

**Bachelor Project**



**Czech  
Technical  
University  
in Prague**

**F3**

**Faculty of Electrical Engineering  
Department of Computer Graphics and Interaction**

## **Interactive Installation with Underwater Flocking Animals for Children**

**Filip Jeřowicz**

**Supervisor: Ing. Uršula Žáková  
June 2023**



## I. Personal and study details

Student's name: **Jeřowicz Filip** Personal ID number: **503178**  
Faculty / Institute: **Faculty of Electrical Engineering**  
Department / Institute: **Department of Computer Graphics and Interaction**  
Study program: **Open Informatics**  
Specialisation: **Computer Games and Graphics**

## II. Bachelor's thesis details

Bachelor's thesis title in English:

**Interactive Installation with Underwater Flocking Animals for Children**

Bachelor's thesis title in Czech:

**Interaktivní Instalace s Podmořskými Flokujícími Zvířaty pro Děti**

Guidelines:

Get acquainted with the theory of real-time and precomputed flocking algorithms in the gaming industry, simulation and animation. Explore existing solutions for animal flocking in game development, mainly for Unity and Unreal Engine. Analyze flocking behaviour in nature and compare it to existing solutions in game development. Propose five different ways to make an interactive screen installation for museums or galleries. Choose one and design a multiuser interactive screen with underwater life suitable for entertaining children. In the final implementation, at least two life forms should exhibit flocking behaviour. The Flocking algorithm should take advantage of Unity's ECS framework to optimize for performance. Compare the computational demand of flocking implementation based on GameObject and ECS frameworks. Test the interactive installation with a group of children (at least three) of your choice.

Bibliography / sources:

- [1] Satz, Helmut. (2020). The Rules of the Flock: Self-Organization and Swarm Structure in Animal Societies. 10.1093/oso/9780198853398.001.0001.  
[2] Craig W. Reynolds. (1987). Flocks, herds and schools: A distributed behavioral model. In Proceedings of the 14th annual conference on Computer graphics and interactive techniques (SIGGRAPH '87). Association for Computing Machinery, New York, NY, USA, 25–34. <https://doi.org/10.1145/37401.37406>

Name and workplace of bachelor's thesis supervisor:

**Ing. Uršula Žáková Department of Computer Graphics and Interaction FEE**

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **16.02.2023** Deadline for bachelor thesis submission: **26.05.2023**

Assignment valid until: **22.09.2024**

Ing. Uršula Žáková  
Supervisor's signature

Head of department's signature

prof. Mgr. Petr Páta, Ph.D.  
Dean's signature

## III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

\_\_\_\_\_  
Date of assignment receipt

\_\_\_\_\_  
Student's signature



## Acknowledgements

I would like to thank Unity for its effort to improve the stability and extend the functionalities of the Data-Oriented stack technology. Huge thanks belong to my supervisor Ing. Uršula Žáková for the guidance and advice she gave me during my work. My gratitude belongs to Ing. Ondřej Slabý for the calibration of the scene cameras and his entire team for the amazing work they have done on the interactive wall with built-in motion-capture functionalities.

And lastly, I would like to thank my parents and my whole family for their relentless support. It means the world to me.

## Declaration

I hereby confirm that this thesis and the work presented in it is entirely my own. Where I have consulted the work of others this is always clearly stated. All statements taken literally from other writings or referred to by analogy are marked and the source is always given.

In Prague, 20 May 2023

Signature: .....

## Abstract

The main purpose of this work is to get acquainted with the phenomenon of animal flocking observed in nature and the creation of an interactive underwater simulation for children using Unity's *Data-Oriented Technology Stack (Dots)* with *Entity Component System (ECS)*. Available solutions for animal flocking that can be found on the Unity asset store and Unreal Market are examined as well. A modified version of Reynold's Boids algorithm is used for the purposes of the simulation and a comparison between the *Object Oriented Programming (OOP)* approach and implementation that leverages performance using the *ECS* is provided. For the purposes of the interaction, an Interactive wall with motion-capture functionalities developed at Czech Technical University in Prague was used. The simulation was tested with a group that consisted of both children and adults. It was concluded that the performance gained by the usage of the *ECS* approach instead of a classic *OOP* approach is significant and opens new possibilities in real-time simulation.

**Keywords:** Entity Component System, Data-Oriented technology stack, Boids, Interactive simulation

**Supervisor:** Ing. Uršula Žáková  
Karlovo náměstí 13,  
E-423.  
12000 Praha 2

## Abstrakt

Hlavním cílem této práce je seznámit se s fenoménem flokování zvířat v přírodě a vytvoření interaktivní podvodní simulace pro děti za pomoci *Datově orientované technologie (Dots)* dostupné v herním enginu Unity. Dostupná řešení pro flokování zvířat, která jsou k dostání na Unity asset storu a Unreal Marketplace jsou taktéž prozkoumána. Upravená verze Reynoldsova Boids algoritmu je použita pro potřeby simulace a porovnání mezi řešením využívajícím *objektově orientované programování (OOP)* a *Entity Component System (ECS)* alternativu, která značně zvyšuje výkon simulace, je taktéž součástí této práce. Za účelem interakce byla použita Interaktivní stěna vyvíjena na Českém Vysokém Učení Technickém v Praze. Simulace byla otestována na skupině, která se skládala z dětí i dospělých. Na základě porovnání flokovacího algoritmu za použití technologie *ECS* s *OOP* přístupem jsem došel k závěru, že použitím *ECS* přístupu lze dosáhnout značného zvýšení ve výkonu a zároveň *ECS* alternativa otevírá nové možnosti pro simulace v reálném čase.

**Klíčová slova:** Entity Component System, Data-Oriented technology stack, Boids, Interactive simulation

**Překlad názvu:** Interaktivní aplikace s Podmořskými Flokujícími Zvířaty pro Děti

# Contents

<b>1 Introduction</b>	<b>1</b>		
<b>2 Getting acquainted with the phenomenon</b>	<b>3</b>		
2.1 Why do they flock	3		
2.1.1 Migrating birds	3		
2.1.2 Ants	4		
2.1.3 Bees	4		
2.1.4 Answer	4		
2.1.5 A Mystery	5		
2.2 Basic Concepts Of Animal Flocking	5		
2.2.1 Behavior relation to atoms of iron	5		
2.2.2 Criticality	6		
2.2.3 Self-organized criticality	7		
2.2.4 Emergent behavior	8		
<b>3 Pre-implementation part</b>	<b>11</b>		
3.1 Real-time vs. pre-computed algorithms	11		
3.2 Flocking in computer games vs science simulations	11		
3.3 Basic animal flocking algorithms	13		
3.3.1 Boids	13		
3.3.2 Self-driven particles	15		
3.3.3 Density induced transition in the school of fish	16		
3.3.4 Wolf-like behavior	17		
3.4 Available assets	19		
3.5 Suitable space partitioning structures	20		
3.5.1 Voronoi neighborhoods	20		
3.5.2 K-D Tree	22		
3.5.3 Cube space partition map	23		
<b>4 Implementation</b>	<b>25</b>		
4.1 Functionalities	25		
4.1.1 Obstacle Avoidance	25		
4.1.2 Boids flocking behavior	26		
4.1.3 Other optimizations	28		
4.2 Entities, Components and Systems terminology	28		
4.2.1 Basics	28		
4.2.2 Aspects and Authoring scripts	30		
4.2.3 Terminology Sum-up	30		
4.3 Dealing with the new technology and an error-prone code	30		
4.3.1 Decomposition of the operations	30		
4.3.2 File management	31		
4.3.3 Actively search for help	31		
4.4 Interactive screen options	32		
4.4.1 Multiple-touch screen tap-like interaction	32		
4.4.2 Speech recognition interaction	32		
4.4.3 Motion-capture based interaction	33		
4.4.4 Remote phone interaction	33		
4.4.5 Holographic projection	33		
4.4.6 User interaction limitations	34		
4.4.7 Chosen Interaction option	34		
4.5 Testing	35		
4.5.1 OOP and ECS approach comparison	35		
4.5.2 User testing	39		
<b>5 Conclusion</b>	<b>41</b>		
<b>A Bibliography</b>	<b>43</b>		
<b>B Attachments</b>	<b>47</b>		
List of Acronyms	47		
Enclosed Files	48		
Builds	48		
Scripts	48		

## Figures

2.1 Paramagnetic to ferromagnetic v.1	6
2.2 Paramagnetic to ferromagnetic v.2	7
3.1 Description of basic rules of Boids algorithm	13
3.2 Onset of Connectivity	15
3.3 Illustration of the fish neighborhoods used in the research	16
3.4 Formation of the school of fish related to the amount of fish present (N)	17
3.5 Wolf-pack ambush	18
3.6 Voronoi diagram	20
3.7 Distribution of points in 2D space	22
3.8 K-D tree visualization	22
3.9 Cube Space Partition Map	23
4.1 Static obstacle avoidance vector	26
4.2 The result vector "ret" calculation based on hierarchy structure	27
4.3 Performance benchmark visualisation.	38

## Tables

4.1 Testing device specifications. . . .	35
4.2 Boids benchmark . . . . .	37





# Chapter 1

## Introduction

A brief detour to biology will provide the necessary opening by answering a question: “Why do animals flock?”

The next step is to get acquainted with basic concepts of animal flocking such as *Criticality* and *Emergent Behavior* by understanding other natural phenomena that exhibit similar behavior patterns.

Afterward, a series of topics are discussed. First, the differences between real-time vs. pre-computed approach in flocking algorithms and animal flocking in the game industry vs. animal flocking during science simulations are inspected. Second, the list of selected animal flocking algorithms and approaches, providing a template on how to implement flocking behavior is assembled. Third, available functionalities of assets available on Unity Store and on Unreal Engine Market are given for comparison with basic flocking algorithms.

Functionalities are stated and described in as much detail as the scope of this work allowed in the implementation part of this work. The *ECS* workflow and its difficulties are contemplated upon. The comparison between the *ECS* and *OOP* approach is discussed. Testing of the simulation on the Interactive wall and the observations collected during the process are discussed alongside propositions for further improvements.

The conclusion summarizes the topics present in this work, draws a conclusion based on the *ECS* and *OOP* comparison, and puts the learned information into a wider perspective.



## Chapter 2

### Getting acquainted with the phenomenon

#### 2.1 Why do they flock

Animals flock mainly for survival reasons. More eyes see more. Individuals inside of the herd or school or any other flock can possibly rely on the perception abilities of their comrades on the edges and relax. Life in a herd can also increase the probability of finding a suitable mating partner. Overall, the flock is one of the basic defense strategies observed in nature and countless articles and discussions support this claim. But can this behavior be described as a consequence of evolution?

The question "Why do they flock?" is strongly connected with one underlying thought: The survival of the fittest in the Darwinian sense is often surpassed by the survival of the group or species. This claim is supported by the following examples.

##### 2.1.1 Migrating birds

When birds migrate and travel hundreds or even thousands of miles, they travel in specific formations. Not every position in the formation is favorable. Some birds must work harder to maintain the speed and altitude than others. What might come as a surprise is that birds change positions periodically during the flight to equally minimize the expense of energy of each individual bird in the flock. No individual has a preferred position in the formation at the expense of another one. And in that way the probability of survival of all increases, rather than the survival of the strongest specimen[Sat20].

### ■ 2.1.2 Ants

Ants form an amazing ecosystem. They can find unbelievably sophisticated and effective routes for food and material transport. Traffic on such routes is governed by three rules that are adhered to by all without exception:

1. Everyone proceeds with the same speed.
2. Everyone keeps in their lane—no overtaking.
3. At a bottleneck, travelers wait in small groups to allow alternating passage of the traffic in each direction (“zipper system”).

It has been proven that these principles are obeyed even in the case of extremely narrow bridges. And the effectiveness of such an approach can't be questioned (similar rules make the automobile traffic in large American cities such as Los Angeles move much more effectively than it does in old European towns). Based on this observation Helmut Satz provides an interesting hypothesis, that insect states have in a Darwinian sense eliminated cultures based on individual decisions [Sat20].

### ■ 2.1.3 Bees

Consider the bee workers. Apart from having developed an incredibly sophisticated way of communicating discovered sources of food through dance (performed both individually and in the group) [Sat20], another interesting fact begs to be mentioned.

While being perfectly adapted for taking care of larvae and providing the food necessary for the survival of the beehive, they are infertile. They pass on nothing. On the other hand, the queen, that is completely dependent on the workers and incapable of surviving on her own is the source of all future generations. The reason behind this is complex and definitely out of the scope of this work. But in general, supports the idea of the survival of the species at the expense of the survival of the strongest specimen.

The details about why the survival of the beehive depends on the reproduction of a seemingly helpless queen are further discussed in the book *The Animal Flocking* [Sat20].

### ■ 2.1.4 Answer

It gets more interesting the closer we look. The idea of the survival of the fittest is not merely surpassed by the survival of the species. It is its next step of evolution. When animals flock, they become one newly *emergent entity* that is fitter to survive. It possesses new abilities that were inaccessible before. That is why they flock.

For those interested in more behavioral detail, other examples and answers can be found in the book “The Rules of the Flock” by Helmut Satz [Sat20].

### ■ 2.1.5 A Mystery

It might seem that we now understand perfectly why animals flock. But do we? Before delving into more technical details about what mechanisms might initiate the flocking, I feel it necessary to state that we do not. The following example demonstrates such a claim.

In Rome, during evenings thousands of starlings perform outstanding formations in enormous flocks. Unlike in the case of flocking behavior during migration, the reason flocks of starlings in Rome appear in the evening is unknown. They are obviously not in search of food, since they spent the whole day feeding in the fields outside the city. They have no need to avoid any predators. And they are not leaving for several weeks for their summer destinations either. So why would they flock in such a spectacular way? We just don't know. In my opinion, the real reason might be somehow related to the creation of locust swarms.

Upon reaching a critical point of density (75 individuals per meter squared) the locust swarm is created. Sudden production of serotonin takes place in the locust's body. This even results in a change in its color. When such a locust wanders away a strong pull toward the center of the swarm takes place. This was the origin of one of the greatest observed locust swarms in human history in 1875 (approximately 12.5 trillion locusts formed a swarm)[Sat20].

Could it be that starlings form their flocks on Rome's evening sky out of pure happiness (because of the sudden serotonin increase in their bloodstream)? I admit this to be a speculation, but an intriguing one I must add. Further research and experiments in this field would be very interesting.

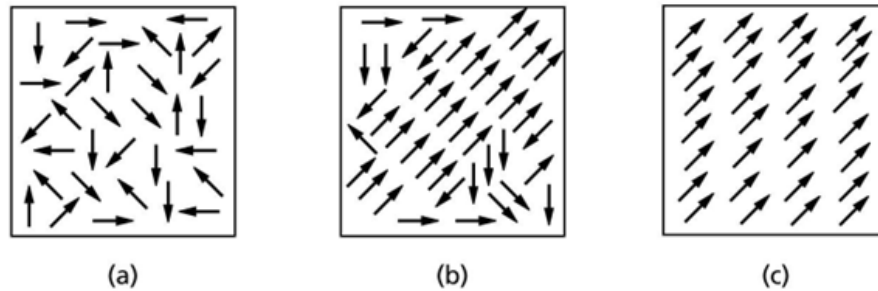
## ■ 2.2 Basic Concepts Of Animal Flocking

Consider the birds feeding on the ground. They form a group, but their bodies point and move in different directions and their movement is asynchronous. Then suddenly they get spooked by some external element. They all fly towards the sky as one body pointing in the same direction and often with the almost identical velocity. There is a parallel for such behavior in physics, that is very well-researched and documented.

### ■ 2.2.1 Behavior relation to atoms of iron

There are two basic states of atoms observed in a piece of iron. The first one is called the paramagnetic state and the second one is the ferromagnetic state. In the following figure 2.1, the difference in the alignment of atoms of iron between these stages can be seen.

Above a certain temperature threshold, known as Curie Point (768 degrees centigrade) all atoms spin in different (random) directions [Sat20]. They have a natural tendency to align themselves accordingly to one another, but the energy provided by temperature disturbs their effort. This state is known as paramagnetic and can resemble birds feeding on the ground. If birds resemble



**Figure 2.1:** Transition from a paramagnetic state above the Curie temperature (a) to a ferromagnetic state below that temperature (b) and a state of perfect order at absolute zero (c)[Sat20].

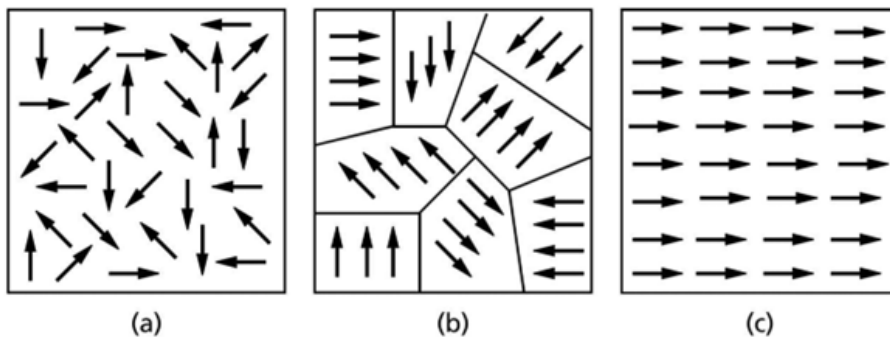
atoms of iron, the temperature might have a similar effect as a need to look for food. Believable implementation of such searching for food could be a noise in communication between the birds. Let's say all birds try to align themselves, but they perceive the orientation of other birds with some degree of error. The bigger the need to search for food on the ground, the bigger the error that leads to the more random movement of birds. Such behavior is proven to be attainable in flocks of high density if sufficient noise in their communication is present [VCBJ<sup>+</sup>95].

When the temperature falls below the Curie point, the tendency of atoms to align themselves can't be suppressed any longer. Groups of atoms aligned in the joint direction emerge abruptly. The temperature is still high enough to cause minor distortions inside the groups and produce inconsistency of alignment between groups in general, but there is at least partial order. Such a state is called ferromagnetic. With decreasing temperature, the alignment error between atoms and groups of atoms decreases. It has been proven both mathematically and through simulations that upon reaching absolute zero (-273 degrees centigrade) all atoms inside iron are perfectly aligned. Generally, they could point in any direction. On earth, they point toward the north. Such a perfect alignment can be obtained before reaching absolute zero though. If the external field was applied to iron in a ferromagnetic state, enough energy would be provided for atoms to align themselves and we would obtain a true magnet.

### ■ 2.2.2 Criticality

Since the times of Gottfried Wilhelm von Leibnitz and Sir Isaac Newton, infinitesimal calculus is known to humanity. This mathematical formalism is based on a sequence of infinitesimally small steps so that the progress appears to be continuous. Any deviation from such continuity is denoted as singular and the points of "jump" are called singularities. Physicists embraced the terms critical behavior for a "jump" and critical point for a point where the "jump" occurs.

The food-seeking tendency cannot battle the fear of potential danger. When



**Figure 2.2:** Transition from a paramagnetic state (a) to a ferromagnetic state exposed to the external magnetic field (c). Groups of atoms aligned in a joint direction are depicted as well (b)[Sat20]

birds get spooked, we could say that the inner structure of the flock changes suddenly. Abrupt (critical) behavior (“jump”) occurs as the flock reaches a critical point. The inner structure of the whole bird formation changes. What’s more, the impulse of fear is so severe that all birds align themselves almost perfectly and attempt to flee in an almost perfectly synchronized manner. The external field was provided in a form of potential danger, and it was strong enough to eliminate all the communication noise between birds. The almost perfectly synchronized flock emerges from seemingly randomly moving individual elements in a blink of an eye.

### 2.2.3 Self-organized criticality

*Criticality* leads to a sudden and united change in direction of the movement of birds. It was also stated that *criticality* is reached thanks to an external impulse. But a flock of birds often achieves this stage on its own. Birds manage to maintain a compact flock of thousands of individuals that move in one general direction. At the same time, the flock appears to be partly morphing and changing its inner structure dynamically. What triggers such behavior?

Nowadays we can claim that we at least partly understand what is happening. The answer is cyphered somewhere inside of the term: “Self-organized *criticality*” [Sat20].

Imagine pouring fine sand onto a flat surface. Each grain of sand represents one bird, and the emerging pile of sand will represent the flock. After reaching a certain critical value, the addition of one more grain of sand will lead to avalanches sliding down the pile to ensure the future stability of the whole structure. That can represent a few birds suddenly changing their place simultaneously inside the flock. It turns out that the number of avalanches produced depends on their size. The larger the avalanche, the rarer its appearance. The bigger the number of birds that simultaneously change their position inside of the flock, the rarer the phenomenon occurs.

Let’s say that at certain points we run out of birds to add to the flock. But

the flock keeps morphing, nevertheless. How is it possible? If we presume that the flock continually finds itself “overpopulated”, then every change in the inner structure is not efficient enough to produce long-term stability and its result must be another transformation (avalanche) of appropriate magnitude.

### ■ 2.2.4 Emergent behavior

When observing starlings dance in the sky, it comes to mind that they are indeed moving as if they constituted a single entity. How does it come that their movement is so synchronized?

As the flock reaches *criticality* on its own, something astonishing happens. Because there is no noise in the communication between neighbors upon reaching *criticality*, the alignment is almost perfect. In fact, for our simulation purposes, let’s assume it is perfect. That has an interesting result. Every bird is connected to every other bird in the flock through some chain of individual birds. Because of that, the movement of every two birds in the same flock is correlated. We speak of complex systems, whenever the intrinsic simple features (align with your neighbors) give rise to a new, emergent scale (synchronized movement of a single entity) - *E Pluribus Unum* (Out of many, one).

One self-explaining equation that represents the onset of connectivity by reaching a point of *criticality* begs to be presented at this point.

$$G(r, T) = \frac{e^{-\frac{r}{\epsilon}}}{r} \quad [\text{Sat20}]$$

$G(r, T)$  is a correlation function between two atoms,  $r$  is the distance between two atoms (birds),  $T$  is a given temperature (level of noise in the communication),  $\epsilon$  is the correlation length that is non-negative and inversely proportioned to  $T$ .

With the temperature (noise in the communication) skyrocketing the correlation between two atoms is almost non-existent (birds feeding on the ground). If we increased the temperature to its maximum level, we would obtain the following result:

$$\lim_{T \rightarrow +\infty} = \lim_{\epsilon \rightarrow 0+} \frac{e^{-\frac{r}{\epsilon}}}{r} = \frac{0}{r} = 0 \quad [\text{Sat20}]$$

Let’s see what happens if we decrease the temperature and approach the Curie point. The following figure implies that two atoms are correlated in their alignment no matter the distance (two birds in the same flock). Through correlation, a newly emergent state occurs: The flock moves as a single entity.

$$\lim_{T \rightarrow T_c} = \lim_{\epsilon \rightarrow +\infty} \frac{e^{-\frac{r}{\epsilon}}}{r} = \frac{e^0}{r} = \frac{1}{r} \quad [\text{Sat20}]$$

When  $T$  approaches the Curie point  $T_c$  (paramagnetic to ferromagnetic state) a point of *criticality* is reached,  $\epsilon$  goes to infinity as a correlation function becomes strictly positive.



A basic feature that determines the sudden onset of connectivity inside a flock of birds was just described.



## Chapter 3

### Pre-implementation part

#### 3.1 Real-time vs. pre-computed algorithms

Do I need my behavior simulation to respond to a dynamically changing environment? If so, then having a precomputed algorithm is too restrictive. Is the behavior loopable? Consider flies flying around a source of light. It would be unnecessary to compute such a behavior dynamically when the job can be done faster and easier by looping some predetermined paths of several flies. How complex is the behavior required to be and how long must it last? How many individuals are there in the flock or herd or school of fish?

Despite the benefits of modern hardware that mostly eliminated the memory shortage and provided us with immense computational power, the question of whether to choose real-time or precomputed algorithms remains. The right answer is dependent on all the questions in the previous paragraph and likely many more. Only through experience, we can determine when to choose which approach.

In general, pre-computed animations were and still are used in the film industry. There are two good examples of the original usage of the animal flocking algorithm to calculate the positions of animals during animation. The first one is Tim's Burton *Batman Returns* (1992), where all positions of bats flying in flocks were precalculated using Reynold's Boids algorithm [Rey87]. The second is the wildebeest stampede scene in Disney's *The Lion King* from 1994, which is regarded as a milestone in the usage of animal flocking algorithms in the movie industry.

Although the increasing possibilities of today's hardware open new potential in real-time behavior algorithm exploration, the precalculated simulations still have their place both in the film industry and in computer game development.

#### 3.2 Flocking in computer games vs science simulations

Is it enough if we persuade the user or viewer of the simulation that he is watching the behavior of an animal? If so, then a believable visual model combined with some basic behavioral patterns suitable for the animal will

be enough. The human brain classifies what it sees based on the patterns it already knows. If it looks like a duck, if it behaves like a duck, it is a duck. For most animals we see in computer games it is enough. If you see a small furry creature with long ears, jumping through the forest you automatically assume it is a rabbit. You care not that the rabbit leaves hole “A”, jumps between the trees for a while, then enters hole “B” and does nothing else.

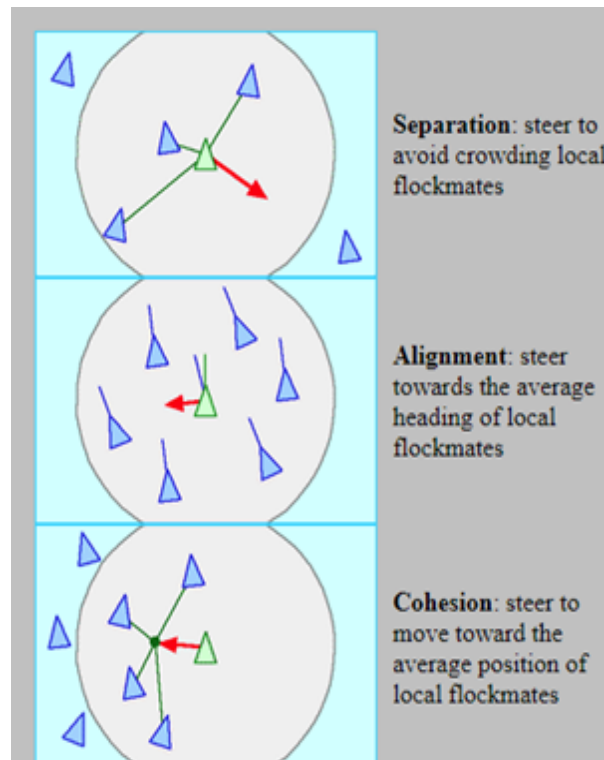
In contrast, during science simulation, we mainly care about behavioral patterns, their accuracy, and completeness. The goal is usually to be able to reproduce scenarios that resemble reality as much as possible and to obtain usable data during simulation. We have no need to convince anyone, that the dot during a simulation resembles a fish. We determine that it is a fish because its behavior will resemble that of a fish (as much as needed for the purpose of study and data collection). It is simply agreed upon. No additional visual demands are necessary.

Implementation complexity can differ greatly in both cases. If we are developing a computer game where your role is to be a shark in the sea that hunts for other fish, the behavior of the schools of fish must be believable and detailed as it will be part of the core of the game. If hunting as a shark is a one-time side quest, a less time-demanding solution for the fish behavior can be implemented, as this feature is no longer a cornerstone of the game. Similarly, in the scientific simulation of fish migration, it is not necessary to implement sophisticated flocking behavior at all. The flock’s resulting route and how it is affected by water streams are more important than the inner structure of the flock itself.

## 3.3 Basic animal flocking algorithms

### 3.3.1 Boids

Developed by Craig Reynolds in 1986, the Boids [Rey87] algorithm remains the cornerstone of the animal flocking behavior of birds, fish, and many more. Combining Separation, Cohesion, and Alignment proved to be effective enough to reproduce believable flocking behavior.



**Figure 3.1:** Description of basic rules of Boids algorithm [Rey87].

The main idea is based on the existence of neighborhoods. If the neighborhood of inspected boid contains other boids, the alignment and position of its neighbors are considered during the movement. Also, if any boid comes too close to another boid a separation force takes place and drives the two boids further away from each other.

The downside is the default time complexity of the algorithm, which is  $O(n^2)$ . This inconvenience exists because every boid must scan for every other boid in the environment during the calculation to determine, whether the other boid is part of its neighborhood.

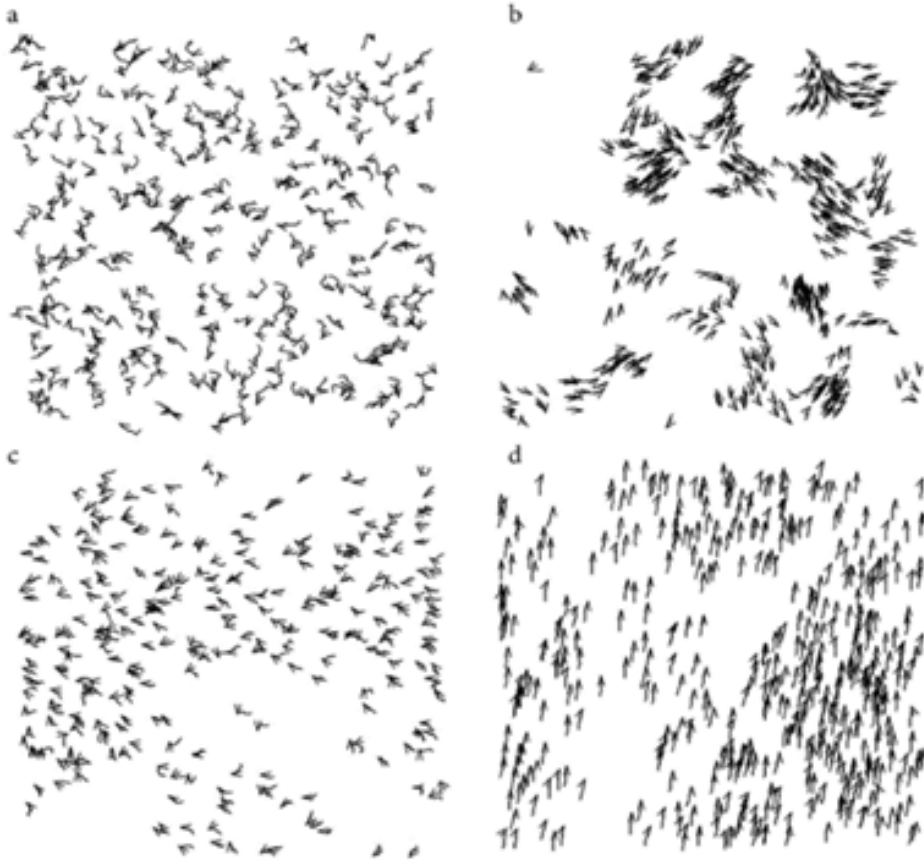
Luckily, many augmentations in the form of spatial distribution algorithms allowed the time complexity to become linear instead. And as we will see later, if we are willing to approximate the shape of the neighborhood the lookup of neighbors happens in a constant time.

Later in 1999 C.W. Reynolds extended this approach and added additional behavioral functionalities, such as pursuit, evasion, obstacle avoidance, wander, leader following and other behavior patterns [Rey02]. These algorithms quickly became popular and countless other researchers, built upon Reynold's discoveries and solutions.

The basic algorithm itself has its limits though. One of such imperfections was shown by P. Jonsson and L. Ljungber [PJ17]. There it was demonstrated that while being suitable for predators hunting flocking prey scenarios, the Boids algorithm is not suitable for flocking predators hunting flocking prey scenarios because it took more time for the flocking predators to accomplish the same results as multiple predators hunting individually.

### 3.3.2 Self-driven particles

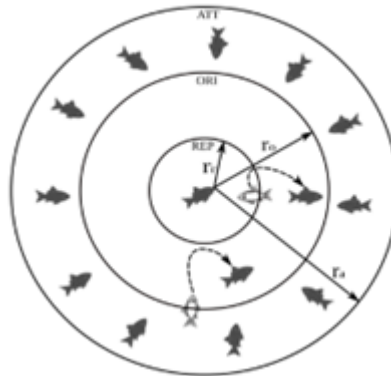
The study in 1995 by Vicsek [VCBJ+95] showed an indisputably apparent relation between the behavior of atoms of iron and self-driven particles. By introducing uniform noise (that has the same effect on boids as the temperature has on atoms) and a necessity of the particles to align their direction with each other, Vicsek successfully reproduced not only the behavior of atoms of iron but also that of locusts forming a swarm.



**Figure 3.2:** The phenomenon of a sudden onset of connectivity upon reaching a critical point when a swarm is created can be observed in this image (transition from step “b” to step “c”) [VCBJ+95].

### 3.3.3 Density induced transition in the school of fish

The research led by D.S. Cambuí [DSC12] did not introduce anything new on top of the Boids model. It only splits the neighborhood of the boid into three distinct parts (see Figure 3.3). Every part of the neighborhood was dedicated either to attraction (cohesion), repulsion (separation), or orientation (alignment). That was the only apparent difference compared to the Boids model, where alignment and cohesion could be applied in the same neighborhood).

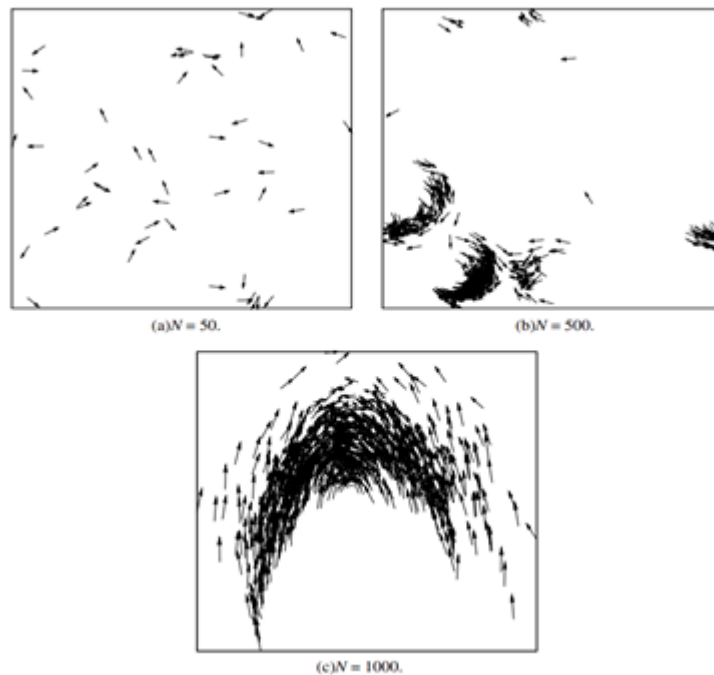


**Figure 3.3:** Illustration of the fish neighborhoods used in the research [DSC12].

What is more important was the goal of the study. It aimed to test whether this model is sufficient to produce the behavior of schools of fish based on empirical data of schools of fish observed in a water tank. And the answer was: Yes, it is. Results show that model can reproduce the patterns of movement observed experimentally for tilapia juveniles.

This produces no insignificant implication. It proves that a model of interaction as simple as Boids could be used effectively to reproduce general patterns observed in a large school of fish.





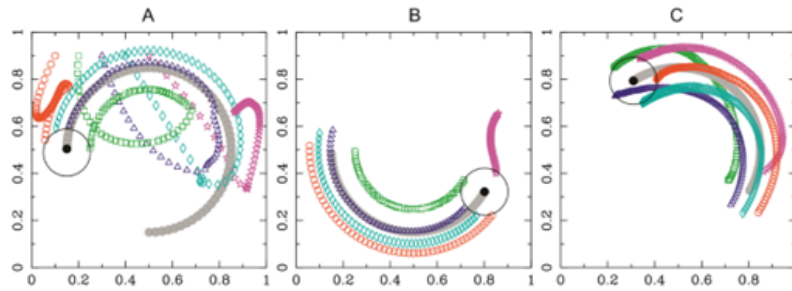
**Figure 3.4:** Formation of the school of fish related to the amount of fish present ( $N$ ) [DSC12].

### 3.3.4 Wolf-like behavior

As mentioned before a significant deficiency in the Boids algorithm was its unsuitability for flocking predators hunting flocking prey scenario. In the study of C. Muroa [MESC11] an attempt was made to recreate scenarios of wolf-like hunting behavior based on two simple laws. It turns out that for various group hunting scenarios to be successful only these two laws had to be abided by the objects portraying wolf-like hunting behavior during the scenario.

1. Move towards the prey until a minimum safe distance to the prey is reached.
2. When at a safe distance, move away from the other wolves that are within the safe area.

This resulted in wolf-like behavior encircling the prey. Some of the more complex hunting patterns were achieved as well. A good example is the ambush-like behavior of the pack.



**Figure 3.5:** Wolf-pack of five individuals hunting a counterclockwise circling prey. Waiting for ambush behavior is exhibited by the wolf denoted by a pink star. Black-filled circle: position of the prey at the last time step drawn. Grey-filled circles: prey trajectory. Colored symbols: wolves' trajectories. Large circle: points at a distance from the prey the last time step drawn. (A) The short approaching phase is followed by the pursuit phase during almost one loop. (B) Continuation of the pursuit phase during almost another loop. (C) Continuation of the pursuit until the final encircling phase in which a stationary distribution is reached: a regular polygon inscribed in the corona C [MESC11].

The question is how well is a wolf-like pattern of hunting and its possible solution transferable to hunting patterns among fish (as that is where the original inadequacy of the Boids model was pointed out by P.Jonsson and L. Ljungber). The answer is: it is transferrable to a certain degree. A yellow saddle goatfish has a modus operandi reminiscent of wolves and lions [SRB18]. One individual adopts a role of a “pursuer” while his comrades block the escape routes. Apart from a yellow saddle goat fish sharks, dolphins, seagulls, or even whales are known to cooperate during the hunt.

The phenomenon of underwater group hunting could be a standalone topic and other adjustments would probably have to take place to fully accustom the Boids model to adopt flocking predator behavior. But certain cases of underwater group hunting are potentially implementable using the two laws presented by C. Muroa as the hunting pattern of some predator fish resembles a wolf-like hunting pattern.

## 3.4 Available assets

Most of the assets available on the Unity store and Unreal Engine Marketplace implement the features presented in Reynold's papers, sometimes with slight changes. I decided to contact the asset developers to provide me insight and share what differentiates their implementation from other assets in terms of model behavior and algorithmic complexity of their solution. I sent an email to sixteen different developers regarding their assets on Unity Store and Unreal Engine Market Store. The responses that I deemed relevant and most valuable are summarized in the following paragraphs.

*FlockAI*[Bey] has eliminated the alignment computation from its behavior model. The feature remained but instead manifested itself as an *emergent behavior* because the boids try to get on top of each other.

The *ECS Swarms*[Tig] asset takes advantage of Unity's *ECS*. With the help of the Burst compiler[Unia], the vector math operations are speeded up as SIMD instructions are utilized better. Additionally, this asset takes advantage of sparse spatial partitioning. This allows the division of the space into chunks with assigned boids stored in a parallel hash map for faster neighbor look-up.

Similar functionality is implemented in the *Flocks*[Kie] asset. A concept from the GPU particle sim in "Wicked Engine"[tur] was used to perform a neighbor look-up in a constant time. It is an approximation but works very efficiently.

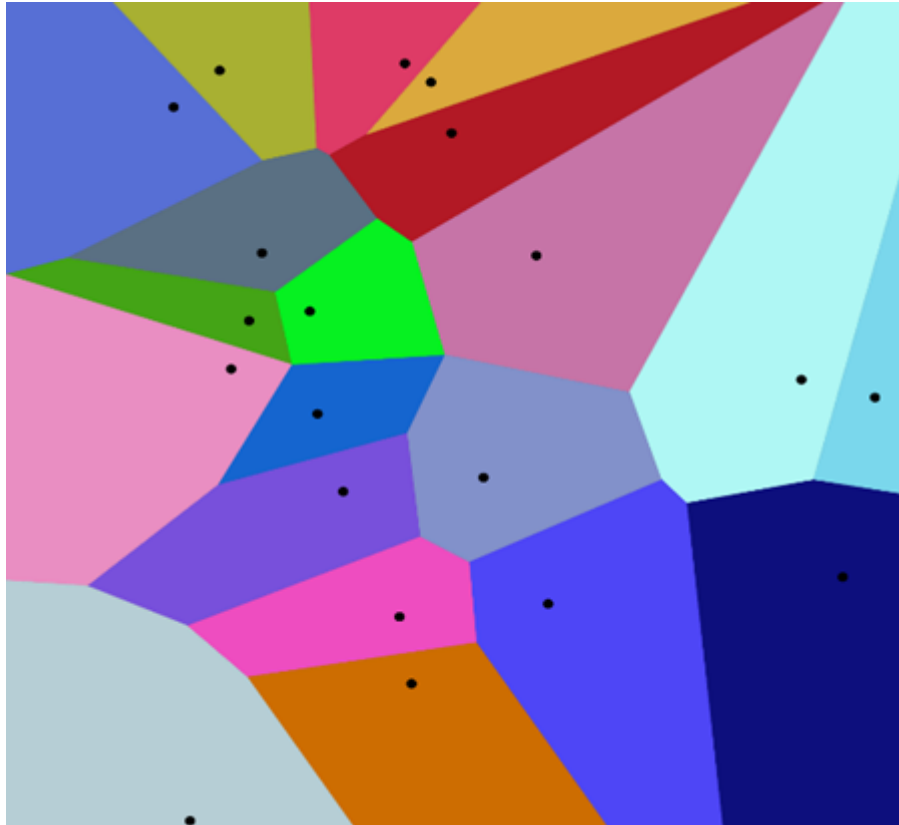
The problem of the nearest neighbor calculation could be a standalone topic. But there is an important aspect in the widely accepted approach to the neighbor issue, that I find even more interesting. Birds (starlings specifically) determine their distance based on topology distance, not a metric one [Sat20]. Metric distance does not affect the number of neighbors at all. The only thing that affects the number of perceived neighbors is starling's perception limitations. That is how far a bird can see and the fact, that a starling can count to number seven. Not more. This quite recent discovery reveals the misconception of the predetermined area around the bird that serves as its neighborhood.

A topological model, such as Voronoi space partitioning should be considered to determine boids neighbors.

## 3.5 Suitable space partitioning structures

### 3.5.1 Voronoi neighborhoods

One of the most attractive features of the Voronoi diagram is that the average number of neighbors of one element is six (approximately the same number as of the starling's neighbors in the flock).



**Figure 3.6:** Voronoi diagram is constructed based on the following idea. Define the neighborhood of a given particle as all those points of the plane closer to it than to any other particle [Sat20]

This model is scale-invariant, topological, and computationally achievable in higher dimensions. It is already used in computer graphics (Voronoi Shattering [Jos], modeling of terrain or other objects given a point cloud, determining meshes for space-discretized solvers) and Voronoi neighborhoods were already proven to be an efficient tool when used in Particle swarm optimization [SGAM09]. What's more, Voronoi neighborhoods seem to be an intrinsic part of the world. Our heart cells are organized in such a manner. Studies suggest that there are some galactic poles where galaxies are clustered in these poles due to gravity. These clusters form a structure that is like the Voronoi diagram.



### 3.5.2 K-D Tree

This is a special case of binary space partitioning tree, where every non-leaf node can be thought of as implicitly generating a splitting hyperplane that divides the space into two parts. The detailed understanding of how this data structure functions is not the focus of this, work but can be studied on many free internet sites[tan].

Imagine a scenario with multiple feeding grounds and a fish randomly positioned in the space. We have multiple static positions that present potential destinations. If the fish had to determine the closest place to go to, it would have to list all the possible feeding grounds. But if the positions of the feeding grounds were stored in a K-D tree all we would need to do is perform a nearest neighbor search operation that has a logarithmic time complexity in most cases (linear in the worst case).

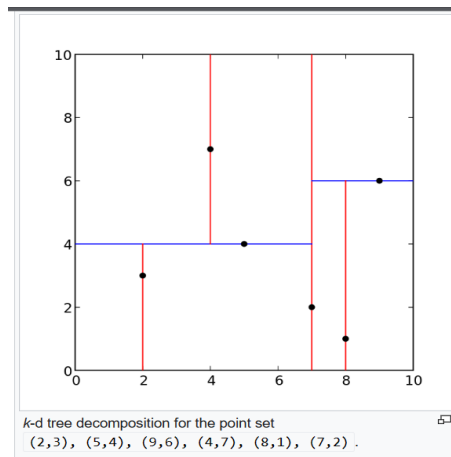


Figure 3.7: Distribution of points in 2D space[Wika].

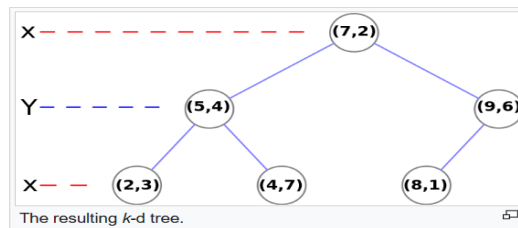
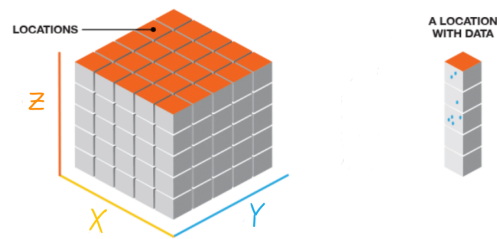


Figure 3.8: K-D tree visualization[Wika]

### 3.5.3 Cube space partition map

A commonly used approach is to divide the area the simulation takes place into cube-shaped chunks. This can be achieved by an integer division of the coordinates the boids are located in. The result of such a division could be then used as a key in a hash map collection where the values would present information about the boids located in the chunk. When a boid needs to locate its neighbors it first calculates a key using the integer division of its world space coordinates. Then it uses the obtained value as a key to access the relevant chunks in the hash map and extracts information (other boid's location, orientation, speed, etc.). This process can be also effectively parallelized. Unity has a built-in native collection that enables such a functionality[Unic].



**Figure 3.9:** Cube Space Partition Map that illustrates the space division into cube chunks with associated data stored in each chunk[Arc].





## Chapter 4

### Implementation

It is appropriate to list a few key functionalities implemented in my solution of the flocking algorithm with the *ECS* approach. There are a lot of small tweaks that boost the believability of the simulation but for the sake of clarity, I decided to list the most important parts of the implementation.

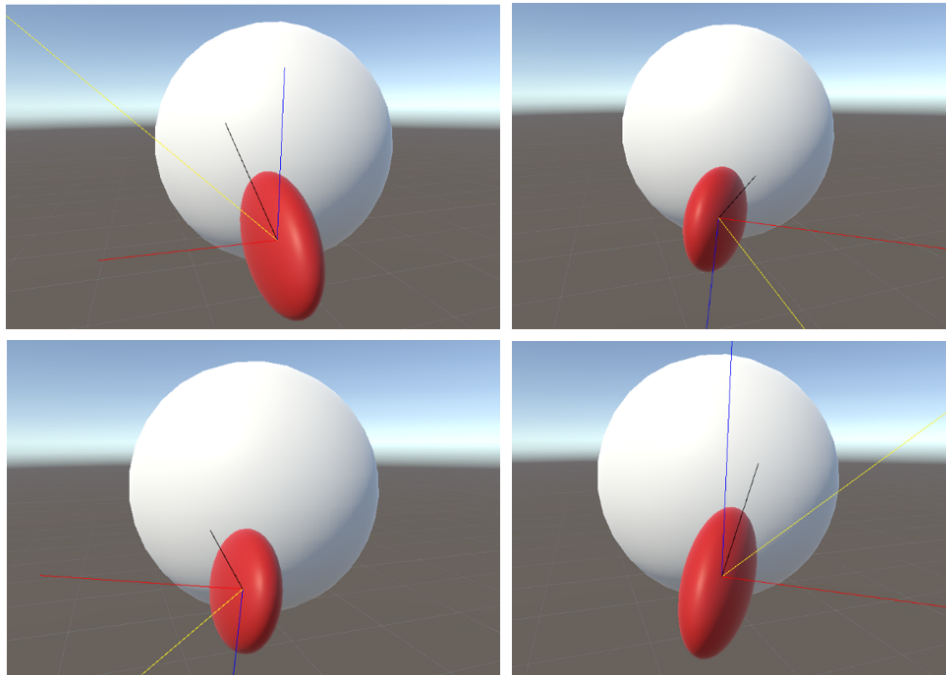
For the creation of the scene, an asset *POLYGON - Nature Biomes - Season One*[Sto] with a collection of biomes was used from the Synty store.

#### 4.1 Functionalities

##### 4.1.1 Obstacle Avoidance

Obstacle avoidance is calculated with simplicity and predictive behavior in mind. A boid casts a sphere in the forward direction. If there is a hit with an obstacle collider the reflection vector[Unie] based on the surface normal of the hit point is calculated and projected on the plane defined by the boid's forward vector (used as a plane's normal vector). The terrain collider has been successfully transformed into the *ECS* world. Therefore, the terrain is treated as a static obstacle as well. The solution works if three conditions are met.

1. All the obstacles have convex colliders.
2. The minimal distance between all the obstacles is big enough for the boid to fit through.
3. There are no blind alleys.



**Figure 4.1:** The black line describes the direction from the boi's center to the hit point with the obstacle's collider. The blue and the red lines describe the horizontal and vertical axis of the projection plane respectively. The yellow line represents the resulting obstacle avoidance vector. [author]

#### 4.1.2 Boids flocking behavior

When it comes to boids behavior, I implemented the well-known Reynold's forces that determine the boi's desired direction of movement. I purposefully omitted the alignment force, as all the boids always intrinsically aim for a destination point (whether it is a feeding or spawning ground or another boi in case of a predator). I divided boids in my scene into two categories: the prey and the predator. Both these categories have slightly randomized movement abilities between each other to provide a non-uniform visual experience. Both prey and predator also avoid statically defined obstacles.

It should be noted that my implementation allows for multiple types of prey to be present. But rather than having two types of prey that would exhibit the flocking behavior to merely fulfill the requirements of the guidelines for this thesis, I decided to opt for the solution with one type of prey and predators chasing it. This allowed me to fine-tune the escape-like behavior of prey and the pursuit logic of predators. This decision was consulted and approved by the supervisor of my thesis.

The prey actively flocks with other prey using Reynold's cohesion property and maintains defined separation that changes when a predator approaches its vicinity. The predator avoidance vector has two forms. The first one is simple avoidance using a reversed direction vector from the prey toward the predator. The second is triggered when the predator approaches the

prey more closely and results in an escape-like behavior of the prey[autb]. The predator chooses its hunt target based on its movement ability defined by rotation and movement speed. It will never choose prey that exhibits a high potential to escape during the hunt. This way I partly mimicked the behavior in nature, where predators choose the ill and weak-looking prey. The predator's flocking abilities also differ from the behavior of the prey. It is more concentrated on the selection of suitable prey and its pursuit. The predator's interaction with other predators is limited to separation calculation to decrease the number of collisions.

An additional problem had to be solved. A necessity for priority management of direction vectors had to be applied. I also aimed to implement the priority structure in a way that would allow each direction vector to be taken into consideration proportionately to its deviation from the vector that preceded it in the hierarchy. In specific scenarios, it proved to be efficient to ignore the predefined way of vector addition. Therefore, an option to ignore the cosine-based calculation is provided in the function as well.

```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
12 references
internal static float3 GetComposedDirectionVectorNormalized(in float3 leadingVector,
    float3 newVector, bool ignoreNewVectorCoinusBasedComputation = false)
{
    if (ignoreNewVectorCoinusBasedComputation)
        return math.normalizesafe(leadingVector + newVector);

    float power = math.cos(MyMath.AngleInRadians(leadingVector, newVector)) / 2f + 0.5f;
    float3 ret = (leadingVector + power * newVector);

    return math.normalizesafe(ret);
}
```

**Figure 4.2:** The “leadingVector” reference is a destination vector computed so far. The “newVector” parameter is the next direction vector in the hierarchy to be added to the resultant vector. Note that with an increasing angle between the “leadingVector” and “newVector” the “power” value approaches zero. [author]



*Entity Component System* is a part of Unity's Data-Oriented Technology Stack (*Dots*) approach that leverages performance over human readability by using stack allocations. While the concept might be more difficult for humans to fully understand and efficiently use (and is more error-prone than the *OOP* approach) it allows the computer to access required data fast. It also puts the memory allocation in the hands of the programmer (instead of relying on the garbage collector) once again. The core concept revolves around dividing the object into three main parts: *Entity*, *Component*, and *Systems*. For simplification purposes, I will compare each part of the *ECS* approach to its relevant *OOP* counterpart. This comparison should be viewed as purely conceptual rather than factual.

An *Entity* does not contain any code nor serves as a container for associated data. It is just an ID that associates unique *Components* together. In the *OOP* approach, the counterpart could be the reference to the object.

A *Component* serves as storage for unmanaged data that are related to a certain *Entity*. The closest counterpart could be an attribute of a class instance in the *OOP* approach.

A *System* provides a logic that transforms the *Component's* data from its current state to another. It contains the necessary logic that is used to modify the data stored in the *Components*. You can further define dependencies between *Systems* and schedule their order of execution. An *ECS* programmer will most often rely on Unity's Job system as well to further increase performance by allowing parallel execution of operations. A relevant comparison to *ECS's System* could be a class instance method in the *OOP* approach. Because in the *ECS* world, most of the data is unmanaged it should be mentioned that my project took advantage of the Burst compiler (a compiler that can compile a subset of C# into optimized native code).



### ■ 4.3.2 File management

One significant downside to the *ECS* approach is that it takes more files to define a single operation. First, the *Components* of an *Entity* need to be defined. Then you create an *Authoring script* that is attached to the object that will be transformed into its *ECS's* counterpart. It is convenient to create an *Aspect* that encapsulates relevant *Components* of an *Entity* and create methods that transform the *Component's* data. At last, a *System* needs to be written to fully enable the transformation of the *Entity's Component* data. This results in the creation of at least four separate files. All this could be written in one script using the *OOP* approach.

When dealing with multiple *Entities*, each one having its own *Components*, *Aspects*, *Authoring scripts*, and *Systems*, it can get chaotic. A straightforward approach places all *Components* in one folder and each *System* into another, etc. This quickly led to a problematic lookup of the necessary file in the project. Instead, I chose a file management approach that distinguishes each *Entity* that has its own *Components*, etc. stored in its own folder hierarchy. This solved the problem in most cases. Sometimes certain *Systems* had to operate on multiple *Components* related to different *Entities*. These *Entities* had to be packed in a folder with a shared *Systems* folder that contained the previously mentioned *System*. This solution proved to be the most efficient way of maintaining readability and relatively easy lookup of the project's files.

### ■ 4.3.3 Actively search for help

The *ECS* in Unity is still a new technology. It only became available in version 1.0 in December of 2022. The documentation exists, but certain detailed descriptions might still be missing to this date. I often ran into a problem that did not have a prescribed way of solution on the official pages. When this happened, the Unity Forum proved to be very useful.

For instance, I managed to get counsel[Avo] on how to effectively transform the terrain collider into the *ECS* world there. To this date (5.5.2023) the official support for terrain-related operations is not present in Unity ECS. That can be said for skinned animations as well. For the purpose of demonstration, I created a few models of fish and animated them using Maya software[auta]. Sadly, there is no tool yet for using this type of animation in Unity ECS. A workaround exists though. It should be possible to divide the models into separate pieces and “manually” move these separate parts as to mimic the effect of the animation of the fish. Unfortunately, this was out of the scope of the allotted time I had to finish a project demo. Still, it might present at least a temporary solution for the missing animation functionality.





place. For this reason, a speech-based interaction is not an optimal option for this installation.

### ■ 4.4.3 Motion-capture based interaction

An interactive projection wall that captures certain gestures mapped to functionalities like spawning, repulsion, or attraction could prove very effective. It would be a sufficient substitute for a tap-like interaction and does not suffer from the tumult limitations as the speech recognition-based interaction. Even a very young child would be able to interact with the simulation by just moving around. By coincidence, a Projection interactive wall is being developed on CTU FEE[CF]. The downsides and complications that are connected with this option are discussed in the sections 4.4.7 and 4.5.2.

### ■ 4.4.4 Remote phone interaction

Another possibility relies on people being able to connect to the Unity project through a mobile app. A simple command could be sent using their mobile phones. They could spawn boids or even force a temporary tap-like interaction on predefined areas of the screen through buttons. These actions could be then stored in a buffer and executed sequentially. The networking interface is also present in both the standard Unity projects and the still experimental *ECS* approach. Another plus is that if some unexpected problems occurred during the implementation of networking in *ECS* all the functionalities could be implemented with a standard object approach and then communicated to the *ECS* world separately inside of the project. It also enables dozens of people being able to interact with the simulation at the same time and removes the necessary constraints on screen size or position and removes the limitations posed by speech recognition interaction. Even very young children can tap on designated parts of the phone screen these days.

The downside relies on the detachment of the user from the simulation by using the phone as a mediator. I would like to provide a sensation of imminent power when the user controls the simulation directly (motion-capture, speech recognition, tap screen). Such a feeling would be hard to achieve if the user needed a mediator to control the situation.

### ■ 4.4.5 Holographic projection

Nowadays it is possible to emulate a 3D visual experience with a projection. With the use of a projector placed at the ceiling and projecting the simulation vertically in a direction to the floor with a special folie placed (in the projection space) at a certain angle, the simulation could appear extremely immersive[Dis]. A motion-capture could enable the simulation to take the user's distance from the stage into consideration. If these features were combined the boids could react to the proximity of the viewers. This sounds both ambitious and technically challenging.



Unity editor. The reason I am stating the issue at this point is that it severely affected my choice of options regarding the interaction possibilities. I was only able to launch the simulation in the Unity editor, nowhere else.

By the end of April, a version of Unity editor 2022.2.16f appeared to fix the issue with the incorrect sub-scene ID assignment and allowed for a successful launch of the application after the build. At this point, I was given the option to consider both the implementation and testing of interaction options. Thanks to my supervisor I was able to gain access to the Projection interaction wall developed on CTU. After including the necessary package, calibration of the scene's cameras, and other tweaks in the project, the simulation was successfully executed on the wall.

At the moment I was given access to the Projection Interaction wall (May 2023) it provided these functionalities: recognition of positions of both left and right hands and recognition of the following gestures: swipe up, down, right, and left (done with hands). Of these functionalities, the one that were the most reliable was the recognition of the hand's position. It was stated several times by Ing. Ondřej Slabý that gesture recognition needs to be made more robust and reliable. Because of this I mainly took advantage of the hand position recognition. The position of the right hand was mapped to the screen space of the simulation and a starting point for a sphere cast was inferred from it. Upon collision with the boids in the simulation either the attraction or repulsion force was to be applied to the boids. I also attempted to map the gestures to the spawning of the boids in the scene and considered mapping certain gestures to swap between the attraction and repulsion modes.

## 4.5 Testing

### 4.5.1 OOP and ECS approach comparison

When comparing the performance between the standard object and *ECS* approach I created four scenarios. Two with the classic *OOP* approach in mind and two that took advantage of the *ECS* approach. Apart from the boids, the scene was empty as the main purpose was to benchmark the performance with various implementations of the flocking algorithm. All benchmark tests were carried out on my notebook with the latest drives installed (updated on the 13th of May 2023). The device is further specified in the table 4.1.

CPU	Intel(R) Core(TM) i5-10500H CPU @ 2.50GHz 2.50 GHz
Graphics card	Nvidia GeForce RTX 3060 Laptop GPU
RAM	16GB
System Type	64-bit operating system, x64-based processor
Windows edition	Windows 11 Home

**Table 4.1:** Testing device specifications.

The first option used a standard object approach with physics colliders set as triggers and standard object containers used to keep track of current neighbor boids. The second option used a space partition map (implemented with the *OOP* approach) for neighbor detection instead of relying on colliders. In both cases, the number of boids taken into consideration was limited to eight.

The third and the fourth scenarios took advantage of the *ECS* approach with a space partition map for neighbors to look up and Unity's native collections as containers to keep track of current neighbors. Operations on these collections were often done in parallel when the game logic allowed such a possibility. The third option had the number of boids taken into consideration as neighbors set to eight. The fourth scenario further increased the number of neighbors to thirty-two. The simulation was able to process the same logic with a significant increase in performance.

All of the benchmark results are present in the table 4.2.

Benchmark: Objects with colliders with 8 maximum neighbors.							
Sum(Boids)	0	100	200	300	400	500	600
Avg(FPS)	~ 397	~ 380	~ 272	~ 200	~ 132	~ 95	~ 56
Sum(Boids)	<b>700</b>	800	900	<b>1000</b>	1300	-	-
Avg(FPS)	<b>~ 31</b>	~ 20	~ 10	<b>~ 8</b>	~ 4	-	-

Benchmark: Objects with space map with 8 maximum neighbors.							
Sum(Boids)	0	100	200	300	400	500	600
Avg(FPS)	~ 397	~ 360	~ 251	~ 180	~ 125	~ 93	~ 68
Sum(Boids)	700	<b>800</b>	900	<b>1000</b>	1300	-	-
Avg(FPS)	~ 46	<b>~ 37</b>	~ 26	<b>~ 20</b>	~ 8	-	-

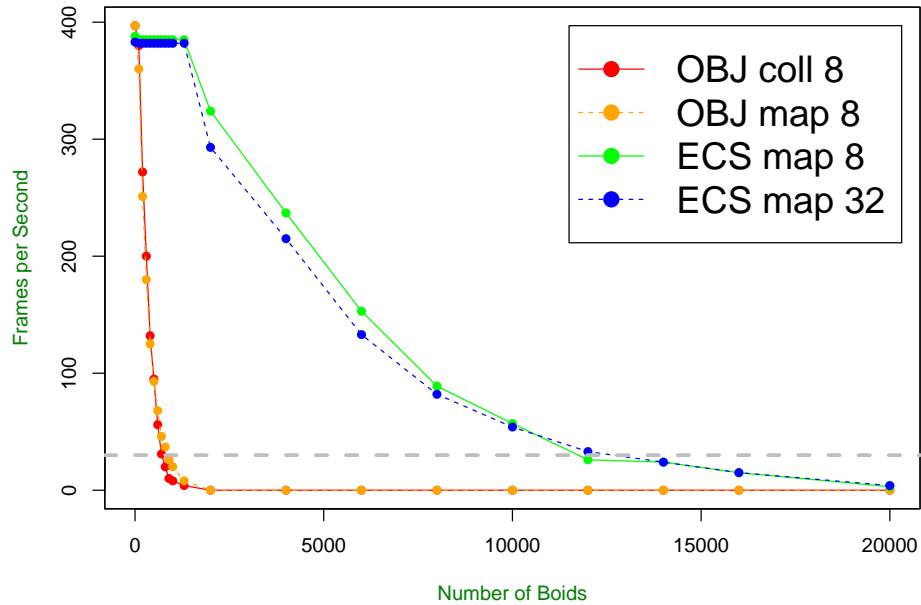
  

Benchmark: ECS with space map with 8 maximum neighbors.							
Sum(Boids)	0	<b>1000</b>	2000	4000	6000	8000	<b>10000</b>
Avg(FPS)	~ 388	<b>~ 385</b>	~ 324	~ 237	~ 153	~ 89	<b>~ 57</b>
Sum(Boids)	12000	14000	16000	20000	-	-	-
Avg(FPS)	~ 26	~ 24	~ 15	~ 3	-	-	-

Benchmark: ECS with space map with 32 maximum neighbors.							
Sum(Boids)	0	<b>1000</b>	2000	4000	6000	8000	10000
Avg(FPS)	~ 383	<b>~ 382</b>	~ 293	~ 215	~ 133	~ 82	~ 54
Sum(Boids)	<b>12000</b>	14000	16000	20000	-	-	-
Avg(FPS)	<b>~ 33</b>	~ 24	~ 15	~ 4	-	-	-

**Table 4.2:** Performance benchmark for *OOP* and *ECS* implementation of my flocking algorithm. "Sum(Boids)" named row of cells refers to the overall sum of all boids in the scene. "Avg(FPS)" named row of cells refers to the average FPS for the sum of boids. The FPS is calculated as a sum of frames from the time span of ten seconds divided by ten. Cyan columns refer to test cases with one thousand boids for direct comparison purposes between benchmarking scenarios. The green columns mark the last case within each scenario where the sum of boids had the respective average FPS greater than thirty.



**Figure 4.3:** Graphic visualisation of table 4.2. "Obj coll 8" refers to the scenario with *OOP* approach with colliders and eight neighbors, "Obj map 8" refers to the scenario with *OOP* approach with space partition map and eight neighbors, "ECS map 8" refers to the scenario with *ECS* approach with space partition map and eight neighbors, and "ECS map 32" refers to the scenario with *ECS* approach with space partition map and thirty-two neighbors.[author]

After I finished the scene and launched the *ECS* simulation with thirty-two maximum neighbors per boid with all visual and post-processing effects the frame rate dropped to approximately thirty frames per second (with ten thousand actively flocking boids). I consider this performance to be acceptable for user testing purposes and therefore I chose it as the default maximum number of boids in the scene.

## ■ 4.5.2 User testing

Results of the testing can be inspected in this video[autc]. If not stated otherwise I always refer to this video in the following paragraphs.

The mapping of the hand's position and inferring the start point to cast the sphere (which caused the boids to be either attracted or repulsed by it) proved to be working reliably. The mapping of gestures to spawn events was tested as well. As it is apparent from the video it worked in most cases. Unfortunately, as the gesture and position of the same hand were considered in this particular case, sometimes the gesture event was triggered when the hand was moved to change the sphere cast location even if the user did not intend to do the "boids spawn" gesture. In other cases, the gesture was not recognized. Because of this, I decided to provide the user with an alternative way for spawning boids by mapping this functionality to some keyboard keys. The ability to switch between repulsion and attraction modes was done using the keyboard as well.

It can be said that children learned the basic interaction rules fast. After a few minutes of explanation of the simulation and a showcase of how to interact with it, they had no problem reproducing the interaction. When they were uncertain they talked to each other and advised their comrades on how to interact with the wall. This behavior is captured in the video as well. They found the attraction effect especially intriguing as it produced a whirlpool-like effect. The kids often stared at it for dozens of seconds without doing anything else. They just held their arm up and observed it. When I asked them about it, they said they enjoyed the feeling of controlling the fish and that was enough for them. It is interesting how fulfilling such a feeling of control is to humans. I myself enjoyed it as well during the early stages of simulation testing. Another interesting moment was when the barracuda fish model (that represented the predator) was spawned and the attraction force took place. Because the fish was spawned in the proximity of the camera it was affected by the effect very soon and the fish was basically unable to escape it. All the kids that stumbled upon this situation spent sometimes minutes just controlling the barracuda fish and forcing it to move in circles. While it produced an interesting moment, it should be noted that the start of the sphere cast should be moved further away from the camera into the scene with the effect being cut off after reaching a specified minimum distance from the sphere cast start point. This way the fish affected by the attraction effect in close proximity to the camera can be given a chance to eventually escape it.

When I asked the kids what it is they liked about the experience they almost always mentioned the attraction effect and especially the moments when they controlled the barracuda fish movement in close proximity from the camera position. Some of the girls said they felt scared when the barracuda appeared and hugged each other, but later found the fish funny and wanted to control it. Among their suggestion was richer underwater wildlife. They wanted to see octopuses, whales, and sharks. They also wanted the famous tv character SpongeBob[Wikb] in the scene alongside its companion Patrick.

Adults seemed to be more experimental, especially at the beginning of the interaction. While the interaction was primarily designed for children, it seemed that adults enjoyed the effects and the overall experience as well. I have been told a few times that the simulation is a "zen-based interaction". They found the results of it calming and relaxing. Some said they could imagine playing with it when waiting for their turn at the dentist. Others said they would expect this simulation at some underwater aquarium gallery.



## Chapter 5

### Conclusion

Understanding basic concepts like self-induced *criticality* and *emergent behavior* helped me to adopt a certain type of philosophical approach during the implementation that had to be compatible with the *ECS* coding limitations and requirements. Correct execution of the *Systems* as well as setting appropriate job dependencies were taken care of.

Exploration of available solutions in the Unity asset store and Unreal marketplace provided the necessary inspiration and served as a basic implementation solution to compare to animal flocking behavior observed in nature. It was concluded that the main difference is the metric-based determination of neighbors used in the simulation (starlings use a topology-based distance). Fortunately, it was proven [DSC12] that at least in the case of fish this approach produces results almost identical to those observable in nature.

When it came to deciding on the interactive screen solution the Projection interactive wall developed at CTU FEE was chosen as it was both a convenient choice suitable for children and also probably the only one (apart from click-based mouse interaction) that was feasible given the time limitations caused by the internal Unity bug (section 4.4.7).

A prey and a predator are two types of fish that exhibit different flocking abilities in my work. The reason why predators' flocking abilities are limited is stated in section 4.1.2. The Moorish Idol and Great Barracuda models were created for the presentation of these types respectively. Their final scale was adjusted for better visibility on the projection wall.

When comparing the difference between the *OOP* approach with colliders and space partition map it is apparent that before the space map could be taken advantage of, the object implementation itself became the bottleneck of the performance.

With the *ECS* approach the boost in performance is apparent as the simulation was able to process tens of thousands of boids instead of hundreds (when using the *OOP* approach with space partition map). The *ECS* concept opens new possibilities for the maximum number of particles in real-time simulations. A simulation with such a high number of real-time computed particles would be unthinkable on mid-price ranged devices (like my notebook) before.

The *ECS Swarms*[Tig] asset provided implementation that enables tens

of thousand boids. It should be noted that while the asset provides similar functionalities it lacks many of the optimization tweaks that enable the predator-prey pursuit scenario in my implementation. I have also separated several functionalities into different *Systems* for code clarity purposes. Because of these tweaks and code separation, multiple creations of the same space partition map had to be performed. Therefore, the performance is slightly decreased compared to the ECS Swarms asset.

The main purpose of the testing was to compare the performance of the *ECS* and the *OOP* approach. This test was successfully carried out on my device after the internal Unity bug (section 4.4.7) was fixed and the application could be launched after it was built. Because of the relatively small time window before the work had to be submitted (once the internal bug was finally fixed), I decided to dedicate the available time I was given to access the Projection interaction wall for user testing purposes rather than to perform a similar benchmark test on the wall. As can be seen from the video the simulation runs smoothly on the wall with the maximum sum of boids in the scene limited to ten thousand.

The user testing proved that both children and adults are interested in interacting with this type of simulation. When gesture recognition of the Projection interaction wall will be made more robust, the spawn and attraction-repulsion switch actions could be mapped on the gestures. This way the keyboard can be omitted from the interaction completely. The simulation appeared to run very smoothly on the interaction wall with twenty thousand actively flocking boids. Therefore it is possible to make the simulation more complex and larger in scale if needed. Because the logic of the simulation is separated from the user interaction logic it is possible to map the behavior to different events if needed in the future.

## Appendix A


### Bibliography

- [Arc] ArcGis, *Cube space partition map image*, <https://desktop.arcgis.com/en/arcmap/latest/tools/space-time-pattern-mining-toolbox/create-space-time-cube.htm>, Accessed on 8 May 2023.
- [auta] author, *Fish animation done in maya*, <https://youtu.be/pRMZgJtLguo>, Accessed on 20 May 2023.
- [autb] ———, *Prey pursuit demo*, <https://youtu.be/xLTxIfJ1LG4>, Accessed on 12 May 2023.
- [autc] ———, *Simulation testing*, <https://www.youtube.com/watch?v=tbQAXZH75Qk>, Accessed on 20 May 2023.
- [Avo] Avol, *Ecs terrain transformation*, <https://forum.unity.com/threads/using-unity-terrain-with-dots-workflow.755105/page-2>, Accessed on 14 May 2023.
- [Bey] BeyondVR, *Flockai*, <https://assetstore.unity.com/packages/tools/ai/flockai-227995>, Accessed on 13 January 2023.
- [CF] CVUT-Fel, *Interactive projection wall*, <https://github.com/iimcz/ipw-firmware>, Accessed on 12 May 2023.
- [Dis] Glimm Display, *Holographic projection*, <https://www.youtube.com/watch?v=r27wrQ0gawo>, Accessed on 12 May 2023.
- [DSC12] Alexandre Rosas Dorílson Silva Cambuí, *Density induced transition in a school of fish*, *Physica A: Statistical Mechanics and its Applications* **391** (2012), no. 15, 3908–3914.
- [Jos] Jose, *Voronoi shattering (part i)*, <https://www.joesfer.com/?p=60>, Accessed on 13 January 2023.
- [Kie] Justin Kiesskalt, *Flocks*, <https://www.unrealengine.com/marketplace/en-US/product/flocks>, Accessed on 13 January 2023.

- [MESC11] C. Muro, R. Escobedo, L. Spector, and R.P. Coppinger, *Wolf-pack (canis lupus) hunting strategies emerge from simple rules in computational simulations*, Behavioural Processes **88** (2011), no. 3, 192–197.
- [Mic] Microsoft, *Cognitive speech services sdk*, <https://github.com/Azure-Samples/cognitive-services-speech-sdk/blob/master/quickstart/csharp/unity/text-to-speech/README.md>, Accessed on 12 May 2023.
- [PJ17] L. Ljungberg P. Jonsson, *Flocking as a hunting mechanic: Predator vs prey simulations*, Ph.D. thesis, KTH ROYAL INSTITUTE OF TECHNOLOGY, 2017.
- [Rey87] Craig W. Reynolds, *Flocks, herds and schools: A distributed behavioral model*, SIGGRAPH Comput. Graph. **21** (1987), no. 4, 25–34.
- [Rey02] Craig Reynolds, *Steering behaviors for autonomous characters*, 763–782.
- [Sat20] Helmut Satz, *The rules of the flock*, Oxford University Press, 2020.
- [SGAM09] Ehsan Safavieh, Amin Gheibi, Mohammadreza Abolghasemi, and Ali Mohades, *Particle swarm optimization with voronoi neighborhood*, 2009 14th International CSI Computer Conference, 2009, pp. 397–402.
- [SRB18] Marc R. Steinegger, Dominique G. Roche, and Redouan Bshary, *Simple decision rules underlie collaborative hunting in yellow saddle goatfish*.
- [Sto] Synty Store, *Polygon - nature bioms*, [https://syntystore.com/products/polygon-nature-biomes-season-one?\\_pos=4&\\_sid=d811435c1&\\_ss=r](https://syntystore.com/products/polygon-nature-biomes-season-one?_pos=4&_sid=d811435c1&_ss=r), Accessed on 12 May 2023.
- [tan] tanvibugdani, *Search and insertion in k dimensional tree*, <https://www.geeksforgeeks.org/search-and-insertion-in-k-dimensional-tree>, Accessed on 13 January 2023.
- [Tig] Tigpan, *Ecs swarms*, <https://assetstore.unity.com/packages/tools/ai/flockai-227995>, Accessed on 13 January 2023.
- [tur] turanszkij, *Gpu fluid simulation*, <https://wickedengine.net/2018/05/21/scalabe-gpu-fluid-simulation/>, Accessed on 13 January 2023.

- [Unia] Unity, *Burst compiler*, <https://docs.unity3d.com/Packages/com.unity.burst@1.8/manual/index.html>, Accessed on 14 May 2023.
- [Unib] ———, *Ecs documentation*, <https://docs.unity3d.com/Packages/com.unity.entities@1.0/manual/index.html>, Accessed on 13 May 2023.
- [Unic] ———, *Nativeparallelmultihashmap*, <https://docs.unity3d.com/Packages/com.unity.collections@1.1/api/Unity.Collections.NativeMultiHashMap-2.html>, Accessed on 8 May 2023.
- [Unid] ———, *Unity physics*, <https://docs.unity3d.com/Packages/com.unity.physics@1.0/manual/index.html>, Accessed on 14 May 2023.
- [Unie] ———, *Unity physics - reflect*, <https://docs.unity3d.com/Packages/com.unity.mathematics@1.2/api/Unity.Mathematics.math.reflect.html>, Accessed on 14 May 2023.
- [VCBJ<sup>+</sup>95] Tamás Vicsek, András Czirók, Eshel Ben-Jacob, Inon Cohen, and Ofer Shochet, *Novel type of phase transition in a system of self-driven particles*, *Phys. Rev. Lett.* **75** (1995), 1226–1229.
- [Wika] Wikipedia, *K-d tree*, [https://en.wikipedia.org/wiki/K-d\\_tree](https://en.wikipedia.org/wiki/K-d_tree), Accessed on 13 January 2023.
- [Wikb] ———, *Spongebob*, [https://en.wikipedia.org/wiki/SpongeBob\\_SquarePants](https://en.wikipedia.org/wiki/SpongeBob_SquarePants), Accessed on 20 May 2023.





## **Appendix B**

### **Attachments**



#### **List of Acronyms**

**ECS** Entity Component System

**OOP** Object Oriented Programming

**Dots** Data-Oriented Technology Stack

**CTU** Czech Technical University

**ČVUT** České Vysoké Učení Technické v Praze

**CTU FEE** Czech Technical University Faculty of Electrical Engineering

