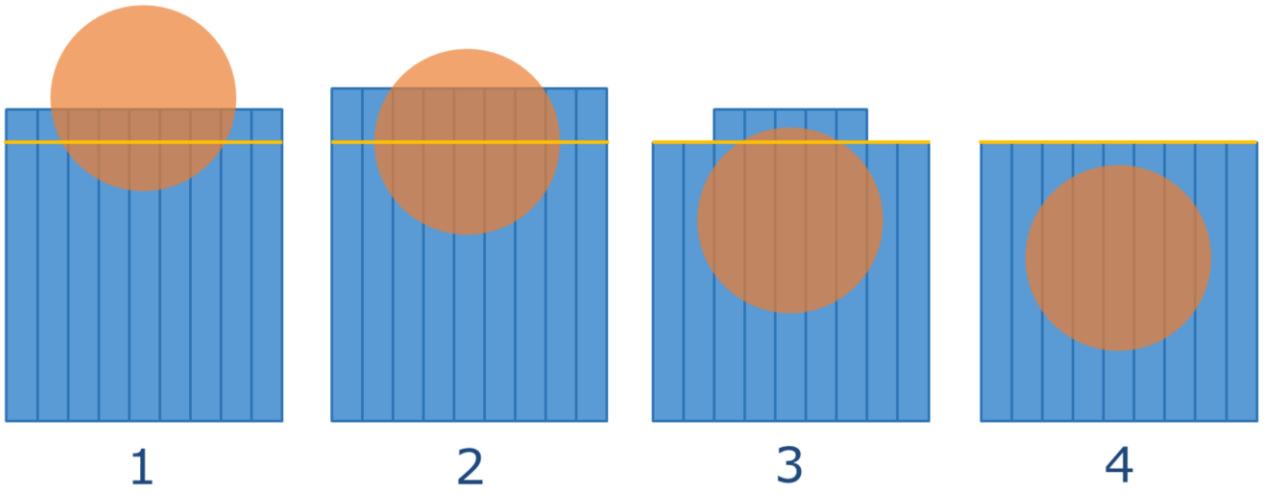


The collision mask represents what is seen above water (look at 3).

4 has no collision since it is fully submerged.

This gives roughly what we'd expect visually.

Correct Displacement

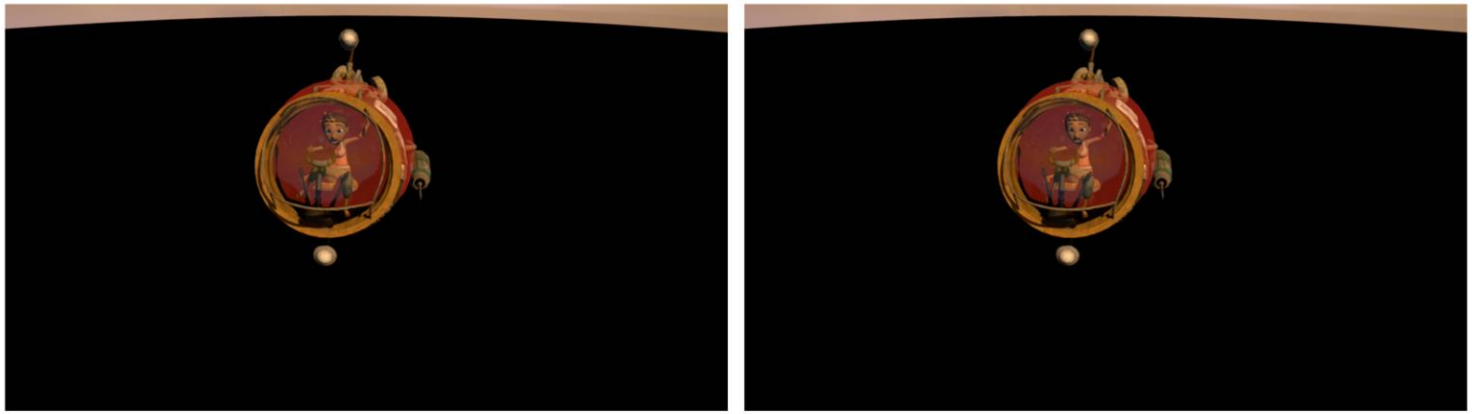


Also, this method attenuates the distribution of overlap volume to zero when the sphere is completely underwater.

Volume displacement attenuates to zero as it gets deeper, until there is no displacement in #4. Note that the overlap volume peaks at 2.

This video demonstrates the same sub falling into the water, but with the improved collision model. There are smaller surface waves as the sub fully submerges, and you don’t notice residual collision artifacts once the sub is fully submerged.

Comparison

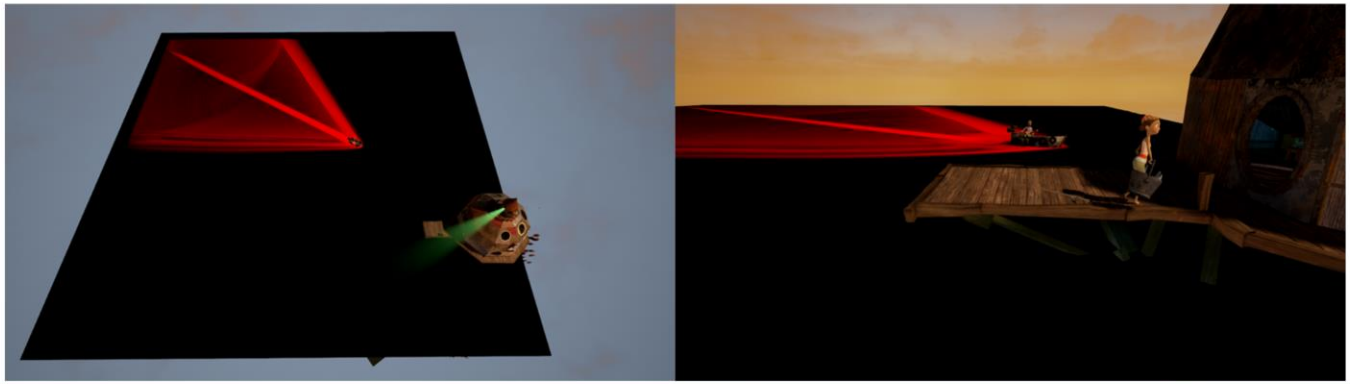


This compares the two videos side by side.

Real-time Considerations

Must use 32 bit buffers

Constraints on domain size



There are important things to consider when creating a simulator such as this in a real-time environment.

Anything less than 32 bits per channel for height and velocity will cause instability, and the resulting simulation will explode.

In the image on the left, the boat is causing a wake, with water displacement height in red, across a large area. The image on the right shows where the viewer is watching a fish splash in the water.

While this could be solved with multiple domains, unfortunately, we could only afford one due to performance considerations. Also, we could only afford 512x512 for our domain size. Thus, it is hard to get enough resolution to correctly resolve all simulated elements.

This was a video of the fish falling in the water from the previous slide. Note that the splash looked low resolution.

We bake out a Houdini FLIP simulation and composite it in with the water surface. The result looks much better.

Comparison

This is a side by side comparison video of the two splashes.

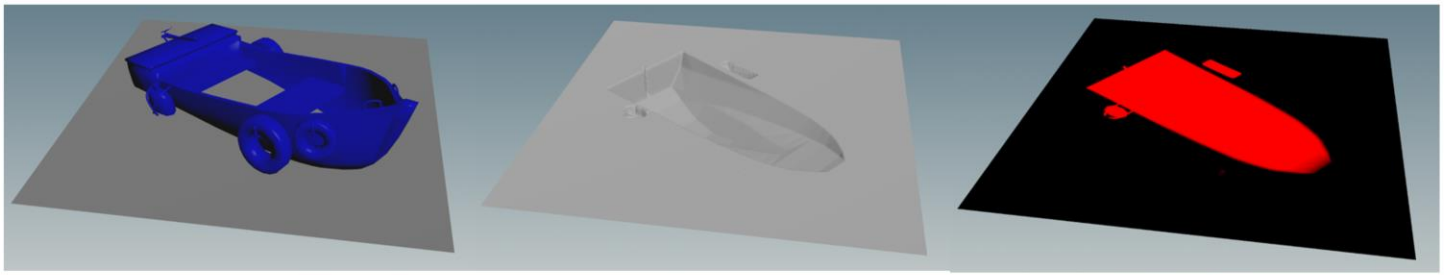
Static Depth Map Colliders

Bake out static collision mask

Parent to moving geometry

Sample to collision grid

Great for boat wakes



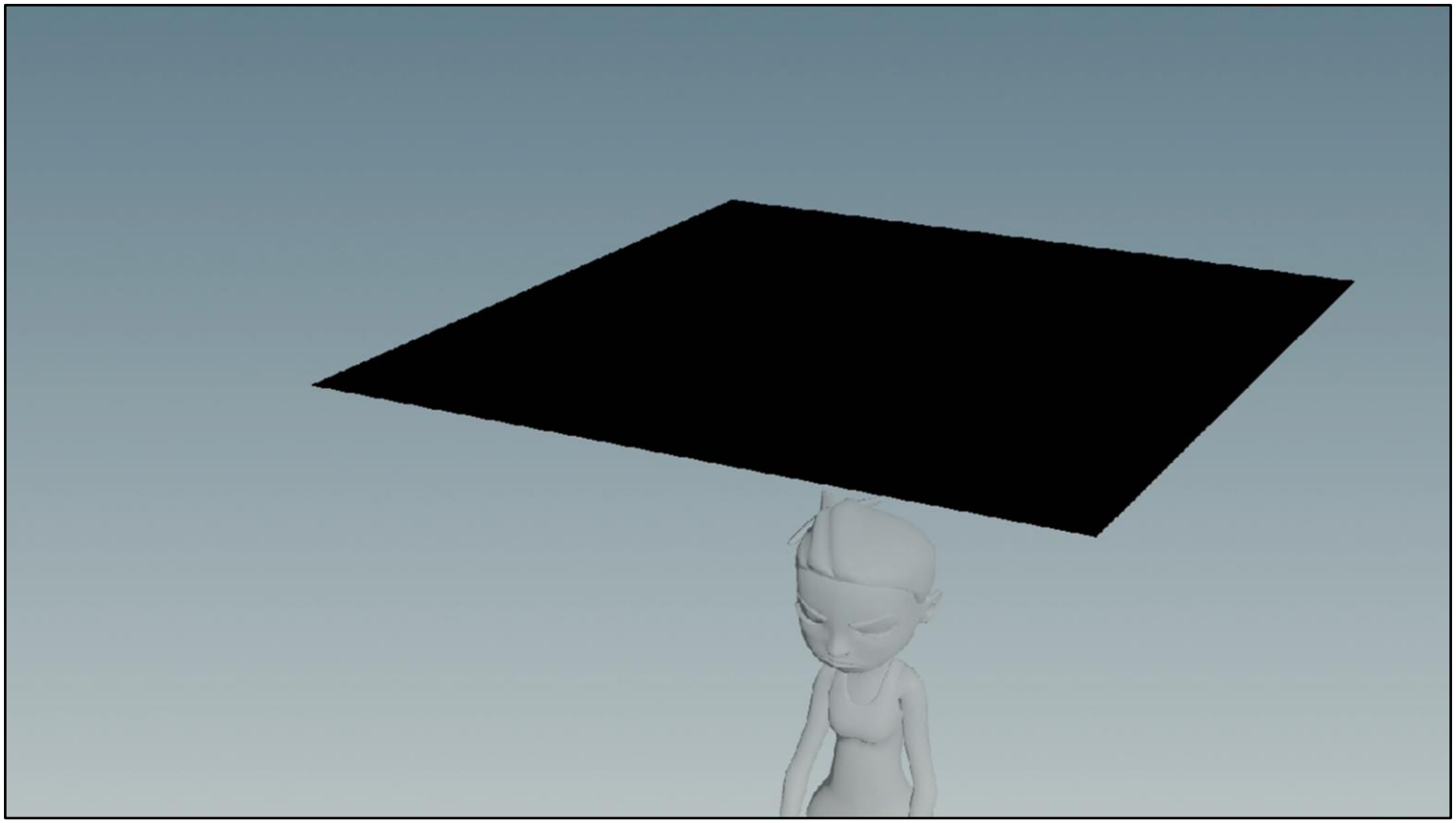
For a boat wake we want to precompute a collision mask for the boat.

First, we determine how much the boat falls below the base waterline.

Next, we bake out this data into a 2d image.

Finally, we parent a plane with this image to the moving boat, and resample the collision depth to the simulation domain.

This is a video of a boat wake and foam simulation.

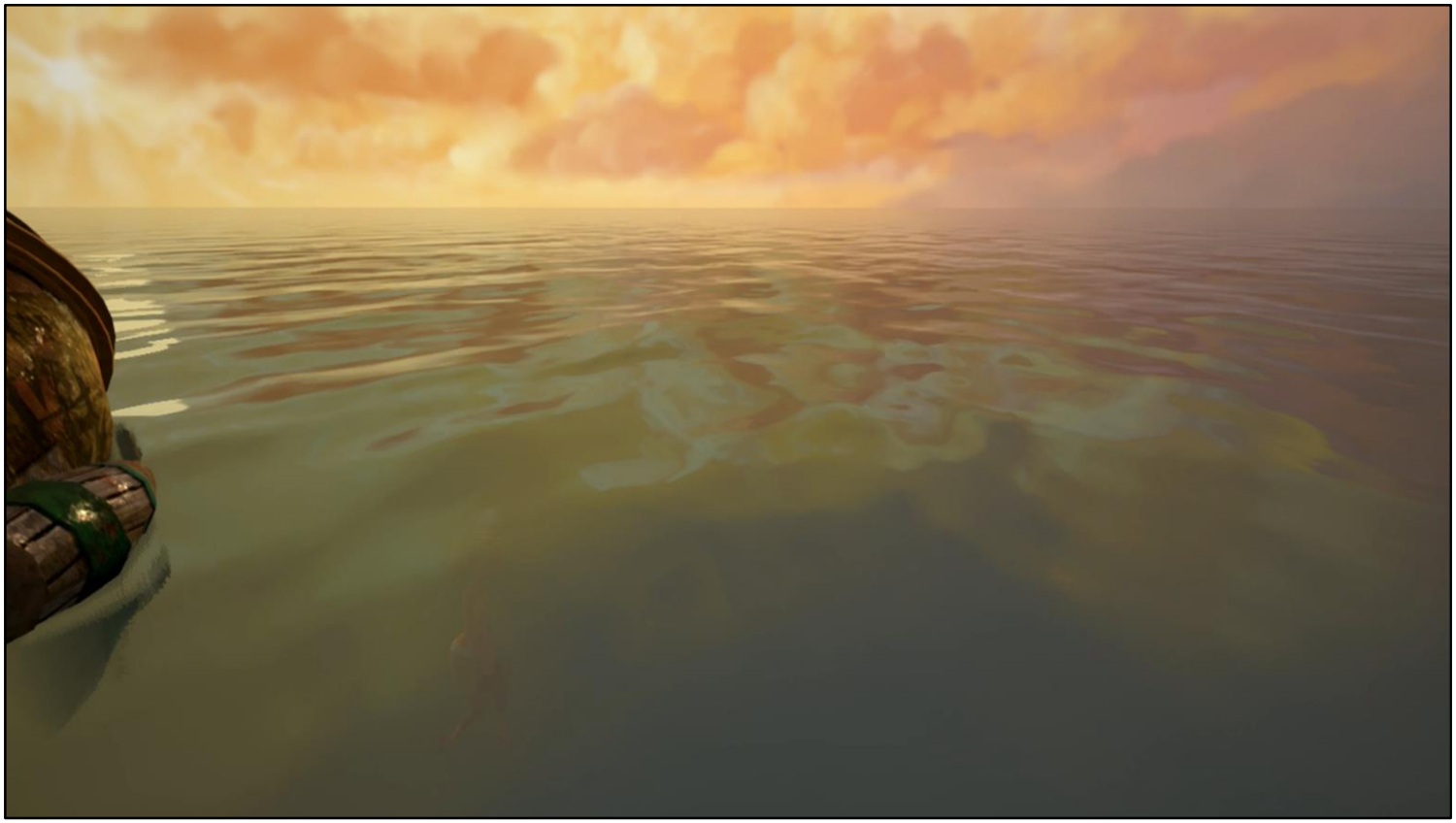


Moving colliders can be baked out into a texture atlas.

This is an example of a character swimming in water, and we compute the penetration depth per frame of her in relation to the water plane.

This is much better than baking out the sim since we only care about the small patch of water where the character interacts.

Similar to the case with the sphere, we only care when the arms are near the water surface, not necessarily intersecting it directly.



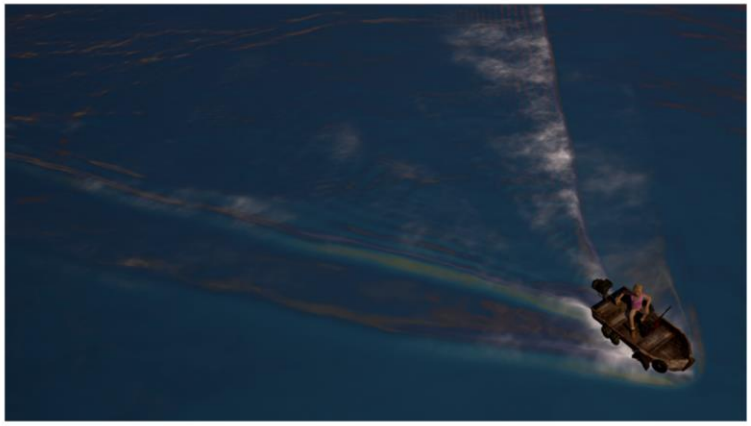
This video shows the simulation of the character swimming in the water.

Whitewater

Emit based on fluid acceleration [Ihmsen 12]

Advect by velocity

Dissipation



We used a simple whitewater model inspired by Ihmsen's paper on a unified solver for spray, foam and air bubbles, which describes techniques used extensively in visual effects.

The basic idea is that we track a scalar field for whitewater, and emit into it according to fluid acceleration. We then advect the whitewater by the fluid velocity, and apply dissipation.

Results

512x512 for height and velocity

1024x1024 for foam

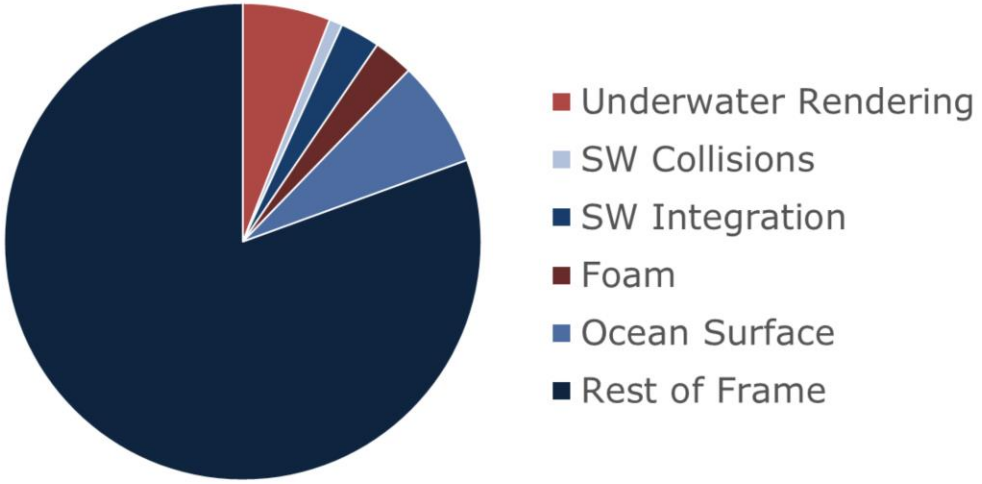
0.6 ms per frame

- 0.1 ms for colliders
- 0.3 ms for integration
- 0.2 ms for foam

We used a 512x512 resolution simulation grid for the shallow water elements.

We used a higher resolution foam simulation field since it is cheaper to compute than the water simulation, and it gives nice detail to the overall look.

Frame Time



Overall, we use about a quarter of our total frame budget in the worst case for the elements discussed in this talk. Note that we'd also have overhead due to rendering the water surface, and other elements have variable performance, so these numbers are approximate.

This video demonstrates how a user could interact with the fluid if given a VR hand controller. Since the water simulation runs in real time, it is very responsible to user input, and gives a terrific sense of immersion in the world.

Citations

[Kulla 11] Importance Sampling of Area Lights in Participating Media, SIGGRAPH 2011

[Hoobler 16] Fast, Flexible, Physically-Based. Volumetric Light Scattering, NVIDIA Developer Technology 2016.

[Hillaire 16] Physically-based & Unified Volumetric Rendering in Frostbite, SIGGRAPH 2015.

[Tessendorf 01] Simulating Ocean Water, SIGGRAPH 2001.

[Gjoel 16] Low Complexity, High Fidelity - INSIDE Rendering, GDC 2016.

[Bridson 16] Fluid Simulation for Computer Graphics, 2016.

[Ihmsen 12] Unified spray, foam and air bubbles for particle-based fluids, The Visual Computer, 2012.

These are the citations used in this paper.

Citations

[Kulla 11] Importance Sampling of Area Lights in Participating Media, SIGGRAPH 2011

[Hoobler 16] Fast, Flexible, Physically-Based. Volumetric Light Scattering, NVIDIA Developer Technology 2016.

[Hillaire 16] Physically-based & Unified Volumetric Rendering in Frostbite, SIGGRAPH 2015.

[Tessendorf 01] Simulating Ocean Water, SIGGRAPH 2001.

[Gjoel 16] Low Complexity, High Fidelity - INSIDE Rendering, GDC 2016.

[Bridson 16] Fluid Simulation for Computer Graphics, 2016.

[Ihmsen 12] Unified spray, foam and air bubbles for particle-based fluids, The Visual Computer, 2012.

Thanks!

I want to give a special thanks to David Hill of Epic Games for helpful discussions about fluid dynamics. Also, thanks to the entire team at Penrose Studios for the amazing experience collaborating on such an innovative project.

Thanks everyone!