Final Project: Water

Hsin-Yuan Huang

B03902124

I am always fascinated by the fundamental elements constituting our world, *i.e.* fire, water, earth and air. The complexity evolved from their simplicity is always an interesting phenomenon. In this project, I considered the simulation and rendering of water. In particular, I use smoothed particle hydrodynamics (SPH) for water simulation. It can be easily extended to the simulation of air by using different motion parameter, but it would take completely different scheme for realistic rending, so only water is considered.

This report is ordered as follows: Part 1 will cover the dynamics of water particles, Part 2 will cover the reconstruction of water surface, and Part 3 will discuss my rendering scheme to create realistic images (videos).

1 DYNAMICS OF WATER UNDER SPH

The most important equation governing the dynamics of water is the well-known Navier-Stokes equation,

$$\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v} = \frac{1}{\rho}(-\nabla p + \mu \nabla^2 \mathbf{v} + \mathbf{f}_{\text{ext}}),$$

where $\mathbf{v} = \mathbf{v}(x, y, z, t)$ is the velocity field for a fixed point in space-time. For grid-based method, the velocity field is updated according to this equation. But for particle-based method, this equation has to be modified (or recovered to it's original form). For particle-based method, thousands of particles move along the velocity stream of the fluid. Thus for a particular particle $\mathbf{v}_i = \mathbf{v}_i(x_i(t), y_i(t), z_i(t); t)$, a function of time. And the total time derivative of $\mathbf{v}_i$ is

$$\frac{d\mathbf{v}_i}{dt} = \frac{\partial \mathbf{v}_i}{\partial x}\dot{x}_i + \frac{\partial \mathbf{v}_i}{\partial y}\dot{y}_i + \frac{\partial \mathbf{v}_i}{\partial z}\dot{z}_i + \frac{\partial \mathbf{v}_i}{\partial t} = \frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v},$$

which is exactly the LHS of Navier-Stokes equation. Thus we can simply calculate the acceleration, $\mathbf{a}_i$, for each particles via the Navier-Stokes equation.

Then for each time step, I would assign $\mathbf{r}_i := \mathbf{r}_i + \mathbf{v}_i * \Delta t$ and $\mathbf{v}_i := \mathbf{v}_i + \mathbf{a}_i * \Delta t$. I used fixed Δt for simplicity. However the tuning of Δt is important, as small value results in long simulation, and large value result in unstable behavior. Also under different variations for dynamics simulation, which will be discussed in the following content, Δt can differ significantly.

1.1 Smoothed Particle Hydrodynamics

This technique is initially developed for simulation of astrophysical problems, but can also be used for fluid simulation. The basic idea for smoothed particle hydrodynamics (SPH) is to interpolate any discrete quantity for particle system. For example, the density for a particle-based system would consists of several Dirac delta functions, but we want a smoothed version of the density field. According to SPH, a scalar quantity A is interpolated as

$$A(\mathbf{r}) = \sum_j m_j \frac{A_j}{\rho_j} W(\mathbf{r} - \mathbf{r}_j, h),$$

where W is a smoothing kernel with finite support, with radius of support equals to h. Also W is normalized such that

$$\int W(\mathbf{r}) d\mathbf{r} = 1.$$

In the previous example on smoothing ρ, we simply plug it into A, and we would get

$$\rho(\mathbf{r}) = \sum_j m_j \frac{\rho_j}{\rho_j} W(\mathbf{r} - \mathbf{r}_j, h) = \sum_j m_j W(\mathbf{r} - \mathbf{r}_j, h).$$

The gradient and Laplacian for these smoothed quantity is simply based on the gradient and laplacian of the smoothing kernel, *i.e.*

$$\nabla A(\mathbf{r}) = \sum_j m_j \frac{A_j}{\rho_j} \nabla W(\mathbf{r} - \mathbf{r}_j, h) \text{ and } \nabla^2 A(\mathbf{r}) = \sum_j m_j \frac{A_j}{\rho_j} \nabla^2 W(\mathbf{r} - \mathbf{r}_j, h).$$

The choice for smoothing kernel follows the implementation of [2, 3]. A poly6 kernel is used for all cases except the calculation of pressure (∇p), which uses Debrun's spike kernel, and viscosity ($\nabla^2 v$), which uses their hand-made kernel.

I have used a simple acceleration structure for SPH calculation. Since the smoothing kernel has a finite support, we only need to iterate through a small amount of neighbors rather than all the particles when calculating the smoothed quantity. To find the neighbors, the space is dissected into fixed grids. Each grid cell with size $h \times h \times h$. When computing the smoothed value for some $\mathbf{r}$, only the particles in the nine neighboring grid cell (including the one containing $\mathbf{r}$) are considered. This simple structure can gain a huge speed-up.

1.2 Viscosity

After discussing the concept of SPH. We will next consider the calculation for the RHS of the Navier-Stokes equation.

We will first focus on viscosity. The idea of viscosity is that when nearby particles are all moving in a particular direction, the left behind particle (or particles moving in the opposite direction) will experience a force that drag them towards the common direction. This is precisely the idea of the Laplacian term in the Navier-Stokes equation,

$$\mathbf{f}_i^{\text{visc}} = \mu\nabla^2\mathbf{v} = \mu\sum_j m_j\frac{\mathbf{v}_j}{\rho_j}\nabla^2 W(\mathbf{r}_i - \mathbf{r}_j, h).$$

But applying SPH will actually yields an asymmetric forces, thus [2] suggests the following modification,

$$\mathbf{f}_i^{\text{visc}} = \mu\sum_j m_j\frac{\mathbf{v}_j - \mathbf{v}_i}{\rho_j}\nabla^2 W(\mathbf{r}_i - \mathbf{r}_j, h).$$

This with their hand-made kernel function complete the calculation of viscosity.

1.3 Surface Tension

Surface tension is also an important characteristic for water. Nevertheless a careful look at Navier-Stokes equation will find nowhere for this term, since it is actually hidden inside $\mathbf{f}_{\text{ext}}$. The cause of surface tension is due to the self-attraction between particles. But since the net force is 0 for particles in the interior of the fluid, only particles on the boundary will experience force. In [2], they come up with a method by interpolating color value (which is simply 1 for every particles) using SPH. And determine the surface tension from the color value. This is the initial method I adopted. But in a later paper [4], they criticized that this method yields an incorrect result and should be redefined as follows,

$$\mathbf{f}_i^{\text{tens}} = -\kappa\rho_i\sum_j\frac{m_j}{m_i}(\mathbf{r}_i - \mathbf{r}_j)W(\mathbf{r}_i - \mathbf{r}_j, h).$$

Their idea is to directly model the attractive force between molecules, thus I decided to follow their definition in my implementation.

1.4 Pressure

I think this is the most important part in fluid simulation, since both μ and κ can be set to 0 without effecting the general motion. But without pressure, every particle will drop to $z = 0$ due to gravity and become a thin film no matter how many particles are used. Thus a crucial question arises: "what is pressure anyway?" A peek to the equation of state (EoS) for ideal gas may give us some intuition, *i.e.*

$$p = \frac{n}{V}RT = k\rho.$$

From the equation, we can easily see that the larger the density is, the larger the pressure will be. Thus each particle will experience a force pushing it towards region with lower density (and the thin film problem will not occur).

In [2], they directly used the EoS for ideal gas to define pressure. And they stated that, if the definition is modified into

$$p = k(\rho - \rho_0),$$

where ρ_0 is the normal density, the simulation will be numerically more stable (this does not affect the original equation of motion since $\nabla(p+c) = \nabla p$, but will affect the motion if SPH is used). Also, the resulting force $(-\nabla p)$ will not be symmetric under SPH, thus I apply the following modification (there are several kinds of modification, this one is proposed in [2])

$$\mathbf{f}_i^{\text{pres}} = -\sum_j m_j \frac{p_i + p_j}{2\rho_j} \nabla W(\mathbf{r}_i - \mathbf{r}_j, h).$$

There is actually a loop hole in the above statement. If the variation of pressure depends on density, it means that the water simulated will be compressible. This may sound absurd, but is indeed true in reality. Since the constant k is very large for water (and most liquid), a small change of density will result in a large restoring force. Thus we may feel that water is incompressible. However, in the simulation, we can not use a large k, since we would then need an extremely small Δt to have a stable result (due to Courant-Friedrichs-Lewy condition), which is computationally impossible. The compressibility issue can be seen in some older water simulation via SPH such as `https://www.youtube.com/watch?v=baoAQaYxQcE`. In 2007, Becker et al. [4] introduced a simple modification that would yield a nearly incompressible fluid with feasible Δt. Rather than using ideal gas EoS, they use Tait's EoS,

$$p = B((\rho/\rho_0)^7 - 1).$$

Because Δt is much smaller than ideal gas EoS (at least 40 times smaller), I will use ideal gas for parameter tuning and prototyping, only the final result uses the Tait's EoS.

1.5 Boundary Condition

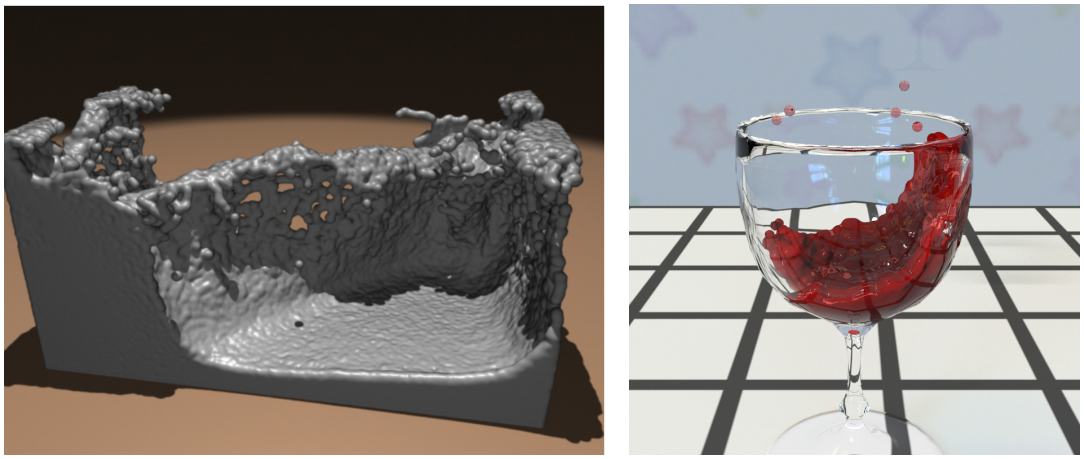
What should we do when a particle is about to pierce through some surface (such as boundary wall or objects)? This is also an important issue with many interesting solutions. A naive solution suggested by [2] is to reflect the velocity vector when it is about to pierce through a surface with normal $\hat{n}$,

$$\mathbf{v} := \mathbf{v} - 2(\mathbf{v} \cdot \hat{n})\hat{n}.$$

The result for my implementation to this method is shown in **water_elastic.avi**. I used compressible fluid and MATLAB for fast real-time rendering (it is actually a 3D simulation, but I put the camera on the side for better visibility). As can be seen from the video, the water is very bouncy and unstable. One reason for this is due to the compressibility, but another more important reason is because elastic collision preserves energy, so the gravitational energy are not released. Therefore I decided to make the collision less elastic, the result is shown in **water_inelastic.avi**. Now, the water is much less bouncy. But some very weird behavior appears on the side, there seems to be small fountains emerging. Thus some other methods have to be used in order to get a realistic motion simulation.

Inspired by the ghost particle method (one variation can be seen in [4]), I constructed a damped repulsive force acting on the particle when it is close to the boundary surface. If the distance to the surface d is smaller than a predefined particle radius r, the particle will experience a force

$$\mathbf{f}_i^{\text{bound}} = \rho_i(\alpha(r-d) - \beta(\mathbf{v}_i \cdot \hat{n}))\hat{n}.$$



(a) Picture from [5]

(b) Picture from [7]

Figure 2.1: Pictures for surface reconstruction using isotropic kernel method, the surface is very bumpy due to the particle nature of particle-based methods.

The α term models the repulsive force from the surface, and the β term models the damping force that lowers the kinetic energy of the particle. Using repulsive force yields a more natural dynamics, but it means the particle is possible to pierce through the surface. To avoid such an event, I would clamp the acceleration caused by other forces to some predefined value. The result is shown in **water_compressible.avi**, which is much better than the previous two. The dynamics for water particles is now complete. However the tuning of all the parameters, e.g. $\alpha, \beta, B, \mu, \kappa, h$ ... etc., is still a heavy work, since they are not clearly written in the papers. It takes days to yield a visibly realistic water dynamics. The codes for SPH are in **SPHfluid.cpp**, which uses ideal gas EoS, and **WCSPHfluid.cpp**, which uses Tait's EoS. A short MATLAB code for rendering is in **drawdots.m**. We will then enter Section 2 on water surface reconstruction.

2 Surface Reconstruction via Anisotropic Kernel

There are several methods for visualizing water surface. I choose the method that reconstruct an iso-surface for some predefined color value. Recall that color value is defined to be one for each particle, so we have

$$c(\mathbf{r}) = \sum_j \frac{m_j}{\rho_j} W(\mathbf{r} - \mathbf{r}_j, h),$$

by the SPH framework. This can be done by using the popular marching cubes method [1]. It constructs a triangle mesh by dissecting the space into grids, and determine the triangles within each grid cell from the color values on the eight vertices. To avoid holes in the original method proposed in [1], all 256 combinations are configured. The table for the triangles under each combinations is referenced from [6], which also gave a nice introduction to marching cubes.

Now the problem turns into finding the color values on the vertices. A naive approach

is to use the above equation directly. This is a feasible and natural solution, but the resultant water surface is not very convincing. Owing to the particle nature of SPH method, the surface can be very bumpy even with large amount of particles. In Fig. 2.1, a few examples are shown. Fig. 2.1a comes from [5], it uses 140K particles, a very large number, but the resulting surface is still very bumpy and doesn't look much like liquid. Fig. 2.1b earned the "honorable mention" award in Stanford's Rendering Competition 2009. However, the surface is clearly constituted by a number of particles.

Therefore, I adopted the method proposed in [5]. They come up with a novel approach that uses anisotropic kernel for color value calculation, with the kernel stretched according to the local principal component. The resulting surface is smooth and much more realistic.

2.1 Determine Anisotropic Kernel via WPCA

The idea of their novel method is that each particle will now use a different smoothing kernel and have a different position depending on the local particle distribution. However, this only happens when we are reconstructing the water surface, and does not effect the simulation of water dynamics. The new position for particle i is defined to be the weighted average (with $\mathbf{r}_i$ as the center) in weighted principal component analysis (WPCA), *i.e.*

$$\bar{\mathbf{r}}_i = \frac{1}{\sum_j w(\mathbf{r}_i, \mathbf{r}_j)} \sum_j w(\mathbf{r}_i, \mathbf{r}_j)\mathbf{r}_j,$$

where w is a weighting function with finite support = $2h$. Since the weighting function has a finite support, we can apply the grid acceleration structure to speed-up the calculation. The definition for the new smoothing kernel is much more complicated, with many details involved. A simplified procedure is to first compute the singular value decomposition (SVD) of the weighted covariance matrix in WPCA, *i.e.*

$$R\Sigma R^T = \frac{1}{\sum_j w(\mathbf{r}_i, \mathbf{r}_j)} \sum_j w(\mathbf{r}_i, \mathbf{r}_j)(\mathbf{r}_j - \bar{\mathbf{r}}_i)(\mathbf{r}_j - \bar{\mathbf{r}}_i)^T.$$

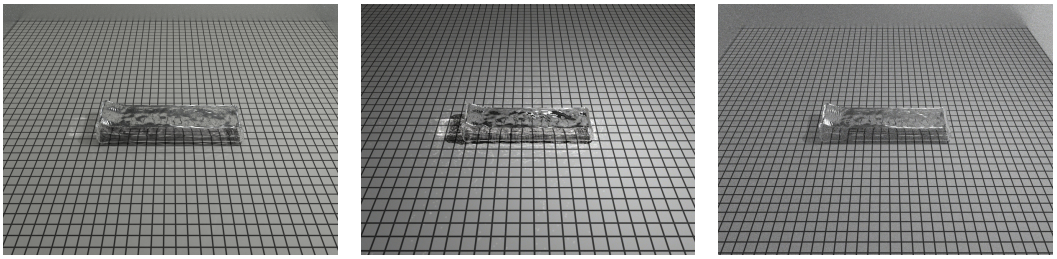
Note that R is an orthogonal matrix and the column vectors of R are the principal axes. I used the code from [8] to calculate the SVD of a 3×3 symmetric matrix (**eig3.h** and **eig3.cpp**) in my implementation. Then the smoothing kernel is defined (simplified version) as

$$W_j(\mathbf{r}) = \det(\Sigma^{-1}) W(\Sigma^{-1} R^T \mathbf{r}, 1).$$

The determinant helps preserving the property that W_j integrates to 1. The modified color value is then defined as

$$\bar{c}(\mathbf{r}) = \sum_j \frac{m_j}{\rho_j} W_j(\mathbf{r} - \bar{\mathbf{r}}_j).$$

To gain an intuition on this modification, we focus on the support of the kernel. Originally, the support for $W(\cdot, h)$ is a sphere with radius = h. In this modified kernel, the support becomes an ellipsoid with three principal axes equal to the column vectors of R and the length of each axes equals to the corresponding singular values. For particles in the interior of the water, $\Sigma \approx \mathrm{diag}(h, h, h)$, thus it is roughly the same as the original kernel. For particles on the



(a) Background: aluminum, Material: glass, uses photon mapping

(b) Background: void, Material: glass, uses photon mapping

(c) Background: aluminum, Material: glass, uses bidirectional path tracing

Figure 3.1: A few different settings for rendering the triangle mesh

boundary of the water, the ellipsoid is flatten in the normal direction of the water surface, thus the surface will become less bumpy and more realistic.

To speed-up the calculation of color value on the marching cubes' grid vertices, I calculated a tight axis-aligned bounding box (AABB) for an arbitrary ellipsoid. And for each ellipsoid, I update only the color value of the vertices inside the AABB. I also used a hash map to store the encountered grid cells, since many of them are sure to contain no triangles. The complete code for surface reconstruction is in **particle2pbrt.cpp**.

3 Rendering Scheme for Water Simulation

The last thing to do is to set up the scene, the rendering strategy and the material for the triangle mesh to make it look like realistic water. I have tried out many different settings, some results are shown in Fig. 3.1. I think the most efficient and satisfying surface integrator is photon mapping. And I choose to enclose the whole thing with an aluminum box (for matte box, the reflection on the surface of water looks very unnatural) to avoid reflecting into emptiness. The refraction index for glass is set to the refraction index of water. For better visibility on water movement, I decided not to render the container that contains the water. I used luxrender for rendering since it is faster than pbrt. To produce a video, I would need to render hundreds of images, so rendering time will be an important issue.

The final videos are **water_comp.mp4** (compressible water) and **water_incomp.mp4** (nearly incompressible water). The prototype scene files are in **render_comp.lxs** and **render_incomp.lxs**, respectively. It has to be slightly modified to render images at different time steps. It takes about ten minutes to create each image, and about 1.5 days to produce a video. For both videos, due to the high computational cost, only about 9K particles are used. Nevertheless the water surface is still very smooth. Some caustics can also be seen under the water, as photon mapping is used. A final note is that the water simulation under SPH, although being visibly correct, is physically incorrect, since a lot of modification is made to the equation of motion as shown in Section 1.

REFERENCES

[1] Lorensen, William E. and Cline, Harvey E. MARCHING CUBES: A HIGH RESOLUTION 3D SURFACE CONSTRUCTION ALGORITHM. In SIGGRAPH '87: Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques, pp. 163~169.

[2] Matthias Müller, David Charypar, and Markus Gross. PARTICLE-BASED FLUID SIMULATION FOR INTERACTIVE APPLICATIONS. In SCA'03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation, pp. 154~159.

[3] M. Kelager. (2006) LAGRANGIAN FLUID DYNAMICS USING SMOOTHED PARTICLE HYDRODYNAMICS (MS Thesis, Univ. Copenhagen).

[4] Becker, Markus and Teschner, Matthias. WEAKLY COMPRESSIBLE SPH FOR FREE SURFACE FLOWS. In SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, pp. 209~217.

[5] Yu, Jihun and Turk, Greg. RECONSTRUCTING SURFACES OF PARTICLE-BASED FLUIDS USING ANISOTROPIC KERNELS. In SCA '10: Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, pp. 217~225.

[6] Paul Bourke. POLYGONISING A SCALAR FIELD. Website: `http://paulbourke.net/geometry/polygonise/`.

[7] Soowoung Ryu. POURING WINE. In Stanford Rendering Competition '09. Website: `https://graphics.stanford.edu/wikis/cs348b-09/bshboy/FinalProjectWriteUp`.

[8] Connelly Barnes. EIGENVECTORS OF 3X3 SYMMETRIC MATRIX. Website: `http://barnesc.blogspot.tw/2007/02/eigenvectors-of-3x3-symmetric-matrix.html`