# Fluid Simulation: Mathematical Breakdown

Neo

February 21, 2025

## 1 Introduction

In this document I explain the physics and mathematics used for my fluid simulation, enjoy!

## 2 Underlying Physics

### 2.1 Simplified Traveling Waves

In more advanced fluid simulations fluids are most often based off of the Navier-Stokes equations and sometimes Lagrangian mechanics to simulate physically accurate water. However as off writing this paper I'm not fully comfortable with these methods so they weren't used. In this project, we use a simplified model of superimposed traveling waves:

$$y(x,\, z,\, t) \;=\; \sum_i \Big( a_i \,\sin\big[ f_i \big( \mathbf{d}_i \cdot (\mathbf{p} - s_i\, t) \big) \big] \Big),$$

where $\mathbf{p} = (x,\, z)$ is the horizontal position of a vertex, $\mathbf{d}_i$ are unit vectors representing wave directions, $a_i$ are amplitudes, $f_i$ are frequencies, and $s_i$ are speeds. This approach captures a convincing wave-like surface without solving the full fluid equations.

### 2.2 Physical Interpretation

- **amplitude** ($a_i$) controls the wave height. Real ocean waves can have amplitudes ranging from centimeters to meters, depending on wind, fetch, and water depth.
- **frequency** ($f_i$) controls the wavelength. Higher frequency means shorter wavelength. Real waves often follow a spectrum of frequencies rather than just one or two discrete values.
- **direction** ($\mathbf{d}_i$) sets the direction of wave travel. In reality, waves often come from multiple directions due to wind shifts, refraction, and interference.
- **speed** ($s_i$) in the code is a simple scalar that shifts phase over time. True wave speeds depend on dispersion relations (e.g., deep-water vs shallow-water wave equations).

This simulation does not account for nonlinear wave interactions, refraction, reflection, or dispersion. It simply overlays two sinusoidal traveling waves to produce a visually appealing surface.

## 3 Wave Generation

In the `updatewaves` function, each vertex on the top surface of the water is displaced by the sum of two sinusoidal waves. For a vertex at position $(x,\, z)$ (extracted from the 3D position $(x,\, y,\, z)$), we compute

$$\alpha_1 \;=\; \mathbf{d}_1 \cdot \big( (x,\, z) \;-\; \mathbf{s}_1\, t \big), \qquad \alpha_2 \;=\; \mathbf{d}_2 \cdot \big( (x,\, z) \;-\; \mathbf{s}_2\, t \big),$$

where

- $\mathbf{d}_1, \mathbf{d}_2$ are unit direction vectors (for each wave),
- $\mathbf{s}_1, \mathbf{s}_2$ are speed scalars (multiplied by time $t$),
- $\alpha_1, \alpha_2$ are the dot products used inside the sine function.

Each wave is then given by

$$\text{wave}_1 \;=\; a_1 \,\sin\!\big(f_1\,\alpha_1\big), \qquad \text{wave}_2 \;=\; a_2 \,\sin\!\big(f_2\,\alpha_2\big),$$

where $a_i$ are amplitudes and $f_i$ are frequencies. The total displacement is

$$y \;=\; \text{wave}_1 + \text{wave}_2.$$

The bottom surface is simply shifted downward by a constant thickness:

$$y_{\text{bottom}} \;=\; y_{\text{top}} \;-\; \text{thickness}.$$

# 4 Normal Calculation via Partial Derivatives

After updating the top and bottom surfaces, the code computes vertex normals by approximating partial derivatives in the $x$ and $z$ directions. For a top-surface vertex, we sample neighboring heights:

$$\frac{\partial y}{\partial x} \;\approx\; \frac{y_{\,x+1} - y_{\,x-1}}{2}, \quad \frac{\partial y}{\partial z} \;\approx\; \frac{y_{\,z+1} - y_{\,z-1}}{2}.$$

We label these approximations $\Delta x$ and $\Delta z$ in the code. The normal for the top face is then

$$\mathbf{n}_{\text{top}} \;=\; \big(-\Delta x,\; 1,\; -\Delta z\big) \quad\Longrightarrow\quad \mathbf{n}_{\text{top}} \;=\; \frac{\mathbf{n}_{\text{top}}}{\|\mathbf{n}_{\text{top}}\|}.$$

For the bottom face, we invert directions:

$$\mathbf{n}_{\text{bottom}} \;=\; \big(\Delta x,\; -1,\; \Delta z\big) \quad\Longrightarrow\quad \mathbf{n}_{\text{bottom}} \;=\; \frac{\mathbf{n}_{\text{bottom}}}{\|\mathbf{n}_{\text{bottom}}\|}.$$

# 5 Transformations: Model, View, and Projection

In the vertex shader, each vertex is transformed by the model matrix $\mathbf{M}$, the view matrix $\mathbf{V}$, and the projection matrix $\mathbf{P}$. The code effectively computes:

$$\textbf{gl\_Position} \;=\; \mathbf{P}\,\mathbf{V}\,\mathbf{M}\,\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}.$$

## 5.1 Model Matrix

Here, $\mathbf{M}$ is the identity (no additional transformations), so

$$\mathbf{M}\,(x,\,y,\,z,\,1)^{\mathsf{T}} \;=\; (x,\,y,\,z,\,1)^{\mathsf{T}}.$$

## 5.2 View Matrix (LookAt)

The view matrix $\mathbf{V}$ places the camera in the world. Using a function LookAt($\textbf{cameraPos}, \textbf{target}, \textbf{up}$), the matrix transforms world coordinates so that the camera is located at **cameraPos** looking at **target** with **up** as the upward direction.

### 5.3 Projection Matrix (Perspective)

We create a perspective projection:

$$\mathbf{P} \;=\; \text{perspective}\big(\text{fov},\ \text{aspect},\ \text{near},\ \text{far}\big),$$

often implemented as

$$\mathbf{P} \;=\; \begin{pmatrix} \frac{1}{\text{aspect}\,\tan(\text{fov}/2)} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan(\text{fov}/2)} & 0 & 0 \\ 0 & 0 & -\frac{\text{far}+\text{near}}{\text{far}-\text{near}} & -1 \\ 0 & 0 & -\frac{2\,\text{far}\,\text{near}}{\text{far}-\text{near}} & 0 \end{pmatrix}.$$

# 6 Toon Shading

In the fragment shader, a simple toon shading step is computed:

$$\lambda \;=\; \max\big(\mathbf{n}\cdot\mathbf{l},\ 0\big),$$

where $\mathbf{n}$ is the surface normal and $\mathbf{l}$ is the light direction. We then quantize $\lambda$ into discrete bands using `uSteps` steps:

$$\lambda_{\text{quant}} \;=\; \frac{\lfloor \lambda \times \texttt{uSteps} \rfloor}{\texttt{uSteps}}.$$

Finally, we interpolate between a dark color and a light color:

$$\textbf{color} \;=\; (1-\lambda_{\text{quant}})\,\textbf{darkColor} \;+\; \lambda_{\text{quant}}\,\textbf{lightColor}.$$

This creates the flat, banded look of toon shading.

# 7 Resizing and Aspect Ratio

When the window is resized or toggled fullscreen, a callback updates the aspect ratio:

$$\text{aspect} \;=\; \frac{\text{width}}{\text{height}}.$$

The projection matrix uses this aspect ratio each frame:

$$\mathbf{P} = \text{perspective}\big(45^\circ,\ \text{aspect},\ 0.1,\ 500.0\big).$$

This keeps the water surface centered and prevents stretching or squashing as the window dimensions change.

# 8 Summary

- **wave generation:** uses sinusoidal traveling waves with adjustable amplitude, frequency, speed, and direction
- **physics:** does not solve the full fluid equations, but rather uses a simplified model of superimposed plane waves
- **normal calculation:** approximates partial derivatives in the $(x, z)$ plane for top and bottom surfaces
- **transforms:** applies model, view, and projection matrices to move vertices from object space to clip space
- **toon shading:** quantizes the lambert term into discrete bands and interpolates between dark and light colors
- **resizing:** updates the aspect ratio on window resize or fullscreen changes to keep the water proportionate