Big Fish
3D Simulation of web performance

Tacit Knowledge
Ahmed Layouni

2012-2013

Contents

Chapter 1

Introduction

Performance pressure just keeps growing; to drive more sales and boost brand image, todays Websites are increasingly dependent on sophisticated technologies that attract attention, hold interest or move visitors towards their virtual shopping cart. But if the technology behind the marketing vision creates delays or fails to work properly, visitors may quickly abandon the site and run to the competition.

The subject of this document is Big Fish, an online tool built on big data technology repurposed from Tacit Knowledge to monitor the availability and site performance of every major e-commerce site in the United States and the United Kingdom. It pulls and aggregates information from Alexa for traffic size, and Google for page speed to monitor over 2000 e-commerce sites.

Big Fish is a 3D animation based on ThreeJS, a lightweight cross-browser JavaScript framework used to create and display animated 3D computer graphics on a web browser. It contrasts online store performance using a 3D graphical data view that allows to compare websites performance in a pleasant way, based on three metrics: response time, traffic and availability.

The idea came as an attempt to break the new and distinguish Tacit Knowledge as an innovative brand. The concept consists of representing each website by a swimming fish in the ocean. The size of the fish represents traffic, while its velocity represents response time. If the website is down the fish dies and floats into the surface.

Since the main purpose of the project was to impress and attract visitors' attention, the challenge was to design a motion that doesn't repeat itself and looks as realistic as possible, while still representing data accurately. To that end 3D movement trajectories have been cautiously designed to allow perceiving websites basic parameters clearly, while tolerating collisions to happen so that a collision prediction correlated to a collision avoidance algorithms can operate to add an interactive, realistic touch to the motion.

This article serves as documentation for the motion design, as well as the collision prediction and collision avoidance algorithms.

Chapter 2

Movement design

The goal of this chapter is to explain the raw motion design, based on which the collision prediction and avoidance algorithms will be established.

The velocity of a fish is meant to represent its site's response time; in order to clearly notice it, circular trajectories have been chosen to define the nature of the movement.

However the trajectory might change depending on a fish's state. For this reason, a fish aggregates the motion logic to its state thereby applying the strategy design pattern.

Note that the rest of this section assumes the reader to be familiar with Euler rotations which are a corner stone in 3D development. Figure 2.1 can help understanding the rotations mechanism; it illustrates step by step the application of Euler rotations given ZXY as Euler order:

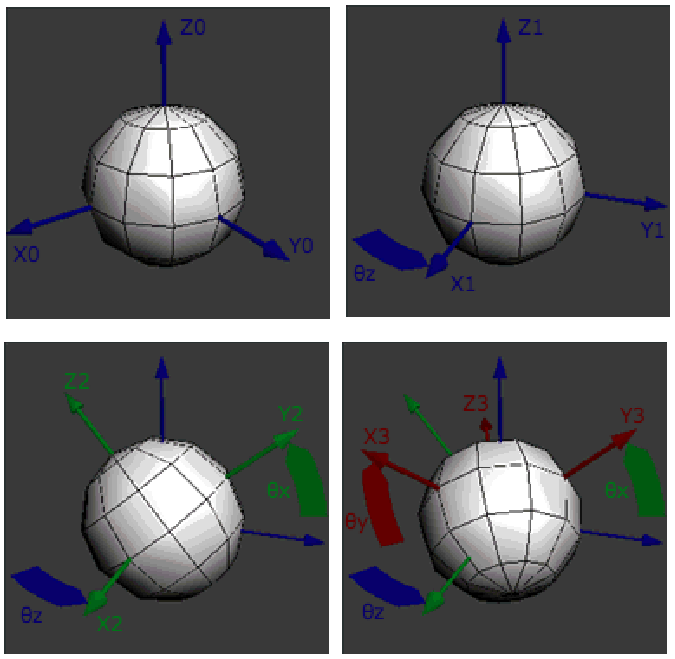

Figure 2.1: Euler Rotations

The 4 snapshots illustrated in figure 2.1 represent:

- The axis initial states before any rotation gets applied.
- The axis after applying a rotation θ_z around the Z axis.
- The axis after applying θ_z followed by θ_x.
- The axis after applying θ_z followed by θ_x followed by θ_y.

The rest of the chapter describes the different states a fish might go through.

2.1 Swimming state:

The swimming state is where a fish spends most of its time. Even when a particular event triggers a change of state, as soon as the cause disappears, the fish immediately returns to the swimming state.

The movement nature of the swimming state is circular; the radius and the center of the circle is common to all the fishes in the scene. This means that all fishes are swimming within the same sphere surface. However to create diversity in the movement, the planes holding each fish's trajectory are different. Considering YXZ as the Euler rotation order, all the trajectories' planes have the same θ_x rotation defined as a constant to represent the planes' inclination, whereas θ_y rotations are different per fish. The difference is obtained by adding a $\Delta\varphi$ deviation whenever a new fish gets loaded.

A plane's orthonormal coordinate system is defined by two orthonormal axis and a center. The center is common to all the trajectories' planes i.e. the center of the scene, whereas calculating the axis coordinates required some mathematical analysis that led to the following results:

Let $\overrightarrow{X_1}$, $\overrightarrow{X_2}$, $\overrightarrow{X_3}$ be the rotated axis, where $X_1X_2X_3$ denotes Euler order, and let θ_1, θ_2 and θ_3 be the corresponding Euler rotations. If we consider $(\overrightarrow{x_1}, \overrightarrow{x_2}, \overrightarrow{x_3})$ as the reference coordinate system, $\overrightarrow{X_1}$, $\overrightarrow{X_2}$ and $\overrightarrow{X_3}$ coordinates can be obtained by applying the following formula:

$$\overrightarrow{X_1} = \begin{pmatrix} \cos\theta_2\cos\theta_3 \\ \sin\theta_3\cos\theta_1 + \cos\theta_3\sin\theta_2\sin\theta_1 \\ \sin\theta_3\sin\theta_1 - \cos\theta_3\sin\theta_2\cos\theta_1 \end{pmatrix}_{(\overrightarrow{x_1},\overrightarrow{x_2},\overrightarrow{x_3})}$$

$$\overrightarrow{X_2} = \begin{pmatrix} -\sin\theta_3\cos\theta_2 \\ \cos\theta_3\cos\theta_1 - \sin\theta_3\sin\theta_2\sin\theta_1 \\ \cos\theta_3\sin\theta_1 + \sin\theta_3\sin\theta_2\cos\theta_1 \end{pmatrix}_{(\overrightarrow{x_1},\overrightarrow{x_2},\overrightarrow{x_3})}$$

$$\overrightarrow{X_3} = \begin{pmatrix} \sin\theta_2 \\ -\cos\theta_2\sin\theta_1 \\ \cos\theta_2\cos\theta_1 \end{pmatrix}_{(\overrightarrow{x_1},\overrightarrow{x_2},\overrightarrow{x_3})}$$

Mapping the indices to the corresponding axis will convert the coordinates relatively to the reference coordinate system, thus giving the new rotated axis $(\overrightarrow{X}, \overrightarrow{Y}, \overrightarrow{Z})$.

Example

If we consider YXZ for instance as the Euler rotation order, $(\overrightarrow{X_1}, \overrightarrow{X_2}, \overrightarrow{X_3})$ would map to $(\overrightarrow{Y}, \overrightarrow{X}, \overrightarrow{Z})$, and $(\overrightarrow{x_1}, \overrightarrow{x_2}, \overrightarrow{x_3})$ would map to $(\overrightarrow{y}, \overrightarrow{x}, \overrightarrow{z})$, where $\overrightarrow{X}$, $\overrightarrow{Y}$, $\overrightarrow{Z}$ are the results of applying Euler rotations to the reference coordinate system $\overrightarrow{x}$, $\overrightarrow{y}$, $\overrightarrow{z}$ respectively. Applying the formula above gives:

$$\overrightarrow{Y} = \begin{pmatrix} \cos\theta_x \cos\theta_z \\ \sin\theta_z \cos\theta_y + \cos\theta_z \sin\theta_x \sin\theta_y \\ \sin\theta_z \sin\theta_y - \cos\theta_z \sin\theta_x \cos\theta_y \end{pmatrix}_{(\overrightarrow{y}, \overrightarrow{x}, \overrightarrow{z})}$$

$$\overrightarrow{X} = \begin{pmatrix} -\sin\theta_z \cos\theta_x \\ \cos\theta_z \cos\theta_y - \sin\theta_z \sin\theta_x \sin\theta_y \\ \cos\theta_z \sin\theta_y + \sin\theta_z \sin\theta_x \cos\theta_y \end{pmatrix}_{(\overrightarrow{y}, \overrightarrow{x}, \overrightarrow{z})}$$

$$\overrightarrow{Z} = \begin{pmatrix} \sin\theta_x \\ -\cos\theta_x \sin\theta_y \\ \cos\theta_x \cos\theta_y \end{pmatrix}_{(\overrightarrow{y}, \overrightarrow{x}, \overrightarrow{z})}$$

By converting the coordinates to the reference axises order $(\overrightarrow{x}, \overrightarrow{y}, \overrightarrow{z})$, we obtain:

$$\overrightarrow{X} = \begin{pmatrix} \cos\theta_z \cos\theta_y - \sin\theta_z \sin\theta_x \sin\theta_y \\ -\sin\theta_z \cos\theta_x \\ \cos\theta_z \sin\theta_y + \sin\theta_z \sin\theta_x \cos\theta_y \end{pmatrix}_{(\overrightarrow{x}, \overrightarrow{y}, \overrightarrow{z})}$$

$$\overrightarrow{Y} = \begin{pmatrix} \sin\theta_z \cos\theta_y + \cos\theta_z \sin\theta_x \sin\theta_y \\ \cos\theta_x \cos\theta_z \\ \sin\theta_z \sin\theta_y - \cos\theta_z \sin\theta_x \cos\theta_y \end{pmatrix}_{(\overrightarrow{x}, \overrightarrow{y}, \overrightarrow{z})}$$

$$\overrightarrow{Z} = \begin{pmatrix} -\cos\theta_x \sin\theta_y \\ \sin\theta_x \\ \cos\theta_x \cos\theta_y \end{pmatrix}_{(\overrightarrow{x}, \overrightarrow{y}, \overrightarrow{z})}$$

$\overrightarrow{X}$, $\overrightarrow{Y}$ and $\overrightarrow{Z}$ denote the axis of a fish relatively to its rotations. Given the $\overrightarrow{Y}$ axis is normal to the trajectory's plane, the coordinates of a fish relatively to $(\overrightarrow{X}, \overrightarrow{Y}, \overrightarrow{Z})$ can be obtained as follows:

$$\begin{pmatrix} R.\sin\theta_y \\ 0 \\ R.\cos\theta_y \end{pmatrix}_{(\overrightarrow{X},\overrightarrow{Y},\overrightarrow{Z})}$$

Where R is the radius of the circular trajectory and θ_y denoting the curvilinear abscissa. Applying the conversion matrix obtained by combining $\overrightarrow{X}$, $\overrightarrow{Y}$ and $\overrightarrow{Z}$ would convert the coordinates relatively to $(\overrightarrow{x}, \overrightarrow{y}, \overrightarrow{z})$.

Having all the ingredients to calculate a fish's absolute coordinates depending on its curvilinear abscissa θ_y within it's circular trajectory, the movement logic for a fish in the swimming state is implemented by adding an angular move step δ_y to θ_y each new frame. δ_y gets calculated as follows:

$$\delta_y = \pm\frac{Cste}{R \times (response\ time)}$$

Note that δ_y is inversely proportional to the radius R, which makes sense since a reduction of radius should result in faster round turns executions, and vice versa. Furthermore δ_y is also inversely proportional to the corresponding website's response time, which means the smaller the response time is, the greater δ_y gets and thus the faster moves the fish. "*Cste*" represents a configuration parameter that allows to tweak the responsiveness of the animation, while the sign (negative or positive) directing a fish's orientation gets assigned randomly. Note that δ_y is also used to adjust the frequency of the animation of a fish's tail proportionally to its speed.

Another issue to consider, is the orientation of fishes. The orientation defines the direction a fish should point to while swimming accordingly with its movement. In other words a fish should not be moving in a direction while pointing (looking) to another direction. The right orientation can be obtained by making a fish look at its next position. In mathematical terms, if we consider M as a fish's current position, and M' as the immediate next position, the orientation $\overrightarrow{o}$ should match $\overrightarrow{MM'}$. M' coordinates can be obtained by applying the following formula:

$$\begin{pmatrix} R.\sin(\theta_y + \delta_y) \\ 0 \\ R.\cos(\theta_y + \delta_y) \end{pmatrix}_{(\overrightarrow{X},\overrightarrow{Y},\overrightarrow{Z})} \quad . \quad CM$$

Where CM denotes the conversion matrix used to convert the coordinates from $(\overrightarrow{X}, \overrightarrow{Y}, \overrightarrow{Z})$ the fish relative coordinates system, to $(\overrightarrow{x}, \overrightarrow{y}, \overrightarrow{z})$ the scene coordinates system.

2.2 Avoiding state

While a fish is in the swimming state or in phase 3 of the avoiding state (see chapter 4), a collision prediction algorithm keeps listening to its movement and predicting whether pursuing the movement within the same trajectory would result in an imminent collision; if so the fish will switch its state to the avoiding state.

2.3 Dying state

A fish switches to the dying state when the corresponding website stops responding. The dying state goes through three steps:

1. Slow down decreasingly until the fish stops moving.
2. Rotate the fish slowly around its relative $\overrightarrow{z}$ axis until it lays on its back, and turns upside down.
3. Move the fish slowly towards the water surface while rotating around its $\overrightarrow{y}$ axis.

2.4 Dead state

Once a dying fish reaches the water surface, it immediately switches its state to the Dead state during which the fish keeps oscillating vertically in the surface to simulate the floating motion.

2.5 Reviving state

A dead fish keeps listening to its availability status; as soon as it changes to "Online", the fish immediately switches to the Reviving state. The Reviving state can be thought of as a transitional phase towards the Swimming state. It runs through 4 phases:

1. Slowly rotate the fish around its $\overrightarrow{Z}$ axis until it gets right-side-up again.
2. Prepare the fish to dive by rotating it slowly around its $\overrightarrow{X}$ axis until it points to the bottom of the sea.
3. Dive the fish towards the bottom of the sea until it gains back its last position where it died.
4. Rotate the fish increasingly around its $\overrightarrow{X}$ and $\overrightarrow{Y}$ axis until it is oriented in accordance with its swimming trajectory.

Once the fish oriented appropriately, it switches back to the swimming state.

2.6 Resizing state

A fish switches its state to Resizing when the corresponding website's traffic changes. During this state a fish keeps shrinking and expanding repeatedly between the current and target size until it stabilizes. Once stabilized, the fish switches back to the swimming state.

The relationship between the scale of the Morph representing a fish and the corresponding traffic size is proportional:

$$(Fish\ Scale) = (Cste) \times (Traffic)$$

Where $Cste$ is a configuration parameter that allows to tweak the proportion scale/traffic as needed.

Chapter 3

Collision prediction

Movement trajectories have been designed to intersect with the goal of tolerating collisions to happen so that a collision prediction correlated to a collision avoidance algorithms can operate and add an interactive touch preventing the motion from repeating itself and making a pleasant animation that is interesting to watch.

Note the term prediction instead of detection; a traditional collision detection algorithm detects a collision the moment it happens, whereas what is needed is to predict collisions before they happen so that the involved fishes get enough time and space to react and avoid each other.

This chapter explains in details the collision prediction mechanism.

3.1 Collision prevention setup

The collision prediction and avoidance algorithms need to compute important parameters only once at load time to cache them for ulterior use when the animation starts which will prevent re-performing the same logic each frame and gain in terms of performance.

The list of operations applied to each loaded fish during the collision prevention setup includes:

- Add a movement listener to the fish, so that whenever its position evolves as it moves, the collision prediction algorithm gets invoked.

- Compute trajectories intersections; will be explained in details under sub-section 3.1.1.
- Determine which fishes are sharing the same trajectory if any. Given all trajectories are circular having the same radius and center, 2 fishes share the same trajectory if they circulate within the same plane. 2 trajectories share the same plane if they share the same relative rotated Y axis; the latter being normal to a trajectory's plane.
- Compute the "unit grid box" size based on the fishes' bounding box. Briefly explained, the proximity collision prediction algorithm divides the scene in a network of big grid boxes which will be cleared and repopulated by fishes each new frame. For better performance these unit grid boxes should be big enough to fully carry the biggest fish in the scene while considering a margin. More details about calculating the unit grid box will be explained under sub-section 3.1.2.

3.1.1 Intersections

Figure 3.1 lists all the data encapsulated under the Intersection structure:

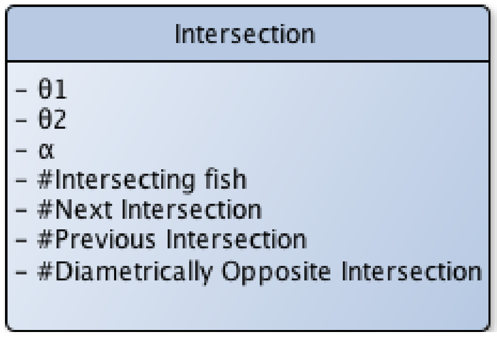

Figure 3.1: Intersection Data Model

In this section I will go through each field listed above.

Curvilinear abscissas (θ_1, θ_2)

The intersections, as opposed to what may come in mind in the first intuition, are not stored as 3D points, if they were they would need to be recalculated every time a fish avoids a collision by changing the radius of its trajectory (more details in chapter 4). To prevent this, and also because the movement is managed by θ_Y (the curvilinear abscissa of a fish within its circular trajectory), it is more efficient to define an intersection as a curvilinear abscissa as well. Proceeding thereby will make it easier to determine if a fish is close, leaving or going towards an intersection based on its curvilinear abscissa and the sign of its angular move step δ_y.

Note that an intersection has two different curvilinear abscissas θ_1 and θ_2 each being relative to the corresponding intersecting trajectory.

Another advantage of using a curvilinear abscissa to locate an intersection, is that each intersection has a diametrically opposite intersection (two intersecting trajectories always intersect twice in two diametrically opposite points), with that in mind only half intersections need to be calculated, the second half will be obtained by respectively adding π to each of the former intersections.

A mathematical analysis led to the following θ_1 and θ_2 formulas:

$$\begin{cases} \theta_2 = \arctan\left(\dfrac{Z_{2y}\ .\ (X_{1z}\ .\ Z_{1x} - X_{1x}\ .\ Z_{1z}) + (X_{1x}\ .\ Z_{2z} - X_{1z}\ .\ Z_{2x})}{Z_{1y}\ .\ (X_{1z}\ .\ X_{2x} - X_{1x}\ .\ X2_{2z})}\right) \\ \\ \theta_1 = \pm \arccos\left(\dfrac{Z_{2y}}{Z_{1y}}\ .\ \cos\theta_2\right) \end{cases}$$

Where (X_{ix}, X_{iy}, X_{iz}), and (Z_{ix}, Z_{iy}, Z_{iz}) are the respective coordinates of the rotated axis $\overrightarrow{X_i}$ and $\overrightarrow{Z_i}$ defining trajectory$_i$'s plane (rotated axis formulas were given in section 2.1). Note that the sign of θ_1 gets determined by processing further equations which I omitted on purpose to avoid overloading this section. It is to mention also that the equations listed above have requirements that are always verified given the trajectories design.

In order to visually check if the intersections were correctly calculated, a visual test has been implemented to draw the intersection of every couple of different trajectories, and check whether they are accurately located:

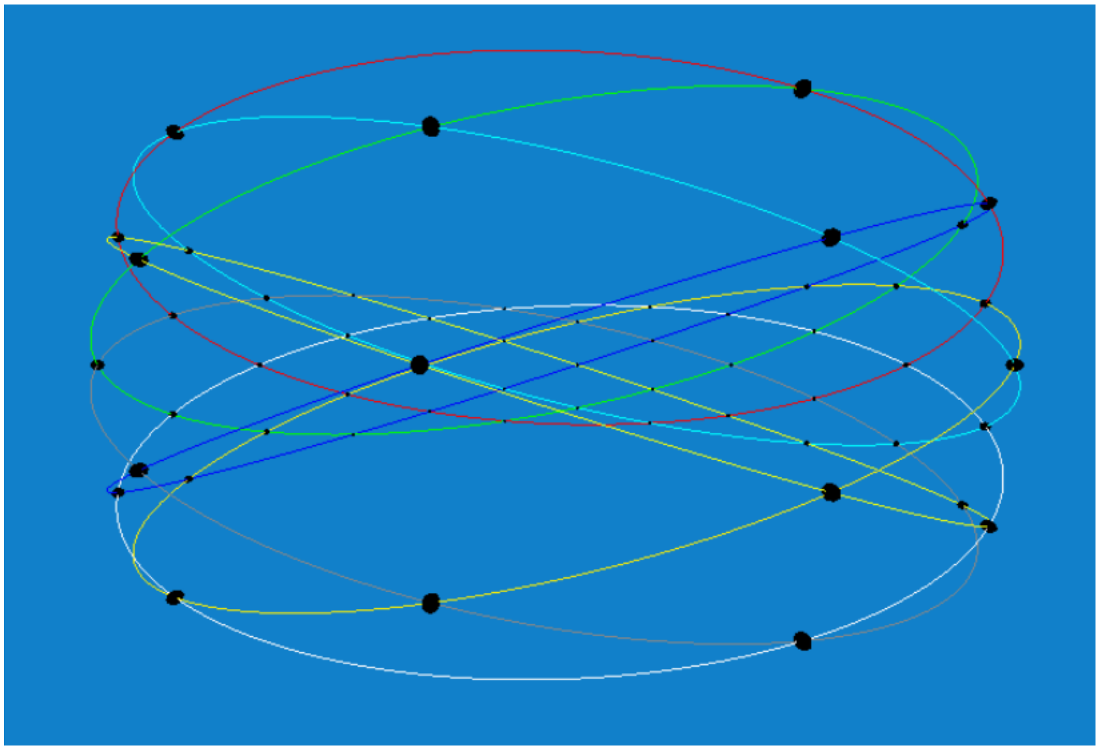

Figure 3.2: Intersections Test

Acute angle between two trajectories (α)

The intersection data model also encapsulates α denoting the acute angle between the two planes carrying the intersecting trajectories; α will be used in section 3.3.

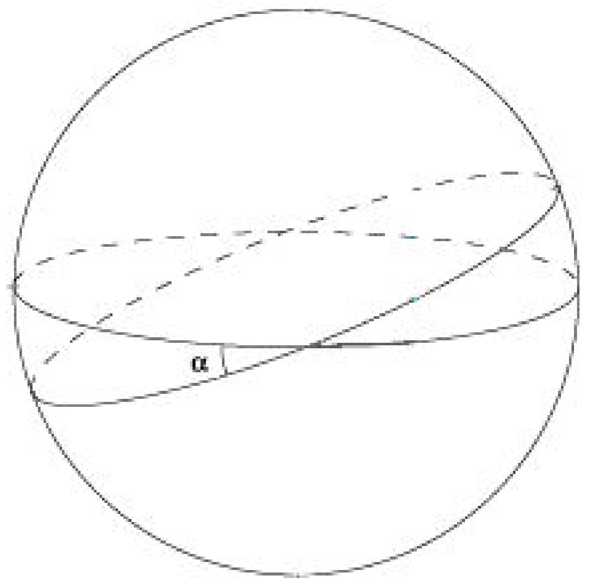

Figure 3.3: Angle between two trajectories (α)

If we consider $\overrightarrow{T_1}$ and $\overrightarrow{T_2}$, as two tangent vectors respectively to both trajectories passing through one of their intersections, the equation to calculate each of both vectors is as follows:

$$\overrightarrow{T_i} = \begin{pmatrix} R\ .\ \sin\left(\theta_i + \frac{\pi}{2}\right) \\ 0 \\ R\ .\ \cos\left(\theta_i + \frac{\pi}{2}\right) \end{pmatrix}_{(\overrightarrow{X_i},\overrightarrow{Y_i},\overrightarrow{Z_i})} .\ CM_i$$

Where CM_i is the conversion matrix from trajectory$_i$'s coordinates system $(\overrightarrow{X_i}, \overrightarrow{Y_i}, \overrightarrow{Z_i})$ to the scene coordinates system $(\overrightarrow{x}, \overrightarrow{y}, \overrightarrow{z})$. CM_i was explained in section 2.1

α can be defined as $(\widehat{\overrightarrow{T_1}\ ,\ \overrightarrow{T_2}})$, which can be calculated by applying the scalar product of both vectors as follows:

$$\alpha = (\widehat{\overrightarrow{T_1}\ ,\ \overrightarrow{T_2}}) = \pm \arccos\left(\frac{\overrightarrow{T_1}\ .\ \overrightarrow{T_2}}{\|\overrightarrow{T_1}\|\ .\ \|\overrightarrow{T_2}\|}\right)$$

Further processing is needed to determine α's sign, but will be omitted to alleviate this section.

Reference to the intersecting fish

The avoidance state loops through the avoiding fish close intersections to predict whether returning to the previous trajectory would result in a collision. The prediction requires access to parameters such as the bounding box of the intersecting fishes, as well as their trajectories and velocities. These parameters are encapsulated under the fish structure, which justifies the need to reference the intersecting fish from an intersection.

Intersections modeled as circular bidirectional linked list nodes

For better performance, the avoidance algorithm needs to efficiently loop over neighbor intersections in both directions (clockwise and counter colockwise) given a close intersection. A bidirectional circular linked list has been adopted for this purpose. Figure 3.4 gives a preview of the relationships between intersections:

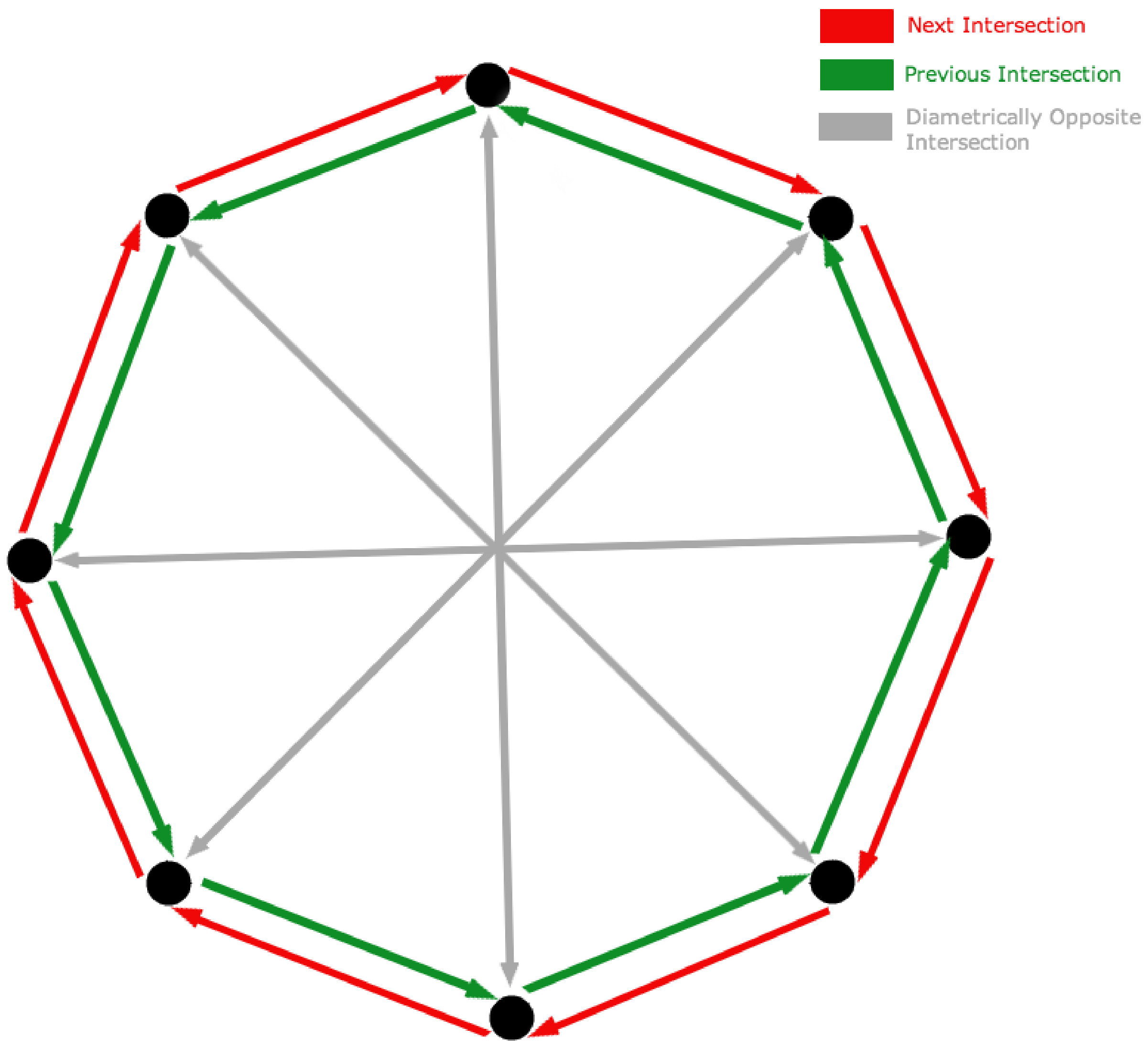

Figure 3.4: Intersections circular bidirectional linked list

Mapping fishes to intersections

The proximity collision prediction algorithm (section 3.2) forwards a couple of fishes to the kinetics based prediction algorithm (section 3.3) whenever they are close enough. The latter algorithm needs an efficient access to the appropriate intersection given a couple fishes. This need can be fulfilled by encapsulating an associative array under the fish structure that maps each intersecting fish's id to the corresponding intersection.

Note that each couple of different trajectories intersect twice in two diametrically opposite points, meaning mapping a fish's id to only one intersection is not enough, which explains the need to reference the diametrically opposite intersection in the intersection structure (see figures 3.1 and 3.4).

3.1.2 Unit grid box

The proximity collision prediction algorithm divides the scene into a network of big grid boxes, which for better performance (see section 3.2.1) should be big enough to carry any fish in the scene while considering a margin. However since the orientation of a fish keeps changing as it moves, the corresponding bounding box edges adapt by changing accordingly. Given no edge can surpass the initial bounding box diameter no matter the orientation, the biggest diameter of all fishes' bounding boxes plus a margin has been adopted as the edge of the unit grid box. The purpose of the margin will be explained under section 3.2.

Note that by the time this document was written, ThreeJS being a new framework, offered only a utility method to calculate a static bounding box of a raw morph regardless of its position, scale or Euler rotations (orientation). Support for those was implemented from scratch.

3.1.3 Avoidance trajectories margin

When a fish avoids a collision it switches the radius of its trajectory to a smaller one. The difference between the two radiuses (Δ_R) needs to be calculated in a way that guarantees fishes swimming within a sphere surface having R as radius, will never collide with fishes swimming within a smaller sphere surface having $(R - \Delta_R)$ as radius (more details on this through chapter 4). For this reason Δ_R needs to match half the sum of the two biggest fishes' thicknesses in the scene. Given a fish's orientation is directed by its relative Z axis, the thickness of a fish cannot exceed the maximum of its bounding box's Y and X edges. Note that the margin treated in this section is different from the margin mentioned in the previous section (3.1.2).

3.2 Proximity collision prediction

The proximity collision prediction is the first layer of the prediction algorithm, it filters fishes that are good candidates for imminent collisions and forwards them to the kinetics based algorithm for further processing. The main purpose of using the proximity prediction is to minimize invoking the kinetics based algorithm as possible given its complexity (see section 3.3).

If we consider a three dimensional grid of big boxes serving as a large scaled coordinate system, the idea is to perform the following operations whenever a movement listener gets notified by a change of a fish's location:

- Calculate the bounding box of the fish, considering the new position, the scale, and Euler rotations.
- Add margins to the new bounding box volume. The margins serve as the security distance that separates two fishes, lower than which the kinetics based prediction algorithm gets invoked.
- Convert the position and dimensions of the bounding box to the large scaled coordinate system; let's denote the result "normalized bounding box"; given the large scaled coordinate system unit is big enough to carry the biggest fish in the scene with a margin, the normalized bounding box volume will never exceed a maximum of 8 unit boxes (i.e. when a fish overlaps a corner of a normalized unit box which implies overlapping the 8 normalized unit boxes sharing that corner), but can be a minimum of only one unit box. A performance test has been recorded in this regard (see section 3.2.1).
- Use an associative array to determine which fishes are sharing unit boxes as follows:
  - Convert each unit box coordinates to a string key.
  - Check if the key is already mapping to previously processed fishes. If so a grid box population event gets fired, thereby triggering the kinetics based prediction algorithm. Otherwise a new entry gets added to the associative array.
  - Each new frame the associative array gets cleared, and re-populated from scratch.

This logic will prevent applying the kinetics based prediction algorithm on fairly distant fishes. Each fish will just need to check for collisions only with fishes that share one or more of its populated unit boxes.

Note that a caching mechanism has been placed to prevent checking collisions multiple times for couples of fishes sharing more than one unit box.

3.2.1 Performance

A test has been run for a duration of 5 minutes with 11 fishes in the scene to record some key performance indicators and prove the efficiency of the proximity collision prediction algorithm compared to simply using a nested loop.

First record

Table 3.1 presents records of how often in average a unit box gets populated by a number of fishes.

Number of fishes per unit box	**Frequency (%)**
1	85.74
2	13.21
3	0.91
4	0.09
5	0.044

Table 3.1: Performance - Number of fishes per normalized unit box

Given the first fish that populates a unit box never checks for collisions, we obtain in average $(1 \times 0.13 + 2 \times 0.0091 + 3 \times 0.0009 + 4 \times 0.00044) \approx$ **0.155** collision verification per unit box per frame.

Second record

Depending on its position, size and rotations, a fish may populate 1 to 8 normalized unit boxes. Table 3.2 presents how often in average a fish populates a number of unit boxes.

Number of unit boxes populated by a fish	Frequency (%)
1	29.6
2	47.85
4	20.08
8	2.46

Table 3.2: Performance - Number of normalized unit boxes populated by a fish

Which gives an average of $(1 \times 0.296 + 2 \times 0.4785 + 4 \times 0.2008 + 8 \times 0.0246) \approx$ **2.253** unit boxes populated per fish per frame.

Conclusion

Each frame, one fish populates 2.253 unit boxes, from which only 15.5% trigger collision verifications. Given the test was run with 11 fishes in the scene, we obtain in average $(11 \times 2.253 \times 0.155) \approx$ **3.84** collision verifications per frame, compared to $(\sum_{i=2}^{11} i) =$ **65** collision verification if a nested loop checking for every couple fishes has been adopted.

3.2.2 Appropriate states for avoidance

When a fish avoids a collision, it reduces the radius of its circular trajectory leaving enough margin to prevent any overlapping. However in order to keep the movement look natural, the avoiding fish has to follow a link trajectory towards the smaller radius trajectory. A fish's state is appropriate for avoidance when it's either in the swimming state, or in the avoiding state but not in a link trajectory (see chapter 4 for more details).

The states Dying, Reviving, Dead, and Resizing are not appropriate for avoidance. When a fish is in one of these states (which is very rare) collisions may happen.

When a fish's state is not appropriate for avoidance it won't be placed in the grid, and thus won't be considered by any of both collision prediction algorithms.

3.3 Collision prediction based on kinetics

The algorithm in this section predicts accurately whether a couple of close fishes are willing to collide based on kinetics. In order to simplify things, a couple of intersecting trajectories' arcs get projected into a 2D plane. The projection doesn't perfectly match the motion in 3D, but approximates accurately enough to predict any collision before it happens with a comfortable time and space margin. Figure 3.5 illustrates the projection:

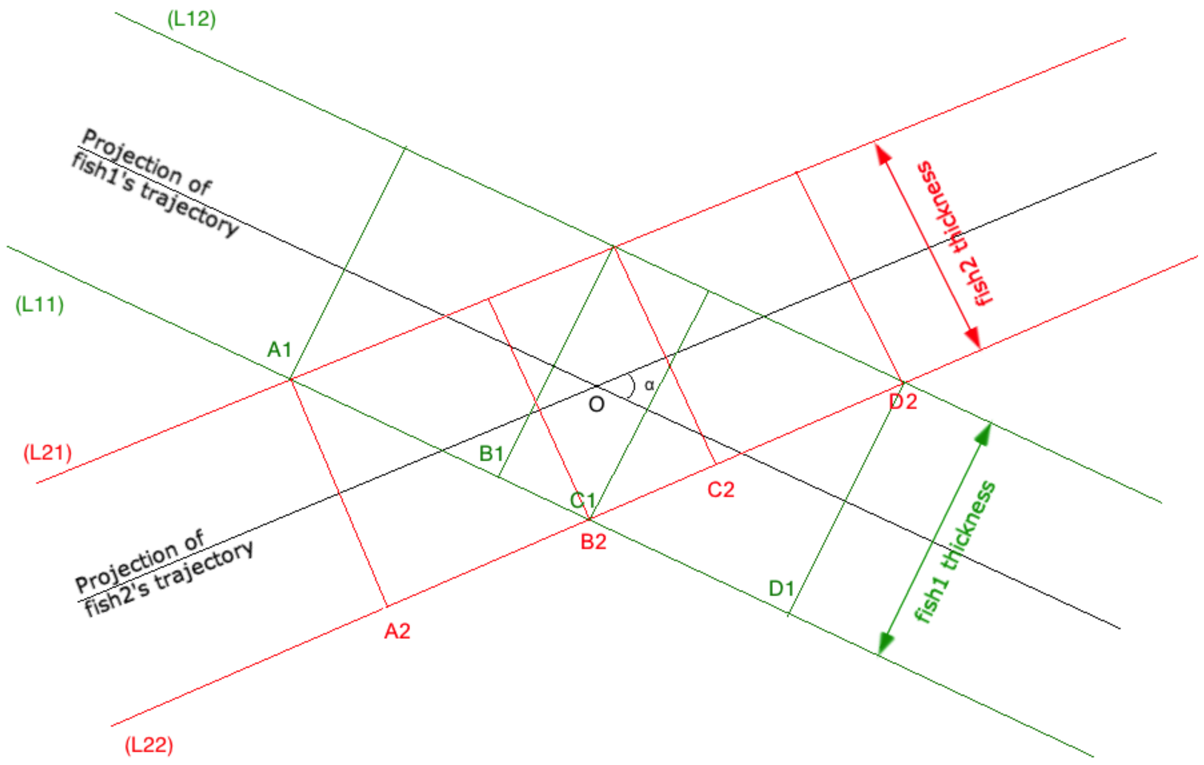

Figure 3.5: 2D projection - same key positions order for both fishes

- Given that a fish's orientation is directed by its relative Z axis, the thickness of a fish cannot exceed the maximum of its bounding box's Y and X edges.
- α is the angle between the two planes carrying both trajectories (see sub-section 3.1.1).

- (L_{ij}) represent the borders delimiting the area within which fish$_i$ is swimming.
- A_i, B_i, C_i and D_i are key locations for the kinetics based algorithm (see section 3.3.2). They can be defined as follows:
  - $A_i = (L_{i1}) \cap (L_{\bar{i}1})$, with $\bar{i} = 2$ if (i = 1), otherwise $\bar{i} = 1$.
  - $B_i = (L_{i2}) \cap (L_{\bar{i}1})$.
  - $C_i = (L_{i1}) \cap (L_{\bar{i}2})$.
  - $D_i = (L_{i2}) \cap (L_{\bar{i}2})$.

Note that in the definitions above, the key locations have been defined as points; later on, they should be thought of as the orthogonal lines to the borders passing through these points. That said, if through what follows the notation A_iB_i for instance gets used, it should be thought of as the distance between two parallel lines rather than two points. These lines will be located by their curvilinear abscissas within their corresponding trajectories.

Note also that in figure 3.5 the order of the key positions is the same for both fishes, however when their thicknesses' difference is big enough, the order changes. Figure 3.6 shows that for fish$_1$ (less thick) the key positions order is A$_1$, B$_1$, C$_1$ then D$_1$, whereas for fish$_2$ (thicker) the order is A$_2$, C$_2$, B$_2$ then D$_2$.

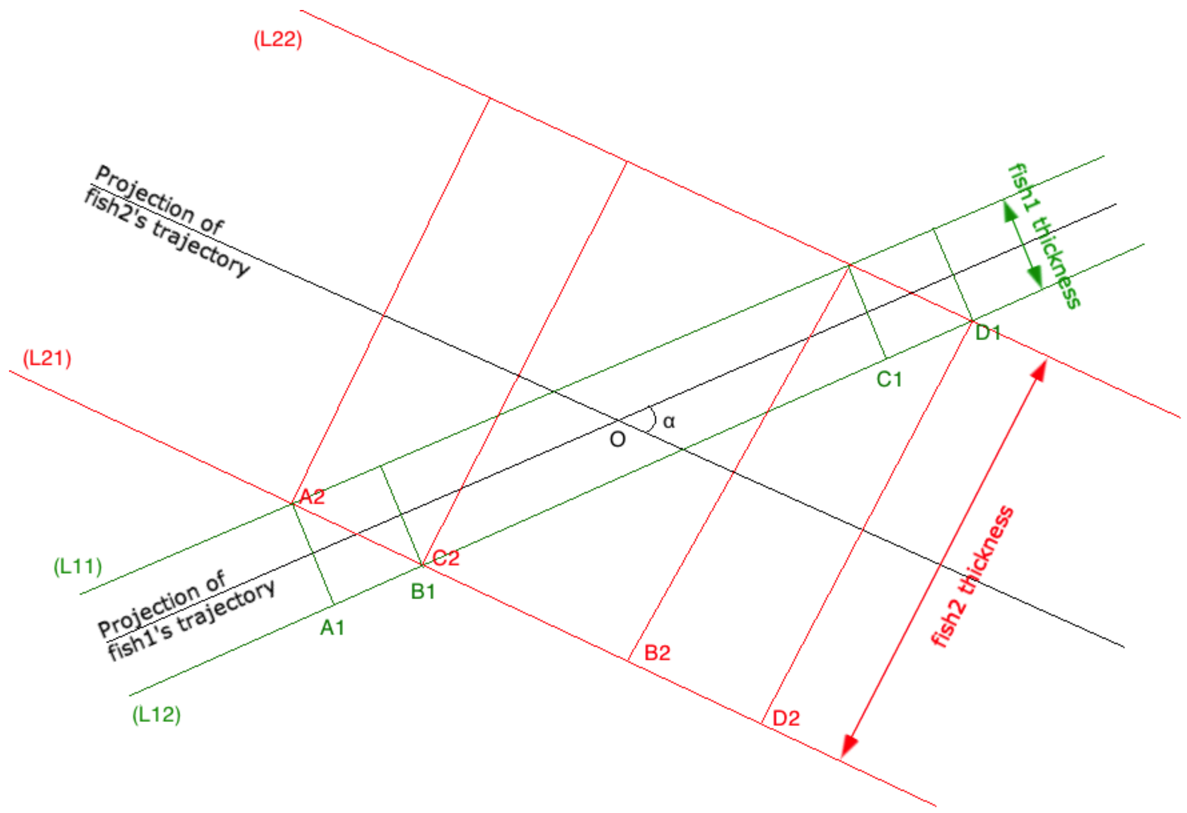

Figure 3.6: 2D projection - different key positions order per fish

The key locations order is important when a fish is trying to return from an avoidance trajectory, and predicting whether its return would result in a collision (more details in chapter 4).

3.3.1 Locating the key positions

Following are the mathematical formulas required to compute the key locations:

$$A_iB_i = \left|\frac{thickness_i}{\tan(\alpha)}\right| \quad , \quad A_iC_i = \left|\frac{thickness_{\bar{i}}}{\sin(\alpha)}\right| \quad , \quad A_iD_i = A_iB_i + A_iC_i$$

$$OA_i = \frac{A_iD_i}{2} \quad , \quad OB_i = |OA_i - A_iB_i| \quad , \quad \overline{OD_i} = -\overline{OA_i} \quad , \quad \overline{OC_i} = -\overline{OB_i}$$

$$sign(\overline{OA_i}) = sign(\delta_i); where\, \delta_i = (\theta_{Y_i} - \theta_i)[2\pi]; with\, \delta_i \in]-\pi, \pi]$$

$$sign(\overline{OB_i}) = sign(\overline{OA_i}) \times sign(OA_i - A_iB_i)$$

Note that θ_{Y_i} is the curvilinear abscissa of fish$_i$ within its trajectory, while θ_i is the curvilinear abscissa of the intersection O relatively to fish$_i$'s trajectory. All θ_i's have been calculated and cached during the collision prevention setup. That said, the curvilinear abscissas for the key positions relatively to the corresponding trajectory can be obtained as follows:

$$\theta_{A_i} = \frac{\overline{OA_i}}{R} + \theta_{Y_i} \quad , \quad \theta_{B_i} = \frac{\overline{OB_i}}{R} + \theta_{Y_i} \quad , \quad \theta_{C_i} = \frac{\overline{OC_i}}{R} + \theta_{Y_i} \quad , \quad \theta_{D_i} = \frac{\overline{OD_i}}{R} + \theta_{Y_i}$$

Where R denotes the radius of the circular trajectory.

3.3.2 Applying kinetics to the two dimensional projection

Based on the 2D projection, the logic applied to predict a collision, depends on many scenarios. Figure 3.7 illustrates all the possible scenarios:

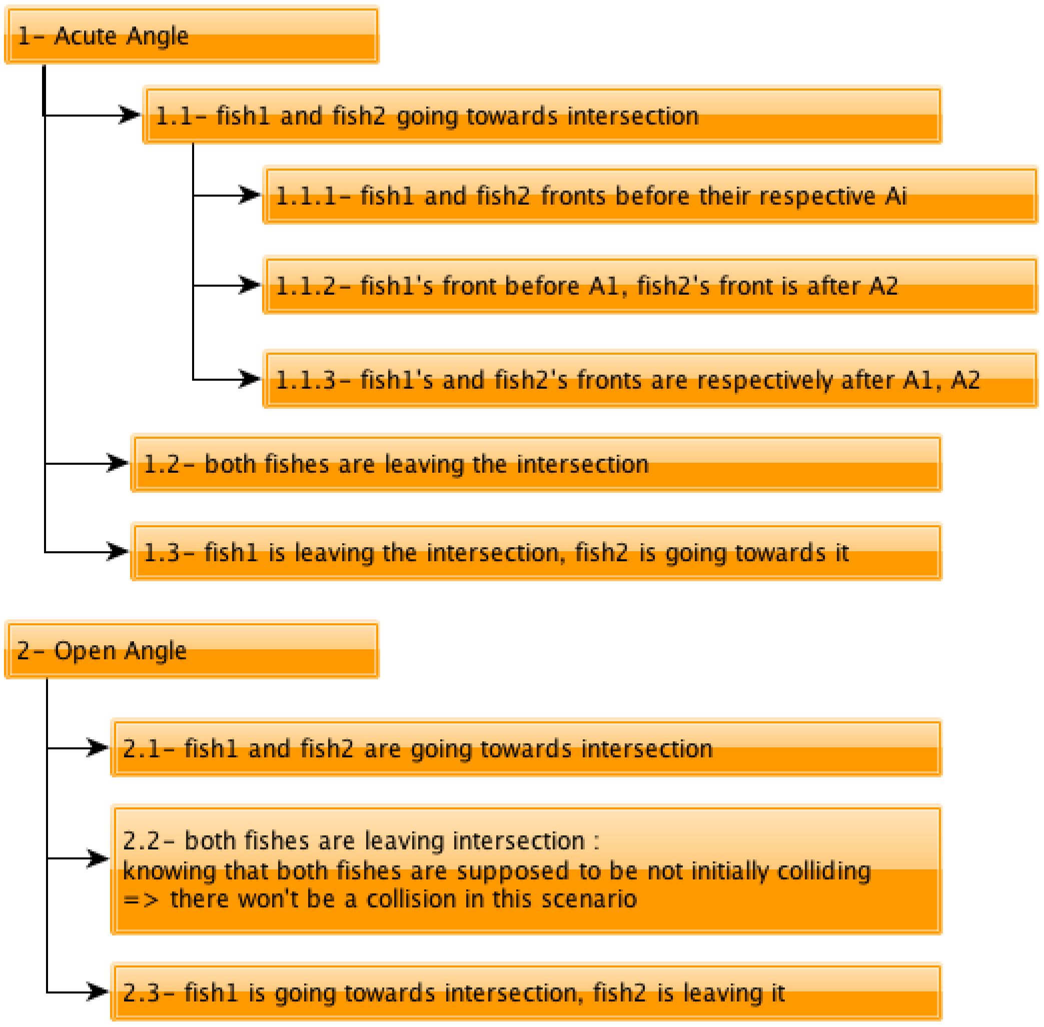
1- Acute Angle
1.1- fish1 and fish2 going towards intersection
1.1.1- fish1 and fish2 fronts before their respective Ai
1.1.2- fish1's front before A1, fish2's front is after A2
1.1.3- fish1's and fish2's fronts are respectively after A1, A2
1.2- both fishes are leaving the intersection
1.3- fish1 is leaving the intersection, fish2 is going towards it
2- Open Angle
2.1- fish1 and fish2 are going towards intersection
2.2- both fishes are leaving intersection :
knowing that both fishes are supposed to be not initially colliding
=> there won't be a collision in this scenario
2.3- fish1 is going towards intersection, fish2 is leaving it

Figure 3.7: Collision prediction scenarios

Where the angle being considered as acute or open refers to $\widehat{A_1OA_2}$, which is either α or $(\pi - \alpha)$ depending on the fishes positions. Note that α is the angle between both planes carrying the trajectories, it was calculated and cached during the collision prevention setup.

The rest of this section will go through the first case 1.1.1, which should be sufficient to inspire processing the rest of the cases. Figure 3.8 presents case 1.1.1:

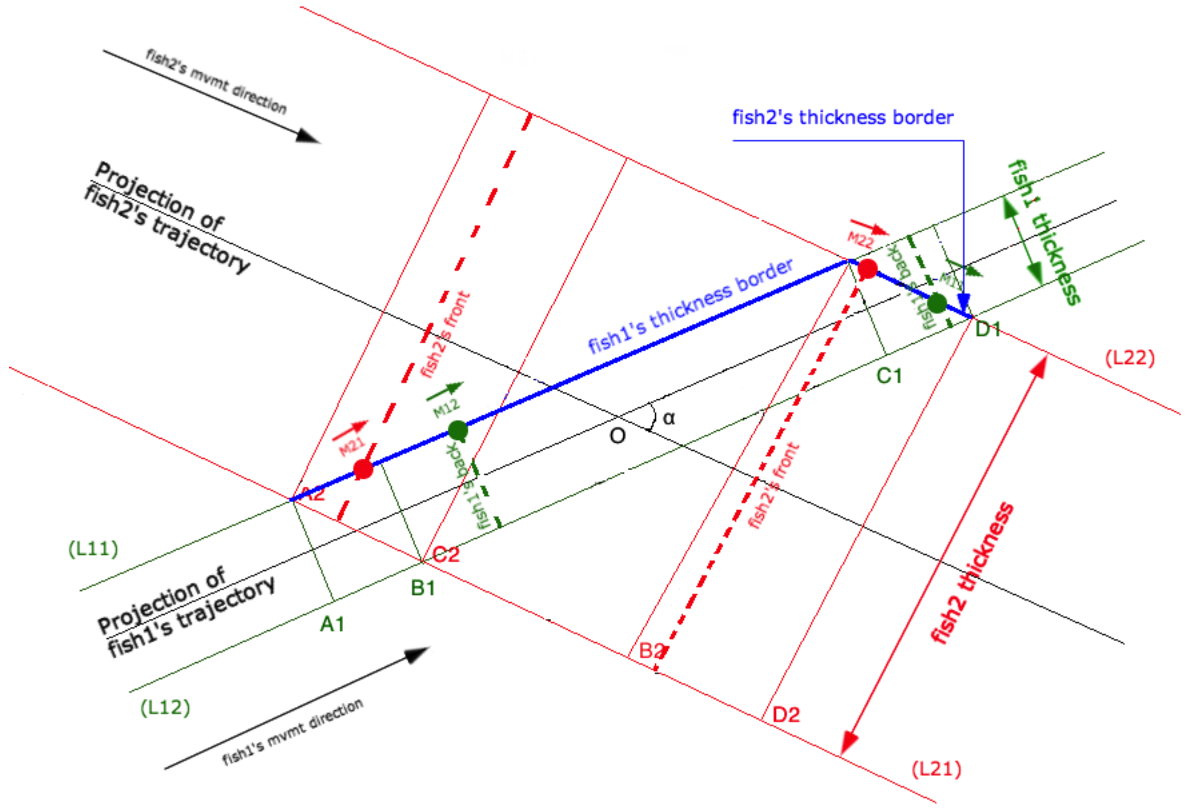

Figure 3.8: Acute angle, both fishes going towards intersection, both fishes fronts are before A_1, A_2

The critical area for each fish is delimited by A_i and D_i, meaning a collision between both is possible (but not certain) only when both are inside their critical areas. The required steps to accurately predict a collision are the following:

First step

Determine which fish will reach A_i first. Note that for better accuracy in the estimations, the fish's front position will be considered instead of simply its position. If we consider T_1 and T_2 as the necessary durations for fish$_1$ and fish$_2$ to arrive respectively to A$_1$, A$_2$, the fish arriving in a minimum duration will be the first. T$_1$ and T$_2$ can be calculated as follows:

$$T_1 = \frac{|\theta_{F1} - \theta_{A1}|}{|\delta_{Y1}|} \quad , \quad T_2 = \frac{|\theta_{F2} - \theta_{A2}|}{|\delta_{Y2}|}$$

Where θ_{Fi} is the curvilinear abscissa of the corresponding fish's front, θ_{Ai} is the curvilinear abscissa of A_i and δ_{Yi} is the angular move step per frame representing the angular velocity. Let's call fish$_1$ the fish that will arrive first.

Second step

Guarantee that fish$_1$'s back will also arrive to A$_1$ before fish$_2$'s front reaches A$_2$. This could be investigated by applying the same logic described above.

Third step

Guarantee that fish$_1$'s back will keep its position in front of fish$_2$'s front, all along the critical area, otherwise a collision may happen. Notice that from A$_1$ to C$_1$ for fish$_1$ (A$_2$ to B$_2$ for fish$_2$) if a collision would happen, the first contact point would be located within fish$_1$'s thickness border (the blue highlighted line), whereas from C$_1$ to D$_1$ for fish$_1$ (B$_2$ to D$_2$ for fish$_2$) the first contact point would be located within fish$_2$'s thickness border. In other words, if we define the following mobile points:

- $M_{12} = (fish1's\ back) \cap (L_{11})$; where $M_{12} \in [A_1C_1]$.
- $M_{21} = (fish2's\ front) \cap (L_{11})$; where $M_{21} \in [A_2B_2]$.
- $M_{11} = (fish1's\ back) \cap (L_{22})$; where $M_{11} \in [C_1D_1]$.
- $M_{22} = (fish2's\ front) \cap (L_{22})$; where $M_{22} \in [B_2D_2]$.

It is obvious to notice that while M$_{12}$ has the same speed as fish$_1$, M$_{11}$ is faster since fish2's thickness border (highlighted in blue) is longer than C_1D_1. Following the same logic for fish$_2$, M$_{21}$ is also faster than M$_{22}$. With that in mind, it is important to assert two facts to ensure a collision won't happen within the critical area:

- First guarantee that M$_{21}$ won't catch M$_{12}$ all along fish1's thickness border. This could be investigated by checking which of both fishes will arrive first to C$_1$, respectively B$_2$.

- Second guarantee that M_{22} won't catch M_{11} all along fish2's thickness border. This could be investigated by checking which of both fishes will arrive first to D_i.

The same logic used in the first step can be applied here to investigate both of the above propositions.

Chapter 4

Collision Avoidance

The collision avoidance algorithm gets triggered when an imminent collision is predicted. Given all the fishes are swimming within the same sphere surface, avoiding a fish without hurting another is guaranteed if the avoiding fish leaves its trajectory's sphere surface to a smaller one (exceptions are treated under section 4.2). The margin between both spheres gets calculated only once at load time (see section 3.1.3).

Note that the trajectory's radius gets reduced when avoiding, but never extended, otherwise the fish will be lost of the camera's sight. However in order to keep the fishes at the same distance from the camera, making it easier to distinguish their sizes and their velocities differences, changing the trajectory's radius should be temporary; as soon as the collision is avoided and the way back is clear, the avoiding fish should return back to its original trajectory.

4.1 Avoidance phases

The deviation from the initial trajectory should be smooth by keeping the movement look continuous and natural. To that end, curvilinear paths towards the target trajectory have been designed as shown in figure 4.1:

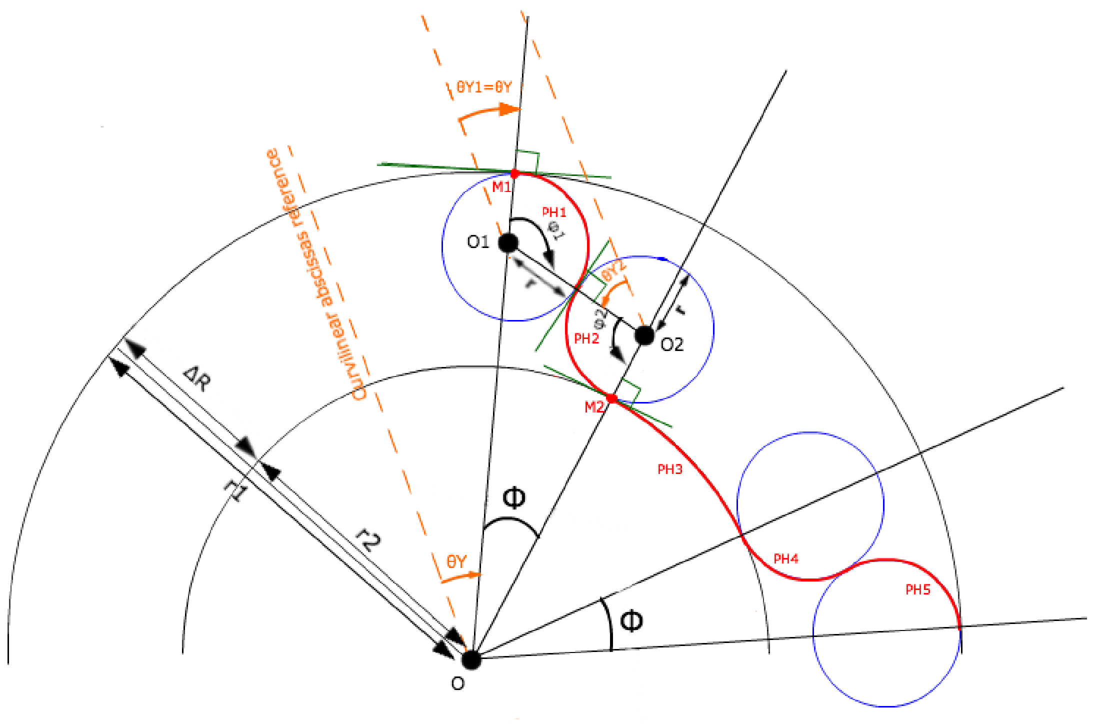

Figure 4.1: Avoidance link trajectories

The followed path starting from the moment an avoiding fish deviates from its initial trajectory until it returns back, is highlighted in red. It consists of five continuous adjacent arcs, each denoting a separate phase in the avoiding state.

Phase 1 and Phase 2

Phase 1 and phase 2 denote the curvilinear link path followed by an avoiding fish towards the target avoidance trajectory. As shown in figure 4.1, the link path is designed as a combination of a couple adjacent circles' arcs, with the goal of making the transition look smooth and natural. Both circles have the same radius r and are respectively adjacent to the start and target trajectories. Both arcs are calculated given the following input:

- r_1 and r_2 denoting the respective radiuses for the start and target trajectories. Note that $r_2 = r_1 - \Delta R$, where ΔR is the safe margin that has been calculated and cached at load time (see section 3.1.3).

- θ_Y denoting the curvilinear abscissa of the avoiding fish within its swimming trajectory just before it deviates to avoid (see figure 4.1).
- M_1 denoting the position of the avoiding fish just before it deviates to avoid (see figure 4.1).
- $\Phi = (\overrightarrow{OM_1}, \overrightarrow{OM_2})$ where M_2 is the contact point between phase 2 and 3 arcs (see figure 4.1). Φ is a configurable constant parameter.

The results return a couple arcs defined by the following parameters:

- A radius r (see figure 4.1). r is a solution of the second degree polynomial equation $ar^2 + 2br + c = 0$ where:

$$\begin{cases} a = 2\ (1 - cos\Phi) \\ b = (1 + cos\Phi)\ (r_1 - r_2) \\ c = 2\ r_1\ r_2\ cos\Phi - r_1^2 - r_2^2 \end{cases} \qquad \Rightarrow r = \frac{-b + \sqrt{b^2 - a\ c}}{a}$$

- O_i the center of the circle carrying the arc (see figure 4.1). Given O is the center of the scene, O_1 coordinates can be inferred from the following formula:

$$\overrightarrow{OO_1} = \frac{OM_1 - r}{OM_1}\ .\ \overrightarrow{OM_1}$$

Inspired by the same logic O_2 coordinates can be inferred from the following formula:

$$\overrightarrow{OO_2} = \frac{OM_2 + r}{OM_2}\ .\ \overrightarrow{OM_2}$$

with :

$$M_2 = \begin{pmatrix} r\ .\ sin(\theta_Y + \Phi) \\ 0 \\ r\ .\ cos(\theta_Y + \Phi) \end{pmatrix}_{(\overrightarrow{X}, \overrightarrow{Y}, \overrightarrow{Z})} \qquad .\ CM$$

where $(\overrightarrow{X}, \overrightarrow{Y}, \overrightarrow{Z})$ denote the avoiding fish relative rotated axises, and CM being the conversion matrix calculated in section 2.1.

- A subtending angle φ_i (see figure 4.1). φ_1 and φ_2 formulas are the following:

  $$\varphi_1 = arccos\left(\frac{\overrightarrow{O_1M_1} \,.\, \overrightarrow{O_1O_2}}{O_1M_1 \,.\, O_1O_2}\right) \quad ; \quad \varphi_2 = arccos\left(\frac{\overrightarrow{O_2O_1} \,.\, \overrightarrow{O_2M_2}}{O_2O_1 \,.\, O_2M_2}\right)$$

- An offset curvilinear abscissa θ_{Yi} (see figure 4.1). θ_{Y1} and θ_{Y2} formulas are the following:

  $$\theta_{Y1} = \theta_Y \quad ; \quad \theta_{Y2} = \theta_Y + \Phi - \pi - \varphi_2$$

The formulas for phase 4 and 5 arcs will be omitted as they can be easily inspired from the logic above.

Note that when following any of the transition paths (i.e. any of phase 1, 2, 4 or 5 arcs), an avoiding fish velocity increases so it avoids the collision fast, which breaks the velocity/response time proportion rule; this should not be a big of an issue as the duration of the transition phases is very short.

Phase 3

Phase 3 starts when the avoiding fish arrives to the target avoidance trajectory keeping it safe from collisions.

Given phase 3 arc has a smaller radius r_2 than r_1, the angular velocity δ_Y should be increased accordingly to maintain the same speed; this can be easily explained by the fact that a turn around a circle having a small radius is executed faster than a turn around a larger circle if the same velocity is maintained; the formula to adapt θ_Y with the avoidance trajectory is the following:

$$\delta_{Y2} \;=\; \frac{r_1}{r_2}\,\delta_{Y1}$$

where δ_{Y1} and δ_{Y2} denote respectively the old and new angular velocities.

An avoiding fish in phase 3 keeps an eye on its initial trajectory so that as soon as the way back is clear, it safely returns back thus triggering phase 4 followed by phase 5.

Phase 4 and phase 5

Through phase 4 and 5, an avoiding fish returns back to its initial trajectory following the same logic adopted in phase 1 and 2.

4.2 Recursive avoidance

The scenario illustrated by figure 4.1 is the most common; However it has exceptions. In the case where 2 avoiding fishes are waiting for their way back to be cleared and an imminent collision between both gets predicted the faster fish will avoid again by going through phase 1 followed by phase 2 towards an even smaller radius trajectory. This logic keeps getting applied recursively whenever necessary, until the avoiding fish eventually reaches a trajectory with a minimum radius, smaller than which the fish would collide with the objects in the center of the scene. If this happens, the involved fishes will no longer predict collisions and will just keep checking for their way back to get cleared. This is the only scenario where collisions may occur, but it is very rare, and there must be a busy traffic in the scene (over 20 fishes) so that it happens.

The following diagram describes with more details how an avoiding fish switches between phases:

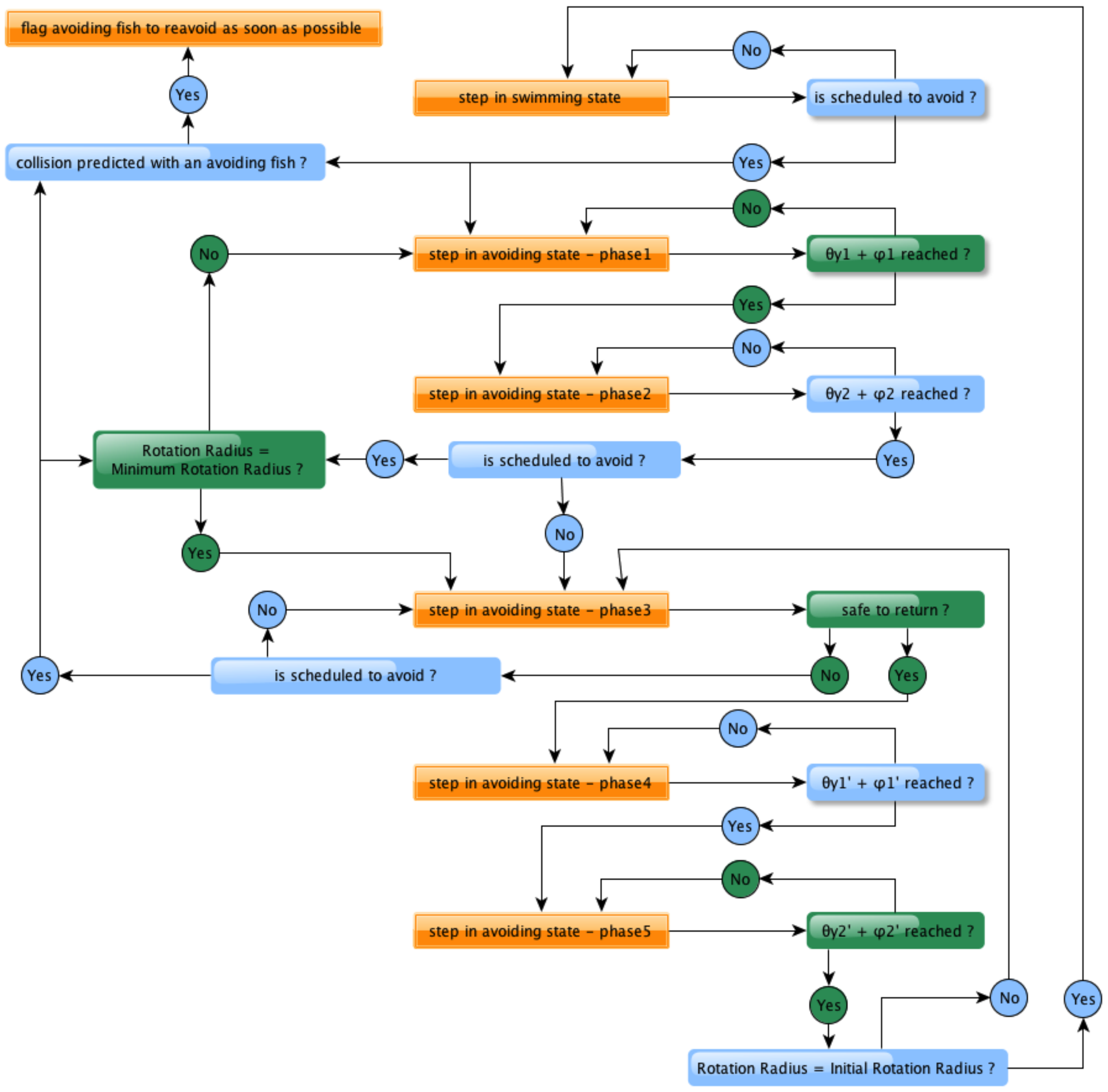
flag avoiding fish to reavoid as soon as possible
Yes
step in swimming state
No
is scheduled to avoid ?
collision predicted with an avoiding fish ?
Yes
No
No
step in avoiding state - phase1
θy1 + φ1 reached ?
Yes
No
step in avoiding state - phase2
θy2 + φ2 reached ?
Rotation Radius = Minimum Rotation Radius ?
Yes
is scheduled to avoid ?
Yes
No
Yes
No
step in avoiding state - phase3
safe to return ?
Yes
is scheduled to avoid ?
No
Yes
No
step in avoiding state - phase4
θy1' + φ1' reached ?
Yes
No
step in avoiding state - phase5
θy2' + φ2' reached ?
Yes
No
Yes
Rotation Radius = Initial Rotation Radius ?

Figure 4.2: Avoidance phases diagram

.

The color code used in the diagram is the following:

- Orange represents actions.

- Cyan and green represent boolean propositions linked to their eventual values. There is no semantic difference between cyan and green; the simple reason behind using two different colors is to highlight the relationship between propositions and their values given the diagram is relatively dense, thus making it easier to read.

The rest of this section will explain some terms used in the diagram; self expressive terms will be omitted.

Is scheduled to avoid?

In the beginning of this chapter, it was briefly stated that the collision prediction algorithm filters the collision candidates and forwards them to the avoidance algorithm without much details. In fact whenever a collision is predicted the involved fishes get pushed to a stack of collision candidates. At the end of the frame the stack gets processed to schedule the faster of each couple conflicting fishes to the avoidance state in the next frame. The slower fish will keep its state.

Flag avoiding fish to reavoid as soon as possible

Whenever a fish predicts a collision and gets scheduled to avoid in the next frame, just before it switches to phase 1, it estimates whether its path to the avoidance trajectory would result in a collision with any other already avoiding fish, if so the latter gets flagged to re-avoid as soon as possible. If the conflicting fish is on phase 1 or 2, it will re-avoid again as soon as it reaches the target trajectory, otherwise (conflicting fish on phase 3) it will re-avoid immediately in the next frame.

$\theta_{yi} + \varphi_i$ reached?

As explained in section 4.1, the link between the initial and target trajectories is made of two adjacent arcs, each having a start angle (θ_{yi}), and a deviation angle (φ_i) defining the arc's boundaries, which means reaching ($\theta_{yi}+\varphi_i$) ends the corresponding arc thereby triggering a switch of phase.

Safe to return?

A fish maintains its state on phase 3 as long as the way back is occupied. Once it gets cleared (safe to return), the fish immediately switches to phase

4 towards its previous trajectory.