CS 488 Project Report

Water Physics with Rigid Bodies

Formerly, Bottles & Water

Palaksha Drolia, Awab Qureshi

pdrolia, a9quresh

1 Final Report

1.1 Rasterisation and Rendering pipeline

We used OpenGL for rasterisation to reduce CPU load and utilise GPU more effectively. Drawing triangles to the screen was done using the standard process described in [1]. We integrated the transformation matrix and our shade function into the OpenGL pipeline to form our vertex and fragment shader, respectively. This was written in GLSL and utilised uniforms passed by our program for effective rendering. We also created a Phong-based shader that accounts for ambient, diffuse, and specular contributions to give a more realistic output. We also created a high-level object-oriented wrapper over certain OpenGL functionality, like shaders, to easily set and verify uniforms.

1.2 Shallow-Water Simulation

The main focus of our project is the physics-based simulation of a shallow, large body of water: the pool. In order to maintain a level of interactability, we use a two-dimensional height field for simulation.

For this, we employ the shallow water equations as given in [2],

$$\frac{Dh}{Dt} = -h(\nabla \cdot v) \qquad \frac{Dv}{Dt} = -g\nabla\eta + a^{ext}$$

where h is the depth of the water, H is the y-coordinate of the terrain on the bottom, $\eta = H + h$ is the y-coordinate of the water's surface, v is the vector (u, w) representing the horizontal velocity of the fluid, g is gravity, a^{ext} is the external acceleration, and D is the material derivative operator as given in the course slides [3].

For better accuracy, we implement a staggered grid for our discretised simulation as specified briefly in the course slides [3] and [2]. We store the heights of a cell, $h_{i,j}$ and $H_{i,j}$ at its centre. And we store the velocity components $u_{i+\frac{1}{2},j}, w_{i,j+\frac{1}{2}}$ on the faces. When computing values not stored, we bilinearly interpolate.

The grid is first updated with values generated by computing the advection of height and velocity. We use the semi-lagrangian method as proposed in [3] in order to solve the advection of $h_{i,j}$, $u_{i+\frac{1}{2},j}$ and $v_{i,j+\frac{1}{2}}$. Let $x_u = ((i+\frac{1}{2})\Delta x, j\Delta x)$ be the position of the grid cell at $u_{i+\frac{1}{2},j}$ and similarly define x_v and x_h We thus compute the new values, $u^{n+1}_{i+\frac{1}{2},j} = interp(x_u - \Delta t \cdot (u_{i+\frac{1}{2},j}, w_{i,j})^T)$, and $w^{n+1}_{i,j+\frac{1}{2}} = interp(x_v - \Delta t \cdot (u_{i,j}, w_{i,j+\frac{1}{2}})^T)$, and the height as $h^{n+1}_{i,j} = x_h - \Delta t \cdot (i\Delta x, j\Delta x)$.

The heights are integrated by adding the following,

$$h_{i,j} = -\left((\overline{h}_{i+\frac{1}{2},j}u_{i+\frac{1}{2},j} - \overline{h}_{i-\frac{1}{2},j}u_{i-\frac{1}{2},j})/(\Delta x) + (\overline{h}_{i,j+\frac{1}{2}}w_{i,j+\frac{1}{2}} - \overline{h}_{i,j-\frac{1}{2}}w_{i,j-\frac{1}{2}})/(\Delta x)\right)\Delta t$$

as suggested by [2], and [3]. Here we implement a heuristic provided by [2], where instead of linearly interpolating to find the values of $\overline{h}_{\_,\_}$, we instead evaluate it to be equal to h in the upwind direction.

The velocities are updated, as [2] suggests, taking the gradient of the water height. For our staggered

velocities, we add the following,

$$u_{i+\frac{1}{2},j} \mathrel{+}= \left(\frac{-g}{\Delta x}(\eta_{i+1,j} - \eta_{i,j}) + a_x^{ext}\right)\Delta t \qquad w_{i,j+\frac{1}{2}} \mathrel{+}= \left(\frac{-g}{\Delta x}(\eta_{i,j+1} - \eta_{i,j}) + a_z^{ext}\right)\Delta t$$

where a^{ext} is the external acceleration.

For boundary conditions, our pool has a well-defined boundary that reflects the waves. We carry out the method suggested in [3] and at the boundary set the heights to the same as their neighbours, and set the velocity component into the wall to be zero.

1.3 Rasterising Water Surface

We rasterise the water surface with the method described in [2]. The height field of our fluid is already a gird of quads, with each (i, j) vertex having height $\eta_{i,j}$ that we may split into triangles. We copy over the vertex data to the GPU and split the quads to emit triangles in a geometry shader. Normals for the water surface are computed on the CPU.

Additionally, as suggested by [2], we slice the quad into triangles across the following diagonal,

$$\begin{cases} (i,j) - (i+1,j+1) & \text{if } \eta_{i,j} + \eta_{i+1,j+1} > \eta_{i+1,j} + \eta_{i,j+1} \\ (i+1,j) - (i,j+1) & \text{else} \end{cases}$$

since picking the diagonal that aligns with the wave's crests reduces artefacts.

For transparency, we rely on OpenGL's blending. After enabling OpenGL's blending, we draw the opaque pool first and then draw the transparent water surface as suggested in [4].

1.4 Rigid-body Physics

For rigid-body physics, we employ Verlet integration as suggested by [5]. A force acts on the centre of mass of the rigid body whose next coordinate is computed via $x_{n+1} \approx x_n + (x_n - x_{n-1}) + \Delta t^2 F_n/m$ where x_n is the vector that represents the coordinates of the body, m is its mass, Δt is the time step, and F_n is the force acting on it. Additionally, we also apply gravity to the body.

For torque and rotation, we calculate the updated angular momentum as $L_{n+1} \approx L_n + \Delta t\tau$ where L_n is the angular momentum at time point n, Δt is the time step, and τ is the torque acting on it.

Calculating Moment of Inertia Next we need to find the Moment of Inertia for our rigid body to compute angular velocity and the new rotation. We achieve this by iterating through the triangular faces in our mesh. Note that we only need to calculate the initial inertia tensor as we can obtain the global inertia tensor at any point of time via $I = RI_0R^T$ where I_0 is the initial rotation tensor and R is the current rotation of the object as a rotation matrix. As suggested by [6], we need to compute 10 integrals in order to compute the volume, centre of mass, and initial inertia tensor. As mentioned in the paper, we can simplify these 10 integrals over the volume using the divergence theorem that states:

$$\int_V \nabla \cdot \mathbf{F}\, dV = \int_{\partial V} \mathbf{F} \cdot \hat{\mathbf{n}}\, dA$$

Using this, we appropriate $\mathbf{F}$ to convert the volume integrals into corresponding area integrals. Since they are area integrals, we can sum them over each individual face.

We solve these integrals using Barycentric parametrisation as suggested by [7] to get a closed-form result. Using these values, we can fill in the entries in the inertia tensor matrix.

Now, with the angular momentum and the current inertia tensor, we can calculate the angular velocity as $\omega = I^{-1}L$. We use a forward Euler scheme to update the rotation quaternion along the angular velocity's axis with the magnitude of $\|\omega\|\Delta t$.

For detecting collisions with objects other than the water's surface, we use a simple axis-aligned bounding box around the objects. We resolve collisions by finding the axis of minimum overlap, and correct based on that as suggested in [8]. We also calculate the force that would have acted on the objects and apply torque based on that. We also use this approach to ensure that objects stay inside the pool.

1.5 Rigid-body and Water Interaction

When a rigid body lands in the pool, we wish to modify the height and velocity of the fluid. To accomplish this, we utilise Algorithm 2 as given in [2]. We essentially subdivide the rigid body's triangles into smaller triangles until their area falls below Δx^2 where Δx is our grid spacing recursively. This subdivision allows fine-grained spatial resolution for capturing interactions accurately between the body and the fluid. For each of these small triangles, the centroid's position $p = (p_x, p_y, p_z)$ and its velocity $v = (v_x, v_y, v_z)$ is computed via barycentric interpolation. We use the triangle's normal n to determine the direction and magnitude of fluid displacement.

The velocity's magnitude relative to the vertical direction is then used to determine how many substeps to divide the current simulation timestep into,

$$num\_substeps = \max\{1, \lfloor |v - v_y y|(\Delta t/\Delta x) + 0.5 \rfloor\}$$

This is because more substeps allow smoother and more stable application of forces along the triangle's trajectory. For each substep, then, we advance the centroid position along its velocity, identify the fluid grid cell closest to this position, (i, j) and calculate the depth of the centroid relative to the fluid surface height. Upon finding the centroid submerged, we compute a decay factor that exponentially reduces influence with depth, reflecting that deeper submerged parts affect the fluid less. The fluid surface's height at the grid cell is then updated by adding a volume displacement proportional to the triangle's area, velocity, and decay factor,

$$h_{i,j} \mathrel{+}= e^{-(\eta_{i,j} - p_y)} \frac{(n \cdot v_{rel}) A \Delta t}{num\_substeps(\Delta x)^2}$$

where $v_{rel} = v - v_{fluid}$ and $depth = \eta_{i,j} - p_y$, $V_{disp} = n \cdot v_{rel}$. The fluid surface's velocity is then updated at the corresponding staggered grid faces by pushing them towards the triangle centroid's velocity, scaled by a coefficient,

$$coeff = \min\left\{1, \frac{e^{-(\eta_{i,j} - p_y)}}{5} \frac{(\eta_{i,j} - p_y)}{\eta_{i,j}} sign \frac{\Delta t}{(\Delta x)^2} A\right\}$$

where $sign$ is the sign of η_y.

The fluid must also interact with the rigid body. For this we consider buoyancy, drag and lift forces. We compute the sum of these forces $F_i = f_{buoyancy}, f_{drag}, f_{list}$ for each centroid of our subdivided triangle grid from before. We compute these forces by the very straightforward equations 14, 15, 16 and 17 as given in [2] which we do not copy here for brevity. The equations let us adjust coefficients C_D, C_L and ω for the drag, lift, and effective area respectively. Based on our research and experiments, we decided the values $C_D = 0.82$, $C_L = 0.007$, and $\omega = 0.9$. These are based on values for typical cylindrical objects.

The total force acting on the body at its centre of mass is given by $F = \sum F_i$. And each F_i produces a torque about the body's centre of mass, so the total torque generated is $\tau = \sum r_i \times F_i$, where r_i the displacement from p to the centre of mass, which is used to update the angular velocity of the body.

2 Bibliography

[1] J. de Vries, "Hello triangle," in *Learn OpenGL*. Kendall & Welling, 2020, ch. 5, pp. 26–41. [Online]. Available: https://learnopengl.com/book/book_pdf.pdf.

[2] N. Chentanez and M. Müller, "Real-time simulation of large bodies of water with small scale details," in *Proceedings of the 2010 Eurographics/ACM SIGGRAPH Symposium on Computer Animation, SCA 2010, Madrid, Spain, 2010*, Z. Popovic and M. A. Otaduy, Eds., Eurographics Association, 2010, pp. 197–206. DOI: 10.2312/SCA/SCA10/197-206. [Online]. Available: https://doi.org/10.2312/SCA/SCA10/197-206.

[3] "Lecture 13: Waves," University of Waterloo, CS 488/688, Spring 2025.

[4] J. de Vries, "Blending," in *Learn OpenGL*. Kendall & Welling, 2020, ch. 24, pp. 190–295. [Online]. Available: https://learnopengl.com/book/book_pdf.pdf.

[5] "Lecture 12: Particles," University of Waterloo, CS 488/688, Spring 2025.

[6] B. Mirtich, "Fast and accurate computation of polyhedral mass properties," *J. Graphics, GPU, & Game Tools*, vol. 1, pp. 31–50, 1996. [Online]. Available: https://api.semanticscholar.org/CorpusID:3445315.

[7] A. H. Barr, "Polyhedral mass properties (revisited)," Geometric Tools, Inc., Tech. Rep., 1992, Accessed: 2025-08-04. [Online]. Available: https://www.geometrictools.com/Documentation/PolyhedralMassProperties.pdf.

[8] Game Technology Group, "Physics - collision detection," Newcastle University, Newcastle upon Tyne, United Kingdom, Tutorial, 2004, Accessed: 2025-08-03. [Online]. Available: https://research.ncl.ac.uk/game/mastersdegree/gametechnologies/physicstutorials/4collisiondetection/Physics%20-%20Collision%20Detection.pdf.

[9] J. de Vries, *Learn OpenGL*. Kendall & Welling, 2020. [Online]. Available: https://learnopengl.com/book/book_pdf.pdf.