Boids - Technical documentation

Vivian Ménard

May 2024

Introduction

The purpose of this document is to provide more details on the key algorithmic and mathematical aspects of my real-time boid simulation project, available on GitHub.

Note: The term "boid" is derived from "bird" and is used to describe entities that interact with each other to form flocks. In my case, these entities are fishes.

1 Entity behavior

1.1 Fish Behavior

The foundation of fish schooling behavior, with the three zones of the field of vision (shown in Figure 1), is inspired by the work of *Craig W. Reynolds* [2].

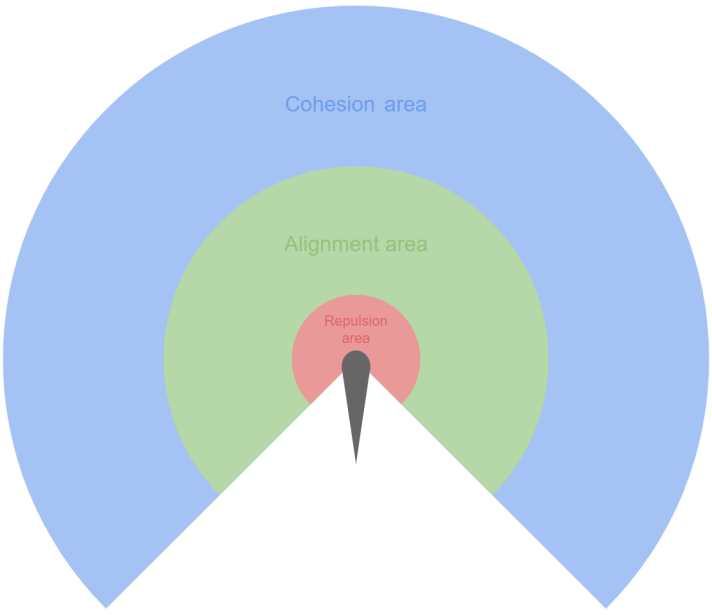

Figure 1: Simplified diagram of the fish's field of vision.

Fish perceive their environment as illustrated in FIGURE 1. The behavior a fish will adopt towards another depends on the zone in which the latter is located:

- If the other fish is outside its field of vision (either too far away or behind), it will ignore it.
- If the other fish is in the repulsion zone (red in FIGURE 1), it will try to move away from it.
- If the other fish is in the cohesion zone (blue in FIGURE 1), it will try to move closer to it.
- If the other fish is in the alignment zone (green in FIGURE 1), it will try to align its direction with it.

A fish's behavior towards a predator is always the same: it will try to move away from it.

In practice, to calculate its new optimal direction regarding nearby fishes and predators, the fish will first compute:

- The optimal direction to move closer to fish in the cohesion zone: the direction towards their barycenter.
- The optimal direction to move away from fish in the repulsion zone: the direction away from their barycenter.
- The optimal direction to align with fish in the alignment zone: the average of their directions.
- The optimal direction to move away from visible predators: the direction away from their barycenter.

The fish's new optimal direction will thus be a weighted average of these four directions and its previous direction.

Note: In practice, to smooth the entities' behavior, the separation between the different zones is not abrupt as shown in FIGURE 1, but rather continuous.

1.2 Predator Behavior

Predators exhibit simpler behavior compared to fish. They will only try to move closer to the fish they see and move away from other predators. The principle remains the same: to calculate its new optimal direction, the predator will first compute:

- The optimal direction to move closer to visible fish: the direction towards their barycenter.
- The optimal direction to move away from other visible predators: the direction away from their barycenter.

The predator's new optimal direction will thus be a weighted average of these two directions and its previous direction.

1.3 State System

To add diversity to their behavior, entities can be in different states that influence how they interact with their environment.

There are three possible states for the boids: AFRAID if they are fleeing from a predator, ALONE if they are isolated, and NORMAL otherwise.

There are also three possible states for the predators: HUNTING if they are searching for boids, ATTACKING if they have found and are attacking them, and CHILL otherwise.

1.3.1 State Characteristics

Each state is characterized by:

- A target velocity for the entity;
- Whether or not it is an *emergency state*;
- Whether the entity's trajectory is calculated based on its surroundings or follows a pseudo-random walk.

When the entity changes state, its target velocity will change, and it will then smoothly accelerate or decelerate to reach the new target velocity (an immediate change in velocity would not be visually pleasing). If the entity's current state is an *emergency state*, it will accelerate or decelerate more quickly to reach the target velocity of that state, as illustrated in FIGURE 2. The states classified as *emergency states* are AFRAID and ATTACKING.

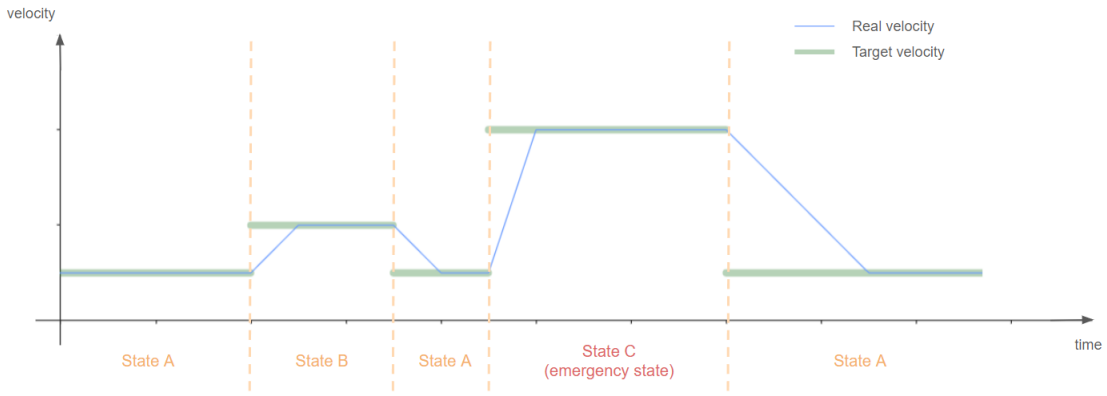

Figure 2: Graph illustrating the evolution of an entity's real velocity based on its target velocity, state, and the urgency of its state.

Each state is also characterized by the behavior the entity should adopt, which can either consider its environment to choose the best direction or follow a pseudo-random walk in 3D space. The states involving a random walk are ALONE and CHILL.

The specific characteristics of each state are detailed in the table in FIGURE 3.

Entity	State	Velocity	Emergency	Random Walk
Boid	NORMAL	Low	No	No
	ALONE	Medium	No	Yes
	AFRAID	High	Yes	No
Predator	CHILL	Low	No	Yes
	HUNTING	Medium	No	No
	ATTACKING	Very High	Yes	No

Figure 3: Summary table of the characteristics of each state.

1.3.2 State Transitions

The logic used to determine the new state of a boid is detailed in FIGURE 4, and the result depends only on the number of boids and predators nearby.

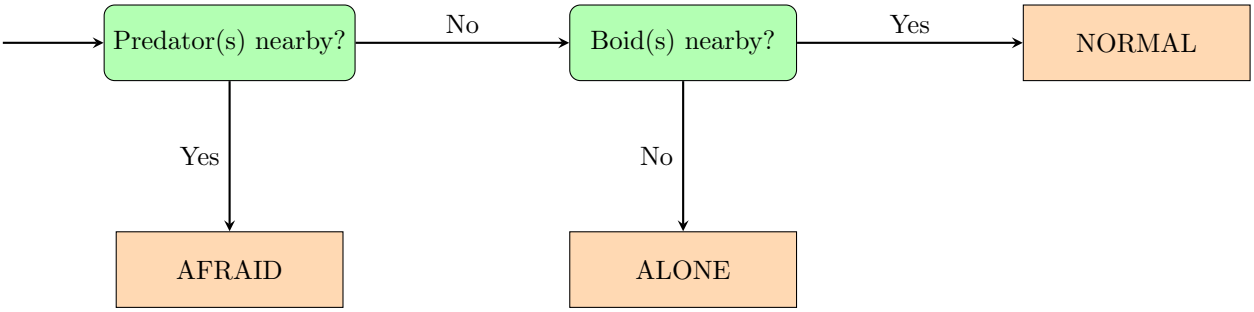

Figure 4: Decision diagram for determining a boid's state based on nearby entities.

This state system of predators is a non-deterministic finite state machine, illustrated in FIGURE 5.

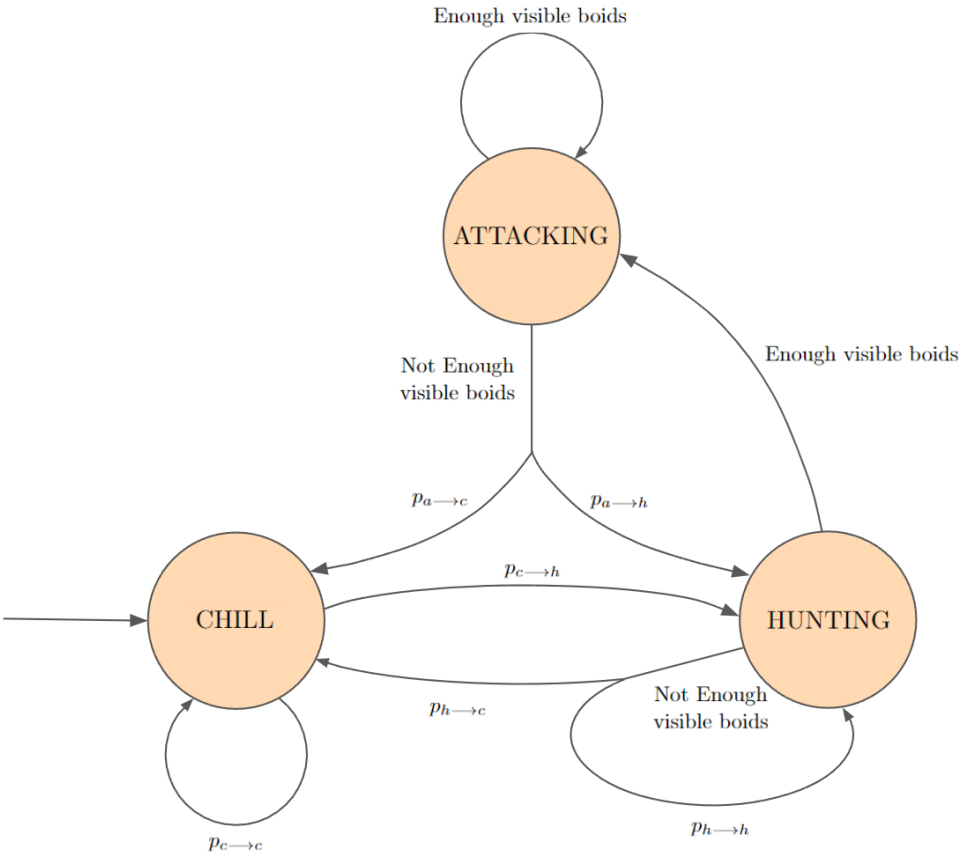

Figure 5: Predator state machine diagram.

With:

- $p_{c \longrightarrow c}$: the probability that a CHILL predator at turn t remains CHILL at turn $t+1$.
- $p_{c \longrightarrow h}$: the probability that a CHILL predator at turn t becomes HUNTING at turn $t+1$.
- $p_{h \longrightarrow h}$: the probability that a HUNTING predator at turn t remains HUNTING at turn $t+1$ (if there are not enough boids in its field of vision).
- $p_{h \longrightarrow c}$: the probability that a HUNTING predator at turn t becomes CHILL at turn $t+1$ (if there are not enough boids in its field of vision).
- $p_{a \longrightarrow c}$: the probability that an ATTACKING predator at turn t becomes CHILL at turn $t+1$ (if there are not enough boids in its field of vision).
- $p_{a \longrightarrow h}$: the probability that an ATTACKING predator at turn t becomes HUNTING at turn $t+1$ (if there are not enough boids in its field of vision).

1.4 Obstacle Avoidance

The obstacle avoidance process used by the entities is illustrated in FIGURE 6. It works as follows:

- With each change of direction, the entity will check that the path is clear in the new desired direction using a raycast (*detection raycast* in the diagram).
- If the raycast detects an obstacle, the entity will need to avoid it by adjusting its direction according to one of the predefined *global avoidance directions*. In 2D (as shown in the diagram), there are only two possible directions: left and right. In 3D, however, there are infinitely many possible directions; I have chosen to use eight: left, right, up, down, and their combinations (upper-left, upper-right, etc.).
- An *avoidance angle* will be chosen based on the distance to the detected obstacle: the closer the obstacle, the closer this angle will be to 90°; the farther the obstacle, the closer this angle will be to 0°.
- A *global avoidance direction* combined with an *avoidance angle* yields a *possible avoidance direction* (the initial direction is deflected by the *avoidance angle* in the *global avoidance direction*). The entity will perform a raycast (*selection raycast* in the diagram) in each of the eight *possible avoidance directions* and will choose the one in which the obstacle is the farthest away (with the convention that $+\infty$ is the distance to the obstacle if none is detected).

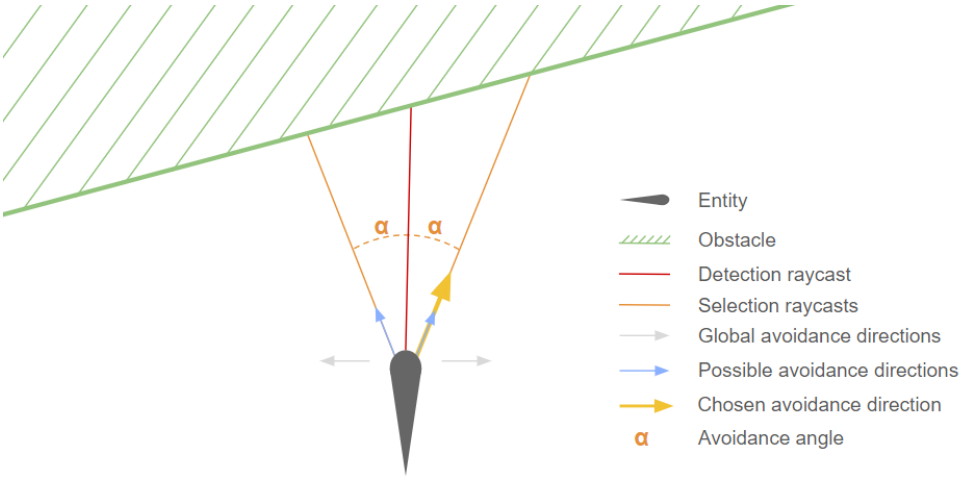

Figure 6: Diagram of the obstacle avoidance process for entities.

Note: A preference mechanism allows sharks to favor certain *global avoidance directions* (specifically, they favor horizontal directions).

1.5 Key Optimizations

In this simulation, the calculation of a new direction that each entity must regularly perform is a major performance bottleneck. Therefore, I have implemented several optimizations (though more could be done, for example, by using a compute shader or UNITY DOTS).

Each calculation of a new direction requires the entity to probe its environment within a certain radius to know the position and nature of nearby entities. To best utilize Unity's spatial optimization structures, a `Physics.OverlapSphere` is used.

If the distance at which a boid can see was constant, it would have many more neighbors to consider in its direction calculations when it is in a flock. However, observations of animal flocks [3] have shown that in such structures, each individual only considers its six closest neighbors. Thus, I have made the boid's vision distance adaptive. Every time a boid probes its environment, it adjusts its vision distance based on the number of neighbors found (if it finds more than six, it will know that next time it can look less far, and vice versa if it finds fewer). This approach reduces the computational complexity for each boid, especially in dense flocks.

Furthermore, to reduce the CPU's computational load, each entity does not calculate a new direction at every `FixedUpdate`, but every N `FixedUpdate`s (in my project, $N = 10$). The entity calculates a new optimal direction at `FixedUpdate` n, then gradually changes its current direction to match the new optimal direction between `FixedUpdate` n and `FixedUpdate` $n + N - 1$. It only calculates a new optimal direction at `FixedUpdate` $n + N$. This approach distributes the computational load across different `FixedUpdate`s: at each `FixedUpdate`, only a fraction $1/N$ of all entities updates its direction.

2 Procedural Animation

In each `FixedFrame`, each entity records in a history log its current position (specifically, the position of its head), its current rotation, and the distance it has traveled since the last frame.

The basic principle of my procedural animation, illustrated in FIGURE 7, is as follows:

- An entity has a spine made up of bones whose distances from the head are precomputed.
- In each `FixedFrame`, we traverse the entity's past trajectory to position its various bones along it.
- The position where we place the bone on the entity's trajectory depends on its initial (precomputed) distance from the head. If a bone is initially $4u$ from the head of its entity, it will be placed $4u$ behind the head of the entity on its trajectory.
- We also assign each bone the rotation that the head had when it was at that position (if the bone falls between two entries in our history log, we perform a `Slerp` between the two rotations to determine the appropriate rotation).

Note: The position history is deleted progressively; we only keep in memory the portion of the past trajectory that is still useful for the entity's animation.

Note: For sharks, a sinusoidal component is also added to the bones. First, we position the bones on the head's trajectory as we do for the fish, then we add the sinusoidal component relative to the local left-right axis of the trajectory. This approach requires adjusting the bones' rotations afterwards to match their new positions.

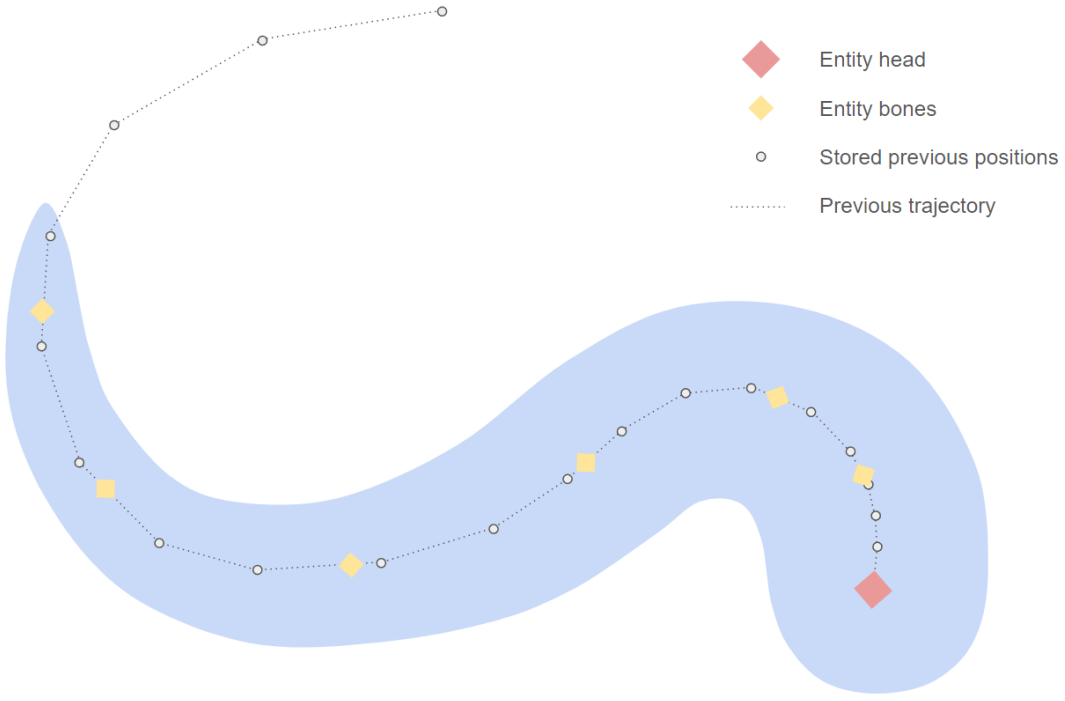

Figure 7: Diagram of the procedural animation process for entities.

3 Water Movement

Note: In Unity, the vertical axis is the Y axis, so the XZ plane is the horizontal plane of the world. This convention will be used throughout the calculations.

3.1 Vertical Basis of Movement

The movement of water is essentially based on the mathematical principles presented in the video "How Video Games Fake Water" [1].

The basic idea is to sum a large number of periodic plane waves with different propagation directions and parameters (propagation speed, spatial frequency, and amplitude).

A **plane wave** is a wave for which the associated disturbance is uniform in all directions perpendicular to the direction of propagation.

One of the simplest examples of a periodic plane wave is the **sine plane wave**, which at each point $\vec{p}$ on the XZ plane and at each instant t, associates a vertical disturbance $\psi_{sin}(\vec{p}, t)$, such that:

$$\psi_{sin}(\vec{p}, t) = Asin(\omega_s \vec{p} \cdot \vec{d} + \omega t)$$

Where:

- A: the amplitude of the wave.
- ω_s: the spatial angular velocity of the wave ($\omega_s = 2\pi f_s$, where f_s is the spatial frequency of the wave).
- $\vec{d} \in \mathbb{R}^2$: the horizontal propagation direction of the wave.
- ω: the temporal angular velocity of the wave ($\omega = 2\pi f$, where f is the temporal frequency of the wave).

However, a better candidate, as it naturally resembles a wave (cf. FIGURE 8), is the periodic plane wave ψ, which at each point $\vec{p}$ on the XZ plane and at each instant t, associates a vertical disturbance $\psi(\vec{p}, t)$, such that:

$$\psi(\vec{p}, t) = A(e^{sin(\omega_s \vec{p} \cdot \vec{d} + \omega t)} - e^{-1})$$

Where A, ω_s, $\vec{d}$, and ω are defined as before.

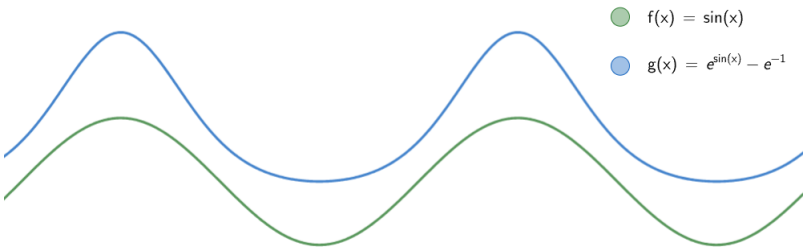

Figure 8: Comparison of appearance between a simple sinusoidal plane wave (in green) and a wave of the type ψ (in blue).

In the following, we will thus use this periodic plane wave rather than ψ_{sin}.

We can then obtain a first version of the water movement by summing a number N of such waves ($N = 16$ in the context of my project). For this first version, the vertical disturbance at a point $\vec{p}$ on the XZ plane at a time t is thus:

$$\Psi(\vec{p}, t) = \sum_{i=1}^{N} \psi_i(\vec{p}, t)$$

Where, $\forall i \in [\![1; N]\!]$:

$$\psi_i(\vec{p}, t) = A_i(e^{sin(\omega_{s,i}\vec{p}\cdot\vec{d_i}+\omega_i t)} - e^{-1})$$

Where A_i, $\omega_{s,i}$, $\vec{d_i}$, and ω_i are respectively the amplitude, spatial angular velocity, horizontal propagation direction, and temporal angular velocity of the wave ψ_i.

Parameter choices for each ψ_i: To have coherent waves, we will start by adding slow components (low ω), wide (low ω_s), and high-amplitude (A high), then we will add progressively faster components (high ω), narrower (high ω_s), and low-amplitude (A low) to add detail to our waves. The propagation directions are chosen randomly in the XZ plane.

3.2 Adding Lateral Movements

To give more life to the water by creating the impression that it gathers to form waves, the video [1] suggests adding a lateral movement to the surface vertices based on the partial derivatives of the disturbance. The idea is to move, at each instant t, each vertex of the surface according to:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto \begin{pmatrix} x + \alpha\frac{\partial\Psi}{\partial x}(\begin{pmatrix} x \\ z \end{pmatrix}, t) \\ y + \Psi(\begin{pmatrix} x \\ z \end{pmatrix}, t) \\ z + \alpha\frac{\partial\Psi}{\partial z}(\begin{pmatrix} x \\ z \end{pmatrix}, t) \end{pmatrix}$$

Where α is the intensity of the lateral movements.

The results of this method are visually convincing, but the lateral movements of the vertices are problematic in the project context (issues at the edges of the cube). To achieve a similar effect without lateral movements, I use another strategy, which is more computationally expensive but better suited to my use case.

For any point $\vec{p}$ on the XZ plane, at any instant t, we start by calculating $\vec{\tilde{p}}(\vec{p}, t)$, which is an estimation of the point on the XZ plane that should have been at position $\vec{p}$ at time t with the method that actually moves the vertices.

$$\vec{\tilde{p}}(\vec{p}, t) = \begin{pmatrix} x - \alpha\frac{\partial\Psi}{\partial x}(\vec{p}, t) \\ z - \alpha\frac{\partial\Psi}{\partial z}(\vec{p}, t) \end{pmatrix}$$

Where α is the intensity of the lateral movements.

We can then apply the disturbance of $\vec{p}(\vec{p},t)$ to the point $\vec{p}$, giving the impression that the point has moved. We can then define $\tilde{\Psi} : (\vec{p},t) \mapsto \Psi(\vec{\tilde{p}},t)$, and the final movement of the surface vertices of the water will be defined at any time t by:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto \begin{pmatrix} x \\ y + \tilde{\Psi}(\begin{pmatrix} x \\ z \end{pmatrix}, t) \\ z \end{pmatrix}$$

Note: The two approaches are numerically different, as $\frac{\partial \Psi}{\partial x}(\vec{p},t)$ is a priori different from $\frac{\partial \Psi}{\partial x}(\vec{\tilde{p}},t)$, but the visual result is substantially similar.

Note: Calculating $\vec{\tilde{p}}$ requires being able to compute the partial derivatives of Ψ, but since the latter is a sum of composed of usual functions, this calculation poses no mathematical problem:

$$\begin{aligned} &\forall \vec{p} \in \mathbb{R}^2, \forall t \in \mathbb{R}, \frac{\partial \Psi}{\partial x}(\vec{p},t) \\ &= \frac{\partial}{\partial x} \sum_{i=1}^{N} \psi_i(\vec{p},t) \\ &= \sum_{i=1}^{N} \frac{\partial \psi_i}{\partial x}(\vec{p},t) \\ &= \sum_{i=1}^{N} A\omega_{s,i}\vec{d_i} \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} cos(\omega_{s,i}\vec{p} \cdot \vec{d_i} + \omega_i t) e^{sin(\omega_{s,i}\vec{p} \cdot \vec{d_i} + \omega_i t)} \end{aligned}$$

Note: I used $\vec{\tilde{p}}$, which takes into account the fictitious displacement of the point, to apply the foam texture on the water that seems to move with the waves.

3.3 Calculation of Normals

In 2D, a vector tangent to the curve of the function f at the point $A \begin{pmatrix} x_0 \\ f(x_0) \end{pmatrix}$ is $\vec{T} \begin{pmatrix} 1 \\ f'(x_0) \end{pmatrix}$. It is then possible to find a normal vector $\vec{N}$ to the same point of the function by taking its perpendicular, as illustrated in FIGURE 9.

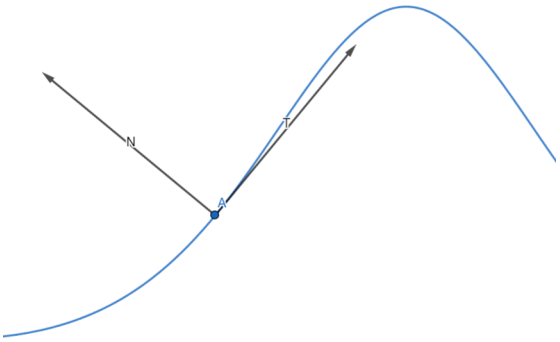

Figure 9: Tangent and normal vectors to a function at a point on its curve.

In a similar way, in 3D we can construct the tangent plane to the curve of a function at a point $B(x_0, f(x_0, z_0), z_0)$ as being the plane defined by the point B and the vectors $\vec{T_x}(1, \frac{\partial f}{\partial x}(x_0, z_0), 0)$ and $\vec{T_z}(0, \frac{\partial f}{\partial z}(x_0, z_0), 1)$. We then obtain the normal vector $\vec{N}$ by taking the perpendicular to the tangent plane: $\vec{N} = \vec{T_z} \otimes \vec{T_x}$.

We have seen in Section 3.2 that there is no mathematical problem in calculating the partial derivatives of the water surface. By using the technique just described, we can therefore calculate the normal at any point of the water surface and at any time, which allows us to compute water reflections accurately.

References

[1] Acerola, *How Games Fake Water*. https://www.youtube.com/watch?v=PH9q0HNBjT4&list=PLz_az9HWXUw1zG6iDTOjvVzGS2qwYiJ_n&index=22&ab_channel=Acerola.

[2] Craig W. Reynold, *Flocks, Herds, and Schools: A Distributed Behavioral Model*, computer graphics, 1987. https://team.inria.fr/imagine/files/2014/10/flocks-hers-and-schools.pdf.

[3] M. Ballerini, N. Cabibbo, R. Candelier, A. Cavagna, E. Cisbani, I. Giardina, ... V. Zdravkovic, *Interaction ruling animal collective behavior depends on topological rather than metric distance: Evidence from a field study*, proceedings of the national academy of sciences, 2008. https://www.pnas.org/doi/10.1073/pnas.0711437105.