DEGREE PROJECT, IN COMPUTER SCIENCE , FIRST LEVEL
STOCKHOLM, SWEDEN 2015

An Implementation and Evaluation of the Least-effort Approach of Crowd Simulation

YANG ZHOU

KTH ROYAL INSTITUTE OF TECHNOLOGY

COMPUTER SCIENCE AND COMMUNICATION

KTH Computer Science
and Communication

An Implementation and Evaluation of the Least-effort Approach of Crowd Simulation

YANG ZHOU

Degree Project in Computer Science, DD143X
Supervisor: Alexander Kozlov
Joint Advisor: Christopher Peters
Examiner: Örjan Ekeberg

Stockholm, Sweden 2015

Abstract

Crowd simulation serves as an important tool in architecture, where efficiency and safety could be improved with the knowledge of how large groups of people behave, and computer graphics as well as entertainment industry, where demands of animating large amounts of avatars exist. This project has implemented and evaluated an approach of crowd simulation based on the *Principle of Least Effort*, a fundamental rule of human behavior. The approach is capable of simulating thousands of agent in real time, and can be parallelized naturally to utilize the power of multiprocessor.

The approach has been implemented using C++ and OpenMP. Results show that the approach generates smooth, collision-free, and visually plausible agent trajectories. To evaluate the approach in a quantitative manner, a set of metrics have been defined, and a set of test cases have been selected. By comparing the approach with RVO, a similar approach that does not consider the *Principle of Least Effort*, the evaluation shows that optimization based on the principle leads to agent trajectories that cost less effort and time. In further case studies, the approach has been proved to be able to generate a number of emergent phenomena verified in real crowd.

Contents

1 Introduction **1**

2 Background **3**
2.1 State-of-the-art . . . 3
2.2 The Principle of Least Effort (PLE) . . . 4
2.3 Evaluation Metrics . . . 5
2.4 Terminology . . . 5

3 Method **7**
3.1 The Least Effort Approach . . . 7
3.1.1 Effort Function . . . 7
3.1.2 Optimization Strategy . . . 8
3.2 Local Avoidance . . . 8
3.3 Optimization via Geometric Method . . . 11
3.4 Global Planner . . . 16
3.5 Acceleration via Bounding Volume Hierarchy . . . 17

4 Results **21**
4.1 Implementation Setup . . . 21
4.2 Metrics . . . 23
4.3 Test Cases . . . 24
4.4 Case Studies . . . 25
4.4.1 Metric Scores . . . 25
4.4.2 *Oncoming* . . . 27
4.4.3 *N-agent-circle* . . . 27
4.4.4 *Bottleneck-corridor* . . . 29
4.4.5 *Downtown* . . . 30
4.5 Showcases . . . 32

5 Discussion and Conclusion **36**
5.1 Relation between PLE and RVO . . . 36
5.2 Limitations . . . 37
5.3 Future Work . . . 38
5.4 Conclusion . . . 38

List of Figures

3.1 Two agents interact with each other. 9
3.2 *Velocity Obstacle*. In this figure, τ is larger than 1. Note the tangent points on two circles. 9
3.3 Construct $ORCA$ geometrically. 11
3.4 Optimization within PV. This figure illustrates the case where $\mathbf{v}_{ideal} \notin PV$. Dots on the edges of PV are candidates of $\mathbf{v}_{curr}$, and the red dot is the true $\mathbf{v}_{curr}$. 12
3.5 C_1 intersects C_2. Here, B does not point toward to A. Neither does D because it already intersects A. However, C does point toward A. . . . 13
3.6 A simple instance of BVH. 18

4.1 Screenshot of simulation application. Colored dots represent agents and arrows represent their trajectories. An obstacle is placed at the center. . 22
4.2 Average step time. 23
4.3 Agent trajectory in *Oncoming* test case. The blue circle indicates the initial position of agent 1, and the red circle indicates the initial position of agent 2. The size of circles does not represent the size of agents. . . . 28
4.4 Screenshots of simulating *N-agent-circle* where $N = 100$, circle radius $R = 25m$, and agent radius $r = 0.5m$. 28
4.5 Growth of *average time* and *average accumulative penetration* along with the increase of agent number where circle radius $R = 25m$ and agent radius $r = 0.5m$. 29
4.6 Screenshots of simulating *Corridor* 30
4.7 *Edge effect*: average speed distribution along Y position from $40th$ to $90th$ step, where X position $\in [-30, -20]m$. 31
4.8 Average Effort in *Downtown*. 32
4.9 *Finger effect*. Agents tend to form into lines (marked by white segments). 33
4.10 Screenshots of simulating *Downtown*. Red boxes mark congestion areas. 34
4.11 Demonstration of the PLE method implemented in the 3D scenario featuring a virtual model of KTH. Figures (a) - (f) demonstrate a variety of cases from Section 4.5. 35

List of Tables

4.1 Average step time (s) 22
4.2 Metric scores achieved by PLE and RVO in all test cases. 26
4.3 Parameters for PLE and RVO. 26

Chapter 1

Introduction

Crowds are basic phenomena of human society. The behavior of crowds has been studied extensively for many reasons, among which a typical one is to develop guidelines for designing facilities and planning. For instance, it is useful to predict how groups of people tend to enter or leave an olympic arena when designing the structure of it, and in urban planning, the knowledge of pedestrian behavior increases the efficiency of roadways as well as reduces loss under emergency.

As an important experiment method, computer simulation involving aggregation of individuals has been created by researchers from various fields such as architectural design, urban planning, computer graphics, physics, robotics, public security, training systems, and sociology. Crowd simulation also finds itself broad application in entertainment industry. In movie production and computer games, large numbers of characters need to be animated. The effect of simulation greatly influences the realism of scene and the experience of interaction.

Crowd simulation is the process of simulating large numbers of humans, or agents, in one environment. The task is challenging because human crowds exhibit highly complex behavior driven by individual decisions of agents with respect to their goals, obstacles, and other nearby agents. There exist numbers of approaches, yet most of them are domain-specific, focusing on different aspects of the collective behavior, and using different modeling techniques. Employed techniques range from sociological simulations, to behavioral models, those that represent each individual as being controlled by more or less complex rules based on physical laws, or those that do not distinguish individuals such as flow and network models.

This project is thus motivated by the broad application and the challenging nature of crowd simulation.

Problem Statement

The main purpose of this project is to explore how virtual agents should move around, reach their goals, and avoid collisions with other agents and obstacles in an environment. The problem is addressed in two parts: an algorithm to simulate

the crowd behavior as well as evaluation on the algorithm that quantizes the performance the algorithm. In addition, this project does not focus on high-quality animation effect of individual model.

The algorithm this project elaborates on is based on the "PLEdestrians" proposed by Stephen. J. Guy *et al.*[1]. There are two reasons for choosing it. First, It computes a natural trajectory that is in most cases collision-free for each agent in crowd. Second, it is efficient enough to simulate a large amount of agents in real time. It is also possible to parallelize the method for further acceleration.

The evaluation of simulated crowd behavior remains a relatively unexplored research topic, yet its importance is self-evident. Besides visual observation, this project intends to benchmark the algorithm by Steerbench[2], a toolbox that consists of a set of typical test cases and mathematical metrics. Also, a number of representative case studies are carried out, serving as combination of qualitative and quantitative evaluation to identify and analyze the simulated behavior.

Visualization is used to help demonstrate simulation results. A real-time demonstration is made in which agents update their trajectories and velocities in predefined scenarios.

Hypothesis

The main hypothesis of this project is that human base their movement on the *Principle of Least Effort* (which is described in section 2.2). If it is true, adopting method following the principle should be able to generate visually plausible movement of each virtual agent. Also, it is expected to observe similar aggregated behaviors between simulated crowd and real crowd.

Thesis Structure

The remaining of this thesis is organized as follows. Chapter 2 reviews the state-of-the-art of crowd simulation techniques and evaluation approaches. As the theoretical foundation of this project, the *Principle of Least Effort* is also introduced in this chapter. Chapter 3 gives detailed description of the approach taken in this project, that is to represent human effort in a mathematical form, to optimize the effort, and to combine the optimization with local avoidance strategy. Chapter 4 presents the implementation of the method and the obtained result. The method is evaluated based on a number of metrics in a set of selected test cases, and four detailed case studies are carried out to further analyze the method. Chapter 5 summarizes the performance of the method including its strengths and what could be done for improvement.

Chapter 2

Background

This chapter first reviews the state-of-the-art of crowd simulation approaches. In particular, *the Principle of Least Effort* is described. Next, the existing metrics for evaluating agent steering behavior and crowd behavior are described. Finally, the terminology used in the thesis is listed.

2.1 State-of-the-art

The short research history leads to the fact that the techniques for crowd simulation are largely stimulated by knowledge from other areas. A large category among them is the force-field models, in which the motion and interaction between agents are based on analogies with physics. As a famous representative, the social force model by Helbing *et al.* [3, 4] proves that it is possible to describe crowd behavior by treating pedestrians as particles with potentials and introduce socio-psychological forces that reflect pedestrians' tendencies. The method is capable of describing the self-organization and several observed collective effects of pedestrian behavior. The social force model has been extended by various means, such as including individualism [5] or combining it with psychological and geometric rules [6] (the HiDAC system).

In contrast with the force-field models, behavioral models use sets of goal-oriented rules to define individual behaviors. Behavioral animation was pioneered by Reynolds [7], who has simulated flocks of birds and schools of fish. In the system, complex global behaviors arise from the introduction and interaction of simple local rules, such as "steer to avoid crowding local mates", or "steer towards the average direction of local mates". However, the result of Reynolds' method was confined to relatively small, low-density groups of animals.

It is possible to simulate crowds either in a continuous manner or a discrete one. The former is usually based on continuum theories of flow or fluid dynamics [8, 9] as in extremely high density condition, crowd exhibits similar properties to fluid. In the model presented by Adrien Treuille *et al.*, [8] a dynamic potential field integrates global navigation with moving obstacles, efficiently solving the motion of

large crowds without the need for explicit collision avoidance. In the latter case, cellular automata is used as an computation model in which agents move between cells [10, 11]. Models based on cellular automata are fast and relatively easy to implement, but they tend to expose blocky artifacts as the world space is discretized into cells.

Most of the above techniques model each individual of a crowd equally, yet researches have also been done to model the heterogeneous features of crowd, which greatly contribute to perceptual realism. It is necessary to consider personality and other psychological factors to simulate heterogeneous crowds. To do so, Pelechano *et al.* introduced personality variables that represent physiological and psychological factors observed in real people [6], which were further varied by the OCEAN personality model [12]. Guy used personality trait theory and derived a mapping from crowd simulation parameters to the perceived behaviors of virtual agents [13].

Finally, crowd simulation obtains useful guides from studies on real-world crowd dynamics [14, 15]. These studies have summarized characteristics of human motion in crowds. An individual tends to seek his or her goal, and naturally reach the goal along a path that requires the least effort [15]. The concept, named *the Principle of Least Effort*, encourages people to take shortest route while moving at their preferred speed. By representing "effort" as a function and minimizing it, agents are expected to travel in smooth trajectories, and take fewer adjustments to avoid collisions. In his phd thesis, G.K.Still used a software collection named *Legion* that solved the problem by a type of simulated annealing, yet the detail is not given due to a commercial non disclosure agreement [15].

2.2 The Principle of Least Effort (PLE)

First proposed by George. K. Zipf [16], the PLE is a general theory in anthropology and psychology which suggests that people choose their entire behavior based on the minimization of effort. Researches have been done to successfully explain human locomotion patterns by applying the PLE [17]. Two algorithmic rules of human movement could be derived from the PLE [15]:

- A person tends to reach his or her goal via the shortest path.
- A person tends to move at his preferred velocity, and tends not to change the velocity frequently.

The PLE could be utilized to simulate crowd behavior. If the effort of an individual could be quantized and represented as a function of his or her trajectory, then by optimizing the function it is possible to find out the trajectory that matches the PLE. In this way the problem of crowd simulation is transformed to the problem of optimizing the effort function. The least effort approach thus includes two components: an effort function and an optimization strategy. They are described in detail in chapter 3.

2.3 Evaluation Metrics

Evaluation metrics are critical to the development of crowd simulation since they provide the direction of improvement. However, it is challenging to evaluate crowd simulation techniques due to several reasons. First, the overall aggregate behavior of the system is contributed by large numbers of individuals, the movement patterns of which are often stochastic. Even the same group of people in the same environment rarely generate identical behaviors twice, though they are very similar. Second, a crowd system may run in diverse scenarios. On the one hand, it is desirable to analyze its performance comprehensively, yet on the other hand, it is infeasible to test all infinite possible scenarios.

Pelechano *et al.* proposed the *level of presence* as a metric for virtual crowd behavior [18] . The method requires human participation. Volunteers interact with virtual crowd members in an immersive virtual environment for the purpose of studying presence in virtual crowds.

Steerbench [2] provides a set of typical benchmarking test cases and scoring rules. The test cases includes both those for evaluating individual behavior such as an agent moving along a curve, and those for evaluating aggregate behavior such as hundreds of agents evacuating from a room or squeezing into a narrow channel. The primary rule used in benchmarking combines the number of collisions, time efficiency and effort efficiency to compute a score of the performance of the tested technique in specific test cases, which could be used for comparison between techniques.

As an alternative that introduces real-world data, *Entropy Metric* by Stephen J. Guy *et al.* uses a two-step process to quantify a simulator's ability to reproduce the collective behaviors of the whole system which is observed in the recorded real-world data [19]. First, Bayesian inference is used to estimate the simulation states which best correspond to the observed data. The prediction error is then estimated via the EM-algorithm. The metric is able to measure the similarity between simulation results and real-world data, and it also accounts for noises.

The PLE itself also serves as an evaluation metric since it suggests that realistic agent behaviors should have lower effort. The least effort approach is thus expected to outperform other techniques on this metric.

Human perception of crowd realism has been studied to improve both crowd behaviors and rendering. McDonnell *et.al* studied people's perception of variety in crowds, and argued that clones of appearance are far easier to detect than motion clones [20]. They further utilized perceptual saliency to identify important features that need to be varied to add visual variety to virtual characters [21].

2.4 Terminology

Agent: A simplified circle model that represents an individual, defined in $\mathbb{R}^2$ space. An agent includes several parameters:

- Position $\mathbf{p}$: the vector that indicates the current position of the agent.

- Velocity $\mathbf{v}$: the vector that indicates the current velocity of the agent.
- Maximum Speed v_m: the upper limit of the speed of the agent, *i.e.*, $|\mathbf{v}| \leq v_m$.
- Radius r: the radius of the particle.
- Goal $\mathbf{g}$: the vector that indicates the goal position of the agent.

Crowd: A set of finite agents that may interact with each other, denoted as $\mathbf{X}$. The ith agent in $\mathbf{X}$ is denoted as $\mathbf{X}_i$. The state of a crowd varies along time, and the snapshot at time t is denoted by $\mathbf{X}^t$.

Obstacle: An obstacle is a polygon defined in $\mathbb{R}^2$ space. Normally, an agent should not collide with, *i.e.*, intersect any obstacle. For simplicity, if there is no special statement, an obstacle is limited to convex polygon, as this is enough for most applications.

PLE: For convenience, in the remaining of this thesis, the term PLE may either refer to the *Principle of Least Effort*, or the crowd simulation method based on the principle described in chapter 3.

With the background knowledge in this chapter, the next chapter explains the PLE-based method in this project.

Chapter 3

Method

This chapter describes how to simulate and evaluate crowd behaviors. The simulation proceeds in a discrete manner, and time is discretized to small steps. At each step, an agent computes its new velocity that minimizes the effort function based on its current position, velocity, and information from neighborhood under the premise of avoiding collisions. The position of each agent will then be updated based on the new velocity. The process terminates once all agents have reached their goals. Section 3.1 introduces the effort function as well as a greedy optimization strategy. Section 3.2 introduces the local avoidance method adopted in this project. Section 3.3 gives a detailed description on how to combine the ideas from 3.1 and 3.2. Section 3.4 introduces the global planner that navigates agents in complex environments in an effort-efficient manner.

3.1 The Least Effort Approach

3.1.1 Effort Function

A well-defined effort function should reflect the two algorithmic rules derived from the PLE (section 2.2). Stephen. J. Guy *et al.* present an effort function based on biomechanical principles [1]:

$$\begin{aligned} E &= \int P(t)dt \\ P(t) &= m(e_s + e_w|\mathbf{v}(t)|^2) \end{aligned} \tag{3.1}$$

where E denotes the energy, or effort, spent during travel period, $P(t)$ denotes the instantaneous power, m denotes the mass of the individual and $\mathbf{v}(t)$ denotes the velocity of the individual, which may vary over time. e_s and e_w are two constants reflecting individuals' different biomechanical properties. It can be shown that traveling along the shortest path with the constant speed $\sqrt{e_s/e_w}$ minimizes the effort function, which is consistent with the PLE: move through the shortest path and move at preferred speed. On the other hand, if the path length is fixed to L,

the minimal energy spent to traverse it is

$$min(E) = 2mL\sqrt{e_s e_w} \tag{3.2}$$

3.1.2 Optimization Strategy

It is difficult to directly minimize the effort function. Instead, an efficient greedy strategy is developed to simplify the function [1]. In order to compute the velocity for current step that lasts τ seconds , the greedy strategy decomposes the whole travel of an agent from $\mathbf{p}$ to $\mathbf{g}$ into two components: the component during the current time step and the remaining component. Therefore the effort spent in both components need to be computed respectively, before they are added together. Because τ is so small that the velocity $\mathbf{v}$ of the agent can be considered as nearly constant, the former contribution is easy to obtain. For the latter contribution, the smallest possible value is simply assumed, as the strategy is called "greedy": only current step component is cared. In this way, the length of shortest path between the position after τ seconds and the final goal is chosen, and $\sqrt{e_s/e_w}$ is set as moving speed. The simplified effort function is represented as

$$\begin{aligned} E = & \underbrace{m(e_s + e_w|\mathbf{v}|^2)\tau}_{\text{current step component}} + \underbrace{2m\sqrt{e_s e_w} \cdot |\mathbf{g} - (\mathbf{p} + \mathbf{v}\tau)|}_{\text{the remaining component}} \\ \mathbf{v}_{curr} = & \underset{\mathbf{v}}{\operatorname{argmin}}(E) \end{aligned} \tag{3.3}$$

Standard approach is then applied to obtain the desired velocity, $\mathbf{v}_{curr}$, for current time step by setting the gradient of $\mathbf{v}$ to zero and solving the equations.

3.2 Local Avoidance

Local Avoidance is an essential feature of virtual agents. It is natural for human to avoid upcoming collision against obstacles or other humans. From the perspective of minimizing effort, collisions are also undesirable because they cause abrupt changes of agent velocity and trajectory, which would much increase the total effort.

In the previous section, the agent velocity for current time step $\mathbf{v}_{curr}$ is optimized within the whole velocity space. This, however, is only feasible when there is no other agents or obstacles in surrounding. In general cases, an agent must choose velocity that would not lead to collisions. To find out the feasible velocity space, the *Reciprocal Velocity Obstacle* (RVO) method proposed by Jur van den Berg *et al.* [22] is adopted. The method can be seen as incremental because it only takes care of possible collisions in the following τ seconds. For each agent, its neighboring agents and obstacles add linear constraints to its feasible future velocity in the velocity space. All linear constraints thus form a convex region.

Consider two agent A and B, with positions, velocities, and radii $(\mathbf{p}_a, \mathbf{v}_a, r_a)$ and $(\mathbf{p}_b, \mathbf{v}_b, r_b)$ respectively (figure 3.1). The region in velocity space within which

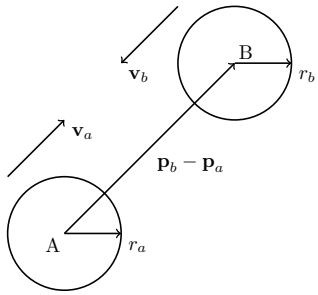


Figure 3.1: Two agents interact with each other.

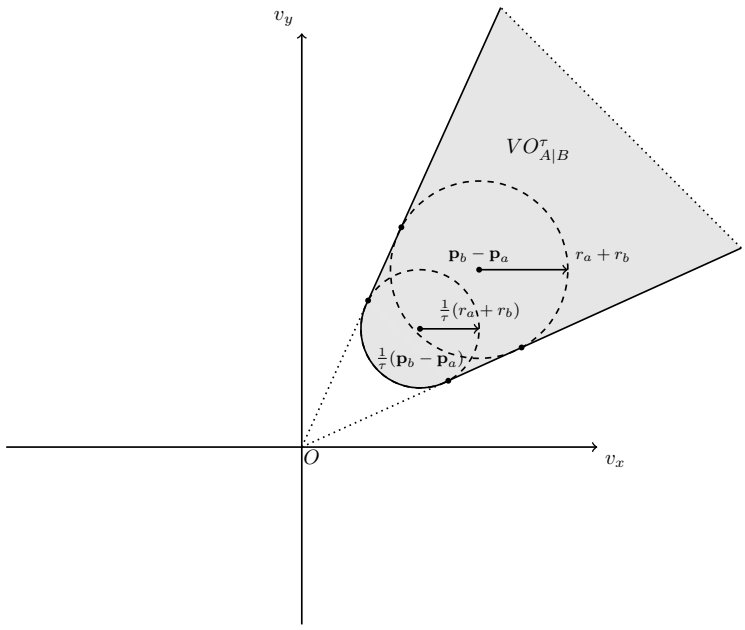


Figure 3.2: *Velocity Obstacle*. In this figure, τ is larger than 1. Note the tangent points on two circles.

any relative speed $\mathbf{v}_r = \mathbf{v}_a - \mathbf{v}_b$ would cause a collision between A and B within the time interval τ could be defined as

$$VO^{\tau}_{A|B} = \{\mathbf{v}_r | \exists t \in [0, \tau], \mathbf{v}_r t \in D(\mathbf{p}_b - \mathbf{p}_a, r_a + r_b)\} \tag{3.4}$$

where $D(\mathbf{p}, r) = \{\mathbf{x} | \, |\mathbf{x} - \mathbf{p}| < r\}$ specifies an open circular region centered at p with radius r. Such region is named *Velocity Obstacle* (VO) [22]. Figure 3.2 illustrates the concept of VO. On the other hand, if $\mathbf{v}_r \notin VO^{\tau}_{A|B}$, A will not collide with B at least within the time interval τ. More generally, if $\mathbf{v}_b \in V_b$, $\mathbf{v}_a$ should not fall into any $VO^{\tau}_{A|B}$ specified by any $\mathbf{v}_b$. In this case, the feasible region of collision avoidance velocity, could be defined by using dilation operation:

$I \bigoplus H = \{(\mathbf{p}+\mathbf{q})|\mathbf{p} \in I, \mathbf{q} \in H\}$.

$$CA^{\tau}_{A|B}(V_b) = \{\mathbf{v}_a|\mathbf{v}_a \notin VO^{\tau}_{A|B} \bigoplus V_b\} \tag{3.5}$$

The case is symmetric for $\mathbf{v}_b$ given $\mathbf{v}_a \in V_a$:

$$CA^{\tau}_{B|A}(V_a) = \{\mathbf{v}_b|\mathbf{v}_b \notin VO^{\tau}_{B|A} \bigoplus V_a\} \tag{3.6}$$

If both the velocities of A and B are respectively limited within V_a and V_b, and satisfy $V_a \subseteq CA^{\tau}_{A|B}(V_b)$ and $V_b \subseteq CA^{\tau}_{B|A}(V_a)$, then it is sufficient to say that A and B are free from collision in the upcoming time interval τ.

The feasible region of collision avoidance velocity is usually not unique. It is thus desirable to obtain the maximal one so that an agent has more choices about its following action, and has more choices to minimize the effort function. The optimal pair of V_a and V_b should be $V_a = CA^{\tau}_{A|B}(V_b)$ and $V_b = CA^{\tau}_{B|A}(V_a)$. Moreover, they should contain more velocities close to the current velocities $\mathbf{v}_a$ and $\mathbf{v}_b$, respectively, than any other pairs. Such pair of V_a and V_b is defined as *Optimal Reciprocal Collision Avoidance* $ORCA^{\tau}_{A|B}$ and $ORCA^{\tau}_{B|A}$:

- $ORCA^{\tau}_{A|B} = CA^{\tau}_{A|B}(V_b)$ and $ORCA^{\tau}_{B|A} = CA^{\tau}_{B|A}(V_a)$
- $\left|ORCA^{\tau}_{A|B} \cap D(\mathbf{v}_a, r)\right| = \left|ORCA^{\tau}_{B|A} \cap D(\mathbf{v}_b, r)\right| \geq \max(|V_a \cap D(\mathbf{v}_a, r)|, |V_b \cap D(\mathbf{v}_b, r)|)$

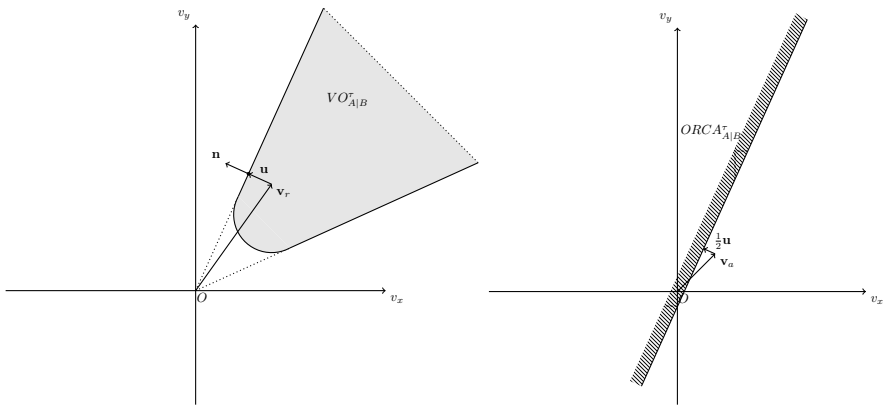
It can be shown that the $ORCA^{\tau}_{A|B}$ essentially adds a linear inequality constraint to the velocity space of A, and can be constructed geometrically [22]. Assume the current velocities $\mathbf{v}_a$ and $\mathbf{v}_b$ would cause a collision in τ, *i.e.*, $\mathbf{v}_r = \mathbf{v}_a - \mathbf{v}_b \in VO^{\tau}_{A|B}$. Let vector $\mathbf{u}$ start from $\mathbf{v}_r$ to the nearest point on the boundary of $VO^{\tau}_{A|B}$, and $\mathbf{n}$ be the outward-pointing normal on that point. $\mathbf{u}$ is the smallest change in $\mathbf{v}_r$ to avoid the upcoming collision (figure 3.3a). To avoid the collision in a reciprocal manner, A only takes a half of the responsibility, while the other half is left to B. The $ORCA^{\tau}_{A|B}$ is thus a half-plane starting at $\mathbf{v}_a + \frac{1}{2}\mathbf{u}$ and pointing at the direction of $\mathbf{n}$ (3.3b). The $ORCA^{\tau}_{B|A}$ can be constructed symmetrically.

$$ORCA^{\tau}_{A|B} = \{\mathbf{v}|[\mathbf{v} - (\mathbf{v}_a + \frac{1}{2}\mathbf{u})] \cdot \mathbf{n} \geq 0\} \tag{3.7}$$

In cases where $\mathbf{v}_r \notin VO^{\tau}_{A|B}$, the $ORCA^{\tau}_{A|B}$ has the same form, yet with slightly different meaning. In former cases, $\mathbf{v}_a$ must change to fall in the $ORCA^{\tau}_{A|B}$ to avoid collision, while in latter cases, it is safe to keep $\mathbf{v}_r$, and the $ORCA^{\tau}_{A|B}$ only provides the allowed scope if A needs to change its velocity. Due to the linear characteristic of ORCA, the optimization of agent velocity is greatly simplified by using a geometric method, which is introduced in section 3.3.

The concept of VO can be extended to help an agent avoid static obstacles. Consider an agent A and a static obstacle O. Since the velocity of O is always zero, the relative velocity is simply $\mathbf{v}_a$, and the $VO^{\tau}_{A|O}$ can be calculated as

$$VO^{\tau}_{A|O} = \{\mathbf{v}|\exists t \in [0, \tau], \mathbf{v}t \in -D(\mathbf{p}_a, r_a) \bigoplus O\} \tag{3.8}$$



(a) (b)

Figure 3.3: Construct $ORCA$ geometrically.

Any velocity that does not belong to $VO^{\tau}_{A|O}$ is safe to choose. However, under the consideration of computational efficiency, the constraint from O is preferred to be also linear. Similar to ORCA, a half-plane could be constructed as $\{\mathbf{v}|(\mathbf{v}-\mathbf{v}_n)\cdot\mathbf{n} \geq 0\}$. Here, $\mathbf{v}_n$ is the nearest point to the origin on $VO^{\tau}_{A|O}$, and $\mathbf{n}$ is the outward-pointing normal at $\mathbf{v}_n$.

3.3 Optimization via Geometric Method

For an agent A and its neighbor B (or obstacle O), The local avoidance method suggests a half-plane and A must update velocity within it. Suppose that there are n neighboring agents, from B_1 to B_n, and m obstacles, from O_1 and O_m. The intersection of all half-planes forms a feasible region. The optimization of the effort function must be performed within the region to avoid collisions. The problem could be formalized as follows:

$$\begin{aligned}
&E(\mathbf{v}) = m(e_s + e_w|\mathbf{v}|^2)\tau + 2m\sqrt{e_s e_w}\cdot|\mathbf{p}+\mathbf{v}\tau, \mathbf{g}| \\
&\mathbf{v}_{curr} = \underset{\mathbf{v}}{\operatorname{argmin}}(E) \\
&\text{subject to } \mathbf{v}_{curr} \in PV \equiv \bigcap_{i=1}^{n} ORCA^{\tau}_{A|B_i} \cap \bigcap_{j=1}^{m} ORCA^{\tau}_{A|O_i}
\end{aligned} \tag{3.9}$$

Figure 3.4 illustrates the situation. It can be proved that the effort function $E(\mathbf{v})$ is convex in the velocity space, and has unique global minima $\mathbf{v}_{ideal}$ that is tangent to the shortest trajectory with the magnitude of $\sqrt{e_s/e_w}$. Moreover, since a half-plane is convex, and the intersection of convex sets is again a convex set, the intersection of half-planes is a convex region. The convex characteristic greatly simplifies the problem, leading the minima to be [1]:

- if $\mathbf{v}_{ideal} \in PV$, then $\mathbf{v}_{curr} = \mathbf{v}_{ideal}$

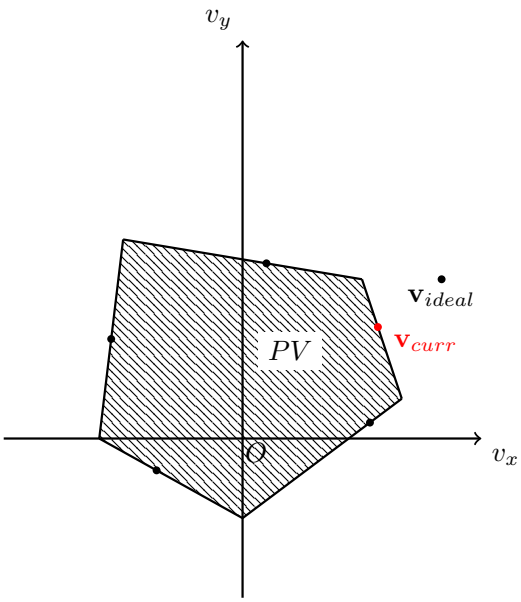


Figure 3.4: Optimization within PV. This figure illustrates the case where $\mathbf{v}_{ideal} \notin PV$. Dots on the edges of PV are candidates of $\mathbf{v}_{curr}$, and the red dot is the true $\mathbf{v}_{curr}$.

- else, $\mathbf{v}_{curr}$ lies on the boundary of PV

It is easy to determine which is the case by checking whether $\mathbf{v}_{ideal}$ lies in PV. The solution for the first case is trivial. For the second case, one could first calculate the boundary of PV by using half-plane intersection algorithm, and then minimizing the effort function on each edge before sorting all minimum on edges to obtain the final minima. By applying a divide-and-conquer strategy, the common intersection of a set of n half-planes can be computed in $O(n \log n)$ time [23]. The algorithm recursively divides the set of half-plane into pairs of subsets and computes the intersection of subset pair. The following pseudocode illustrates the procedure.

Algorithm 1 HALFPLANEINTERSECTION(H)

Input: A set H of n half-planes in the plane
Output: The convex region $C := \bigcap_{h \in H} h$

```
if card(H)=1 then
    C ← the unique half-plane h ∈ H
else
    Split H into H1, H2 where card(H1) = ⌈n/2⌉ , card(H2) = ⌊n/2⌋
    C1 ← HalfplaneIntersection(H1)
    C2 ← HalfplaneIntersection(H2)
    C ← ConvexPolygonIntersection(C1,C2)
end if
```

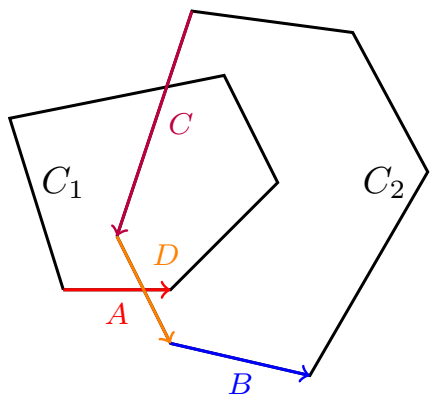


Figure 3.5: C_1 intersects C_2. Here, B does not point toward to A. Neither does D because it already intersects A. However, C does point toward A.

As the key sub-procedure of the half-plane intersection algorithm, ConvexPolygonIntersection computes the intersection of two convex polygons (Note that it is not necessary to take infinite regions into consideration in practice). There are various linear time algorithms for convex polygon intersection, and the one proposed by O'Rourke and his students is adopted due to its simplicity in implementation [24].

Consider two convex polygons C_1 and C_2 (whose vertices are organized in counter-clockwise order). Let A and B be directed edges on each polygon. The basic idea is to assign two edges arbitrarily at the beginning and let them "chase" each other, traversing all vertices of the intersection area, before they go through one whole cycle. At each step, either A or B is advanced based on their geometric relation. If A "points toward" the line containing B, yet does not intersect with it, then A should be advanced in order to get close to an expected intersection. The similar applies to B. If A and B both satisfy the condition, either one can be chosen. Figure 3.5 demonstrates this "point-toward" condition. Let a and b denote the head of A and B respectively, $H(A)$ and $H(B)$ denote the open half-planes on the left side of A and B (to be consistent with the counter-clockwise polygon definition), the "point-toward" condition could be formally described as

$$\begin{aligned} &A \text{ points toward } B\text{: } (A \times B > 0 \text{ and } a \in H(B)) \text{ or } (A \times B < 0 \text{ and } a \notin H(B)) \\ &B \text{ points toward } A\text{: } (B \times A > 0 \text{ and } b \in H(A)) \text{ or } (B \times A < 0 \text{ and } b \notin H(A)) \end{aligned} \tag{3.10}$$

If neither satisfies the condition, the procedure advance whichever has its head outside the half-plane of the other, or either one if they are both outside. Note that the rule above covers all possible cases because if both a and b are inside the half-plane of the other, then one must point toward the other.

According to definition, the intersection polygon is either a part of C_1 or C_2. The above advance rule guarantees that only A or B could be a part of the intersection

polygon. Therefore, an *inside* flag indicating whether A or B is inside should be maintained so that after advancing an edge, the new head is added to the intersection polygon if it is inside.

The following pseudocode illustrates the algorithm. It can be seen that the algorithm has $O(n+m)$ complexity, where n and m are the edge numbers of A and B.

Algorithm 2 ConvexPolygonIntersection(C_1,C_2)

Input: Two convex polygons C_1 and C_2
Output: The intersection polygon $C := C_1 \cap C_2$

```
Choose arbitrary edge A from C1 and B from C2, with head a and b respectively

repeat
  if A intersects B then
    if the intersection point has already been added to C then
      break
    end if
    Add the intersection point to C
    if a is inside then
      inside ← a
    else
      inside ← b
    end if
  end if
  Advance()
until A cycles C1 and B cycles C2

Advance()
if A points toward B then
  Advance A
else if B points toward A then
  Advance B
else if a ∉ H(B) then
  Advance A
else
  Advance B
end if
```

After obtaining the boundary of PV, optimization should be applied to each segment of the boundary. Let the line equation be $v_y = mv_x + e$. Eqn 3.3 could

thus be parameterized by v_x, and its derivative becomes

$$\begin{aligned}
\frac{\partial E}{\partial v_x} &= a_4 v_x^4 + a_3 v_x^3 + a_2 v_x^2 + a_1 v_x + a_0 \\
a_4 &= e_w A^3 \\
a_3 &= e_w (2me + B)^3 \\
a_2 &= A(e_w m^2 e^2 + 2e_w meB + e_w AC - e_s A) \\
a_1 &= e_w Bm^2 e^2 + 2e_w meAC - e_s AB \\
a_0 &= e_w m^2 e^2 C - \frac{1}{4} e_s B^2 \\
A &= 1 + m^2 \\
B &= -2(d_x + md_y - me) \\
C &= d_x^2 + d_y^2 - 2ed_y + e^2 \\
\mathbf{d} &:= (d_x, d_y) = \frac{1}{\tau}(\mathbf{g} - \mathbf{p})
\end{aligned} \tag{3.11}$$

In vertical cases, v_x becomes constant, and Eqn 3.3 is instead parameterized by v_y:

$$\frac{\partial E}{\partial v_y} = v_y^4 - 2d_y v_y^3 + [(d_x - v_x)^2 + d_y^2 - \frac{e_s}{e_w}]v_y^2 + 2\frac{es}{ew} d_y v_y - \frac{es}{ew} d_y^2 \tag{3.12}$$

Since the derivative of Eqn 3.3 is quartic, it can be solved by iterative methods quickly.

To summarize, the pseudocode of the whole procedure is given below. Since each agent updates its current velocity only based on the status of other agents in the last time step, it does not interfere others' update procedure. Thus, it is easy to accelerating the algorithm by parallelizing the for loop in CrowdSimulation.

Algorithm 3 CrowdSimulation($\mathbf{X}, S$)

Input: A crowd $\mathbf{X}$, and a scene S consists of a set of obstacles

```
t ← 0
repeat
  for all X_i ∈ X^t do
    if X_i has not reached its goal then
      query the neighboring agents B ⊆ X^t and obstacles O ⊆ S of X_i
      compute ORCA half-plane set H for B and O
      PV ← HalfplaneIntersection(H)
      v_{X_i^{t+Δt}} ← argmin_v(E) s.t. v ∈ PV
      p_{X_i^{t+Δt}} ← p_{X_i^t} + v_{X_i^{t+Δt}} · Δt
    end if
  end for
  t ← t + Δt
until ∀X_i ∈ X^t , X_i has reached its goal
```

3.4 Global Planner

In the above sections, the distance from an agent's position to its goal is simply represented as the straight line distance. This is, however, not correct in complicated environments with obstacles. In addition, an agent is not capable of navigating itself to the goal in a complicated environment with only local collision avoidance module. A typical example is a U-shaped obstacle. Thus, a global planner is needed to provide a high-level path to agent, ensuring that agent could reach its goal through the path.

In the global planner, the configuration space is first computed based on the obstacles in the environment. A visibility graph is then generated, serving as a map for agents. Note that the visibility graph should have been precomputed before the simulation starts since obstacles are static. At each step of simulation, every agent computes a path from its current position to its goal based on the visibility graph. Here, the standard A* algorithm is applied for efficient computation. An agent then can obtain the next intermediate path node, and move to the node governed by the least effort method and local avoidance method. The agent repeats the procedure until it finally reaches goal.

Congestion avoidance is an important issue related to the global planner. The global planner is desired not to provide agents the same path or paths that will intersect at the same time. Otherwise, a large number of agents will follow the same path, or encounter each other at the same time, which is largely likely to cause a congestion. To avoid formation of congestion and navigate in an effort-efficient manner, Guy suggests a heuristic of weight assignment [1]. Instead of simple euclidean distance, the weight of an edge is better to be an estimation of the effort needed to traverse the edge. Assume the length of edge is L, the weight is given by

$$w = L(\frac{e_s}{|\mathbf{v}_{avg}|} + e_w|\mathbf{v}_{avg}|) \tag{3.13}$$

where $|\mathbf{v}_{avg}|$ is the average agent speed along the edge. If $|\mathbf{v}_{avg}|$ is low, it indicates a local congestion on that edge and larger effort, and w becomes larger to prevent agent to choose it as part of its path. If $|\mathbf{v}_{avg}|$ approximates $\sqrt{e_s/e_w}$, the edge is free from congestion, and w remains small. To compute $\mathbf{v}_{avg}$, the global planner also needs to maintain an list for each edge holding all agents on it.

It is also possible to use other map representation than visibility graph, such as grid-based representation or navigation mesh. The visibility graph is chosen because it is easy to implement, and it does not produce grid-like artifacts.

Now that the global planner has been introduced, the algorithm CrowdSimulation is modified to include it.

Algorithm 4 CROWDSIMULATION($\mathbf{X}, S$) (including global planner)

Input: A crowd $\mathbf{X}$, and a scene S consists of a set of obstacles

```
compute a visibility graph G based on S
t ← 0
repeat
  update the edge weights of G
  for all X_i ∈ X^t do
    if X_i has not reached its goal then
      search path from p_{X_i^t} to g_{X_i}, query next path node
      query the neighboring agents B ⊆ X^t and obstacles O ⊆ S of X_i
      compute ORCA half-plane set H for B and O
      PV ← HALFPLANEINTERSECTION(H)
      v_{X_i^{t+Δt}} ← argmin_v(E) s.t. v ∈ PV
      p_{X_i^{t+Δt}} ← p_{X_i^t} + v_{X_i^{t+Δt}} · Δt
    end if
  end for
  t ← t + Δt
until ∀X_i ∈ X^t , X_i has reached its goal
```

3.5 Acceleration via Bounding Volume Hierarchy

An agent needs to consider all its neighbors in order to update its velocity. It is not hard to recognize that the current complexity of the algorithm CROWDSIMULATION is $O(N^2)$, where N is the total agent number. The quadratic complexity becomes unacceptable as agent number grows. However, there remains space for acceleration. With the prediction interval τ, the max speed of agent v_m, and the max radius r_m given, all other agents outside the range $v_m t + 2r_m$ can be safely ignored. By introducing an space partition data structure for range query, the brute-force $O(N^2)$ complexity could be avoided.

A bounding volume hierarchy (BVH) is thus maintained for efficient range query. A BVH is a recursively defined tree structure. Every tree node has its own bounding volume, and the bounding volume of a parent node always contains all bounding volumes of its children nodes. Only leaf nodes store object information, while internal nodes only store volumes. There are several common types of bounding volume, such as axis-aligned bound box (AABB), object bounding box (OBB), or circle/sphere. Circle is chosen in this project because it is consistent with the circle shape of agent. Figure 3.6 illustrates a simple instance of BVH.

Range query could be applied recursively on a BVH. According to the definition of BVH, if a specified range D does not intersect the bounding volume of a parent node, it certainly does not intersect the bounding volume of any of its children, and the query is safe to be pruned. On the contrary, if D intersects the bounding volume

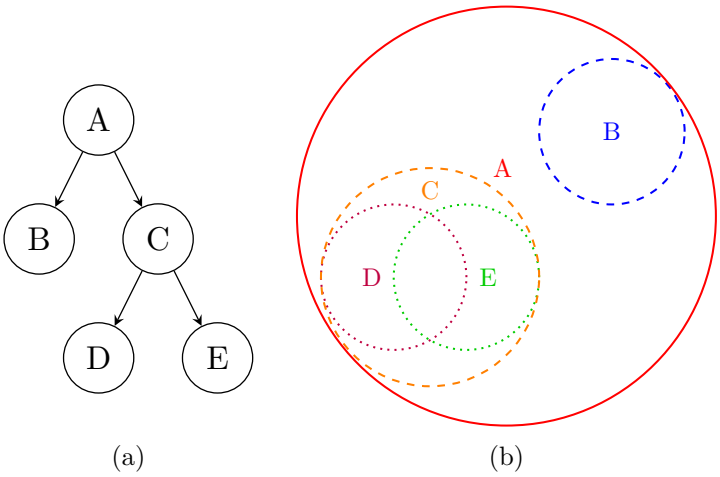


Figure 3.6: A simple instance of BVH.

of a parent node, than it is necessary to check its children. The procedure continues until a leaf node, which contains the information of an agent, is reached. Starting a range query from the root node will return a set of agents within the range, which are considered as neighbors and sent to local collision avoidance module. The following pseudocode demonstrates range query on a BVH.

Algorithm 5 RangeQuery(N,D)

Input: A BVH tree node N, and a circle area D centered at $\mathbf{p}$ with radius r
Output: A list of neighboring agents L

```
if N is leaf node then
    if D intersects the bounding volume of N then
        Add the corresponding agent of N to L
        return
    end if
else
    if D intersects the left child LC_N of N then
        RangeQuery(LC_N,D)
    end if
    if D intersects the right child RC_N of N then
        RangeQuery(RC_N,D)
    end if
end if
```

For simplicity, the BVH used in this project is constructed as a binary tree. There are three general approaches to construct a BVH: the top-down approach, the bottom-up approach, and the incremental approach, while only the incremental one is suitable for frequently updates because it can adjust the hierarchy without

complete reconstruction. For an existing tree (which is empty at the beginning) and a new object, a new node is inserted by traversing through the whole tree from the root. At each internal node, a heuristic is used to determine which child node should be descended to. Although different heuristics yield various tree structures which greatly affect searching performance, and the selection of the optimal heuristic turns out to be difficult, there exist a simple heuristic that provide reasonably good outcome. At each node in the tree the procedure chooses whichever child node that causes the least expansion of the bounding volume. The iteration ends when any leaf node is reached. The leaf node is then split into two new nodes, one for holding the old leaf node and the other for the new object. Note that according to the definition of BVH, it is necessary to recompute the bounding volumes of the ancestors of the new node. The following pseudocode shows the insert operation.

Algorithm 6 INSERT(R,V)

Input: The root node of a BVH R, and a new object with bounding volume V

```
N ← R
while N is not leaf node do
  V_N ← the bounding volume of N
  V_{LC_N} ← the bounding volume of LC_N (the left child N)
  V_{RC_N} ← the bounding volume of RC_N (the right child of N)
  // heuristic for deciding the location of new object
  if GetGrowth(V_N,V_{LC_N}) < GetGrowth(V_N,V_{RC_N}) then
    N ← LC_N
  else
    N ← RC_N
  end if
  Spawn a new leaf node N_new for the object
  Detach N from the BVH
  Spawn a new parent node P for both N and N_new, and connect P to the
  remaining part of the BVH
  Backtrace to recompute the bounding volume of ancestors of N_new
end while
```

At each step of simulation, the BVH should be updated with agents' new positions by deleting old nodes and insert new ones. Here, a self-balancing mechanism is necessary to avoid degenerated cases, and maintain searching performance. Thus, a rotation strategy similar to AVL-tree is applied to ensure balanced tree structure. In addition, a loose factor is also introduced to slightly scale the bounding volume of an agent, which allows the agent to move by small amounts without exceeding the old bound. This saves lots of update operations under conditions such as

congestions where many agents move in low speed.

So far, the methods for combining the Least-effort and the local avoidance approaches have been presented. These methods compute agent trajectories in an environment with obstacles, enabling agents to avoid collisions. In next chapter, the simulation results will be presented.

Chapter 4

Results

4.1 Implementation Setup

The project is implemented in `C++` which is expected to achieve higher performance than dynamic programming languages such as *Python*. An OpenGL-based application is also made for real-time demonstration. With pre-defined scenarios set up, agents are rendered with simple sphere models, and they update their trajectories and velocities based on the algorithm. Figure 4.1 is a screenshot of the application. The colored spheres and arrows represent crowd agents and their trajectories. The box represent an obstacle in the scenario.

All the tests and the evaluation are made on a MAC OS X 10.10 operation system, with an Intel i7 2.3GHz quad core CPU, 16GB of memory, and an NVIDIA GeForce GT 750M GPU. Source code is compiled by `g++ 4.9.2`, with `-O2` flag enabled. The optimization procedure of the effort function makes use of GNU GSL library[1]. OpenMP[2] is used to parallelize the update procedure of agent velocity (see algorithm 3).

Table 4.1 and figure 4.2 measure the performance of PLE and RVO, both with and without OpenMP parallelization, in terms of average step time. The average step time is the time the methods take to simulate a single step for all agents in the scenario. Agents all have the same radius of $0.25m$, and are randomly positioned in a $100m \times 100m$ area. To eliminate possible interference, when sampling step time, agents' positions are fixed, with their velocities computed but not updated. It can be seen that both methods are able to simulate several thousand agents in real time, and gain performance boost from multiprocessor. However, the computation time becomes unacceptable when agent number reaches 10 thousand. As expected, PLE takes longer computation time than RVO due to the optimization procedure of effort function.

The remaining of this chapter carries out the evaluation of the crowd behavior simulated by the algorithm. The algorithm is evaluated by a set of metrics on a

[1] `http://www.gnu.org/software/gsl/`
[2] `http://www.openmp.org`

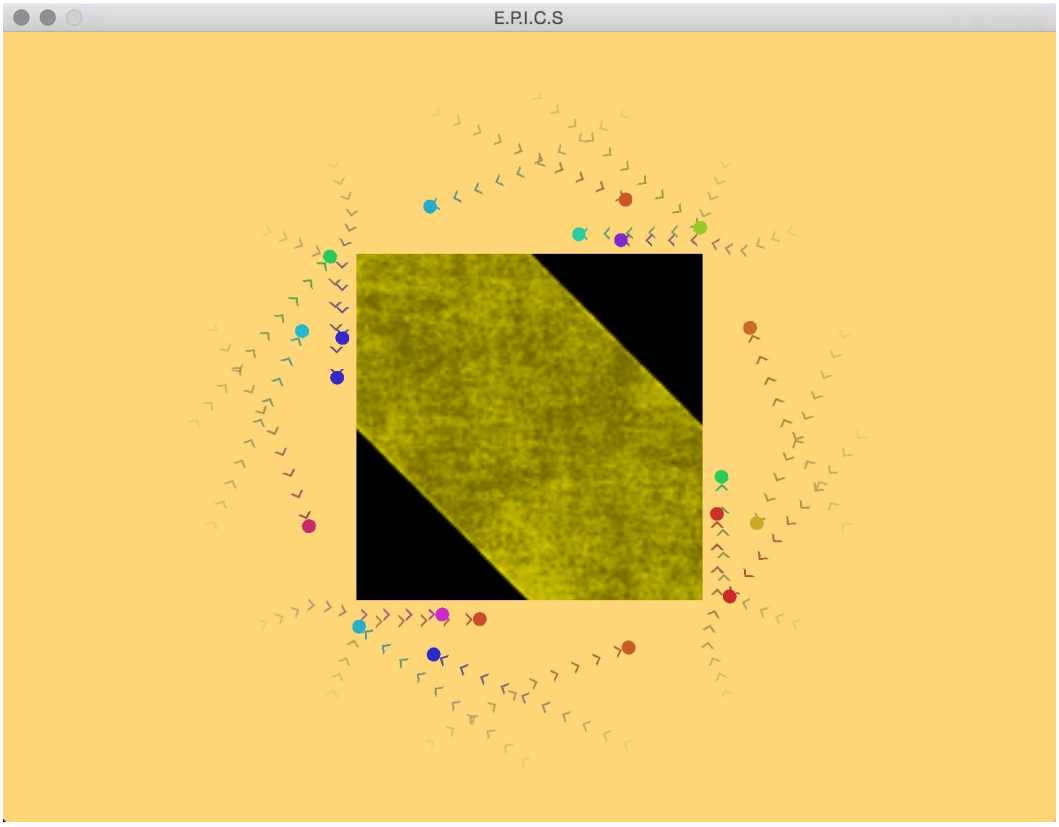


Figure 4.1: Screenshot of simulation application. Colored dots represent agents and arrows represent their trajectories. An obstacle is placed at the center.

Table 4.1: Average step time (s)

Agent Num	PLE	RVO	PLE - OpenMP	RVO - OpenMP
1k	0.0109	0.0082	0.0041	0.0031
2k	0.0156	0.0115	0.0052	0.0038
3k	0.0608	0.0480	0.0151	0.0119
4k	0.1131	0.0941	0.0270	0.0225
5k	0.1719	0.1562	0.0372	0.0338
6k	0.2218	0.1900	0.0475	0.0407
7k	0.3959	0.3413	0.0842	0.0726
8k	0.4803	0.4267	0.1011	0.0898
9k	0.6127	0.5561	0.1269	0.1151
10k	0.7560	0.6289	0.1575	0.1310

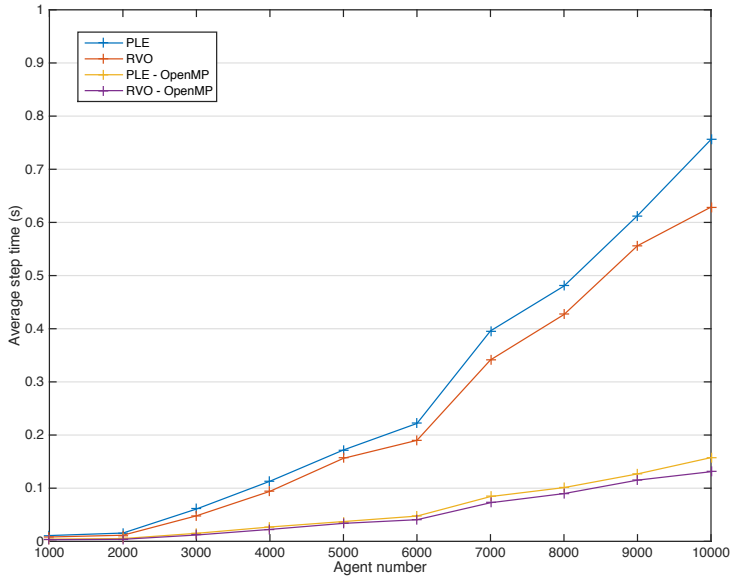


Figure 4.2: Average step time.

selected set of test cases, which are introduced separately in section 4.2 and 4.3. Quantitative scores based on those metrics and detailed case studies are presented in section 4.4.

4.2 Metrics

Four metrics are used in the evaluation, namely *avg. time*, *avg. length*, *avg. effort*, and *avg. accumulative penetration*.

- *Avg. time* measures the average time each agent spends from its initial position to its goal. Let n be the agent number in a crowd, and t_i be the time spent by the i-th agent, the metric is formalized as

$$\frac{1}{n}\sum_{i=1}^{n} t_i \tag{4.1}$$

- *Avg. length* measures the average length of each agent's trajectory from its initial position to its goal. Let l_i be the trajectory length of the i-th agent, the metric is formalized as

$$\frac{1}{n}\sum_{i=1}^{n} l_i \tag{4.2}$$

- *Avg. effort* measures the average effort each agent spends from its initial position to its goal based on Eq 3.1. This metric is used to test the acceptability of the optimization strategy of the effort function. Let τ be time step interval,

T be the total time steps of simulation, v_i^t be the velocity of the i-th agent at the t-th step, the metric is formalized as

$$\frac{1}{n}\sum_{i=1}^{n}\sum_{t=1}^{T}(e_s + e_w|v_i^t|^2)\tau \tag{4.3}$$

- *Avg. accumulative penetration* measures agents' ability to avoid collisions by summing all penetration depth between all agent pairs during the whole simulation and then divide the value by agent number. Recall that an agent is modeled as a circle. If two agents does not collide, the circles representing them will not intersect and thus the penetration depth will be zero. Let r_i be the radius of the i-th agent and $\mathbf{p}_i^t$ be the position of the i-th agent at the t-th time step, the metric is formalized as

$$\begin{aligned} &\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n}\sum_{t=1}^{T} step((r_i + r_j) - |\mathbf{p}_i^t - \mathbf{p}_j^t|) \\ &step(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases} \end{aligned} \tag{4.4}$$

4.3 Test Cases

There are generally three classes of test cases in evaluation, each with different emphasis. *Steerbench* [2] provides numerous typical test cases for steering and crowd simulation algorithm, thus some of its test cases are adopted to perform more comprehensive evaluation.

- *Class A*: Test cases of this class include a small number of agents and no obstacles. They are used to evaluate basic agent behavior and interaction generated by the algorithm.
  - **Oncoming** [3]: Two agents traveling in opposite directions towards a head-on collision.
  - **Frogger** [3]: One agent encounters many perpendicular crossing agents.
  - **4-way-confusion** [3]: Four agents traveling in four opposing directions, meeting at nearly the same time.
  - **Oncoming-groups** [3]: A small group of agents encounters another small group of agents traveling in opposite direction.
  - **20-agent-circle**: 20 agents are spawned on a circle and try to reach the diametrical opposite position on that circle.

[3]These test cases are from *SteerBench*[2].

- *Class B*: Test cases of this class include both a small number of agents and obstacles. They are used to evaluate agent behavior and interaction when interfered by environment.
  - **Curve** [3]: One agent steering through an S-curve to reach the target.
  - **Surprise** [3]: Two agents that do not see each other until the last second because of large obstacles.
  - **3-squeeze** [3]: Two agents facing the same direction encounter an oncoming agent in a narrow hallway.
  - **4-way-confusion-obstacle** [3]: Similar to **4-way-confusion**, yet an obstacle is placed at the center to interfere agents' movement.
  - **20-agent-circle-obstacle**: Similar to **20-agent-circle**, yet an obstacle is placed at the center to interfere agents' movement.
- *Class C*: Test cases of this class include a large number of agents, and may include a complicated scene. They are used to stress test the algorithm, and evaluate aggregated behavior in relatively high density.
  - **Concentric-circle**: $100 + 80 + 60$ Agents are spawned on three concentric circles with slightly different radii and try to reach the diametrical opposite position on that circle.
  - **Bottleneck-corridor**: A long corridor with a bottleneck at the middle. Agents are spawned at the left side of the corridor and try to reach the right end of the corridor through the bottleneck.
  - **Narrow-exit**: All agents must exit a crowded room through a narrow door, and fan-out from the exit.
  - **Downtown**: 1000 agents are randomly placed in a urban-like environment and head to a random exit on the boundary.

4.4 Case Studies

4.4.1 Metric Scores

Table 4.2 shows the resulting scores of the four metrics in all test cases. For comparison, the RVO method, which only considers local avoidance (see section 3.2) and chooses the nearest feasible velocity to desired velocity rather than optimize the effort function,is also evaluated. Therefore, RVO can be seen as a control group to verify the effectiveness of the optimization strategy for the effort function, to see whether the strategy produces better behavior. Parameters used in simulation are illustrated in table 4.3.

[4]This value is only used in the *Concentric-circle* test case.
[5]This value is only used in the *Downtown* test case.

Table 4.2: Metric scores achieved by PLE and RVO in all test cases.

Test Cases	**Metrics**							
Method	**PLE**				**RVO**			
	Avg. Time (step)	*Avg. Length* (m)	*Avg. Effort* (J/kg)	*Avg. Acc. Penet.* (m)	*Avg. Time* (step)	*Avg. Length* (m)	*Avg. Effort* (J/kg)	*Avg. Acc. Penet.* (m)
Class A								
Oncoming	59	19.47	65.31	0	59	19.47	65.31	0
Frogger	54.25	17.63	59.19	0	54.25	17.67	59.31	0
4-way-confusion	80	19.67	71.71	0	79.5	19.53	71.22	0
Oncoming-groups	148.3	46.99	160.0	0	148.6	47.07	160.2	0
20-agent-circle	132.1	39.63	137.0	0	133.4	39.77	137.9	0
Class B								
Curve	118	29.38	106.4	0	118	28.99	105.8	0
Surprise	97	29.02	99.81	0	164	31.35	135.5	0
3-squeeze	145.7	24.76	117.3	0	147.7	24.77	118.4	0
4-way-confusion-obstacle	68.25	20.03	69.21	2.139×10^{-3}	76.25	20.04	72.27	8.754×10^{-3}
20-agent-circle-obstacle	155.7	50.41	169.7	0	160.7	50.29	171.5	0
Class C								
Concentric-circle	635.4	112.9	500.9	8.596	689	107.3	531.9	10.64
Narrow-exit	1814	213.5	1295	4.806	2025	223.2	1416	4.985
Bottleneck-corridor	399.9	93.76	361.0	8.147	380.0	93.81	352.4	11.58
Downtown	453.4	97.22	389.4	8.829×10^{-1}	565.7	93.48	436.7	2.730

Table 4.3: Parameters for PLE and RVO.

Step Time (s)	Agent Prediction Interval (s)	Obstacle Prediction Interval (s)	Max Speed (m/s)	Radius (m)
0.25	$5.0/20.0^4$	2.0	2.0	$0.5/0.2^5$

It can be observed from table 4.2 that in test cases of *Class A* and *B*, PLE and RVO achieve similar scores, while PLE has slightly better performance in most cases in term of *avg. effort*. In more complicated test cases of *Class C*, however, PLE generates agent trajectories that costs less effort more significantly. Although in *Bottleneck-corridor*, PLE generates larger average effort than RVO does, one should notice that RVO fails to avoid agent collision properly, resulting in much larger *Avg. Accumulative Penetration* value. The results provide support for the acceptability of the greedy optimization strategy in 3.1.2. As a "by-product", the results also imply that there is a positive relationship between *avg. effort* and *avg. time*, as PLE achieves shorter *avg. time* in all cases where it also achieves smaller *avg. effort*. On the other hand, exceptions exist for the positive relationship between *avg. effort* and *avg. length*.

To analyze the method in different perspectives, four case studies are here carried out. They serve as mixture of qualitative analysis and quantitative analysis. In particular, some emergent phenomena of real-world crowd are identified and analyzed in this section.

4.4.2 *Oncoming*

The basic *oncoming* is the first test case analyzed. The *oncoming* test case consists of two agents trying to swap their positions. Since they start at the same horizontal coordinate, they cannot reach their goals by simply moving straight ahead without any collision. Instead, they have to walk around each other by steering left or right on the way.

As proposed in 3.3, PLE agent is able to predict the upcoming collision and avoid it, meanwhile optimizing the effort function during the simulation. In figure 4.3, three pairs of trajectories of two agents with different prediction intervals were plotted. With the increase of prediction interval, agents behave in a more cautious manner. When the prediction interval is set to 2, the trajectories do not deviate from straight line until agents encounter each other in a very short distance. In contrast, when the prediction interval is set to 10, two agents perceive the existence of each other at the very beginning of simulation, and walk around each other more "gently", resulting in smoother trajectories. Nevertheless, it does not suggest that larger prediction interval is better in all conditions, because it is more likely to cause deadlock in dense and limited scenarios. As a straight-forward application,it is possible to set different prediction intervals for different agents. This may give viewers a sense of variety in agents' "personalities". However the effect is limited in that changing prediction interval only advances or postpones an agent's perception of its neighbors. The handling mechanism remains the same.

4.4.3 *N-agent-circle*

The *N-agent-circle* is a generalization of the *20-agent-circle* test case. It is used to evaluate the performance of PLE in highly dense situations. N agents are spawned on a circle and try to reach the diametrical opposite positions on that circle. Collisions are expected at the center of the circle since all agents' ideal paths intersect there, and needed to be resolved by the algorithm.

Figure 4.4 captures the simulation process of the *N-agent-circle*, and figure 4.5 plots the variation of *average time* and *average accumulative penetration* along with the increase of agent number. PLE proves its ability to resolve serious potential collisions in this test case, yet it also exposes limitation when density becomes very high. As agent number grows, it takes longer time for computation, and agents collide more seriously. The *average accumulative penetration* starts to become larger than zero and grow slowly when agent number reaches approximately 50. Both metrics grow sharply when agent number reaches 150 because agents almost fill up all space between each other initially.

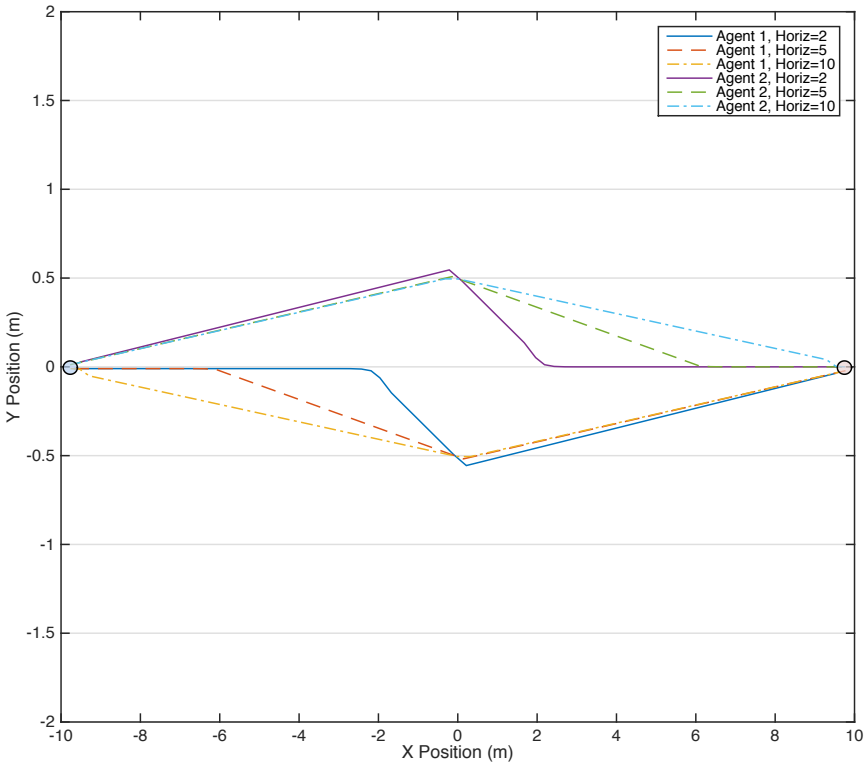


Figure 4.3: Agent trajectory in *Oncoming* test case. The blue circle indicates the initial position of agent 1, and the red circle indicates the initial position of agent 2. The size of circles does not represent the size of agents.

(a) (b) (c)

Figure 4.4: Screenshots of simulating *N-agent-circle* where $N = 100$, circle radius $R = 25m$, and agent radius $r = 0.5m$.

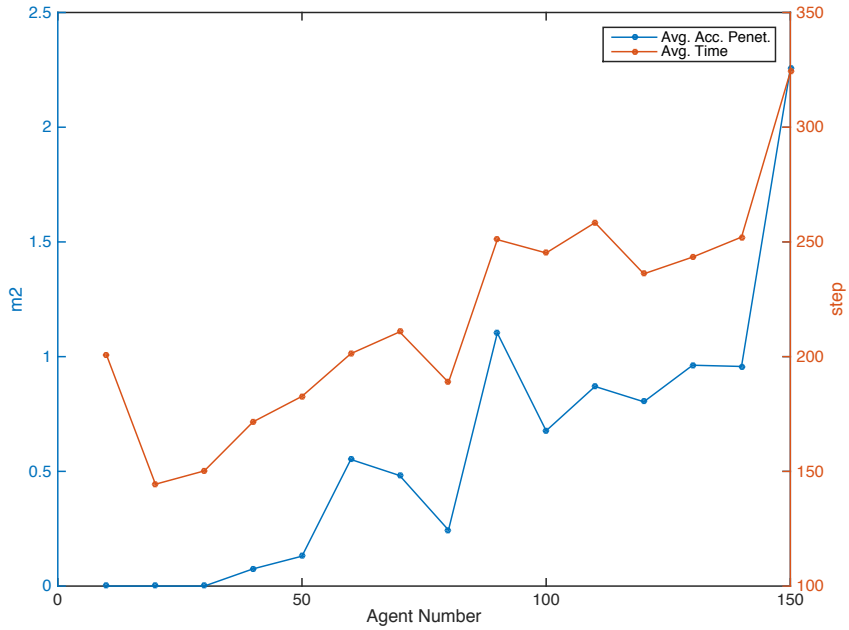


Figure 4.5: Growth of *average time* and *average accumulative penetration* along with the increase of agent number where circle radius $R = 25m$ and agent radius $r = 0.5m$.

An odd result also appears in figure 4.5. While intuitively one may expect simulating more agents is more difficult, the result shows that it costs even more time to simulate 10 agents than to simulate 50 agents. Instead of monotonous increase, obvious oscillation could be observed. It is hard to explain this exception. One may infer that though agent density is an important factor for the difficulty of resolving collision, it is not the only one. Agents' initial positions may be involved as well.

4.4.4 *Bottleneck-corridor*

The *Bottleneck-corridor* refers to G.K.Still's experiment to study emergent phenomena in a crowd flow [15]. The corridor model is $100m$ in length and $20m$ in width, with a bottleneck ($20m$ in length and $10m$ in width) in the middle. Initially, more than 500 agents are populated at the left side of corridor, with radius of $0.5m$ and a simple objective of reaching the right end of the corridor. Figure 4.6 illustrates the simulation of this test case.

Several emergent phenomena can be validated in the simulation. In the early stage of simulation, agents near the boundary of the crowd are able to move faster than those at the center. The phenomenon is known as the *edge effect* [15]. It matches the PLE implicitly because the utilization of extra space outside the bound-

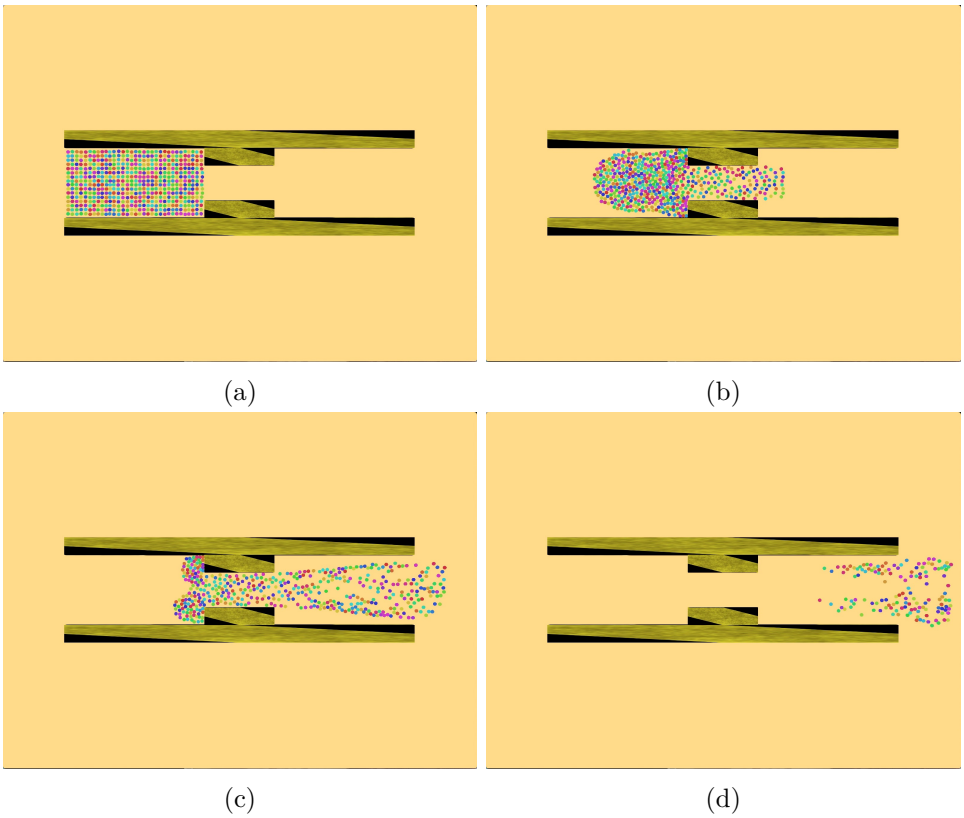


Figure 4.6: Screenshots of simulating *Corridor*

ary of crowd saves effort. Figure 4.7 shows the velocity distribution along a cross section of the crowd. Agents at the left ($-9m$) and the right side ($9m$) are moving more than 50% faster than agents in the middle($-6 \sim 6m$) Subsequently, agents more behind make use of the empty space at the top and the bottom of the corridor. This results in the arc-shaped left contour of the crowd (Figure 4.6b).

One can also observe the formation of jamming at the left of the bottleneck. Although agents at the left of the bottleneck try to avoid protrusion, their action are limited by those at the center. On the other hand, agents behind keep moving up or down due to *edge effect*, which further increases the density of the area at the left of the bottleneck. The congestion will not dissipate until there is enough space in the middle (Figure 4.6c).

After passing the bottleneck, agents only disperse necessarily, leaving the right corners of bottleneck empty (Figure 4.6c and 4.6d, the right part).

4.4.5 *Downtown*

Downtown provides a more complicated scenario to stress test and evaluate the ability of PLE to avoid local congestion. As shown in figure 4.10 , a 10×10 grid

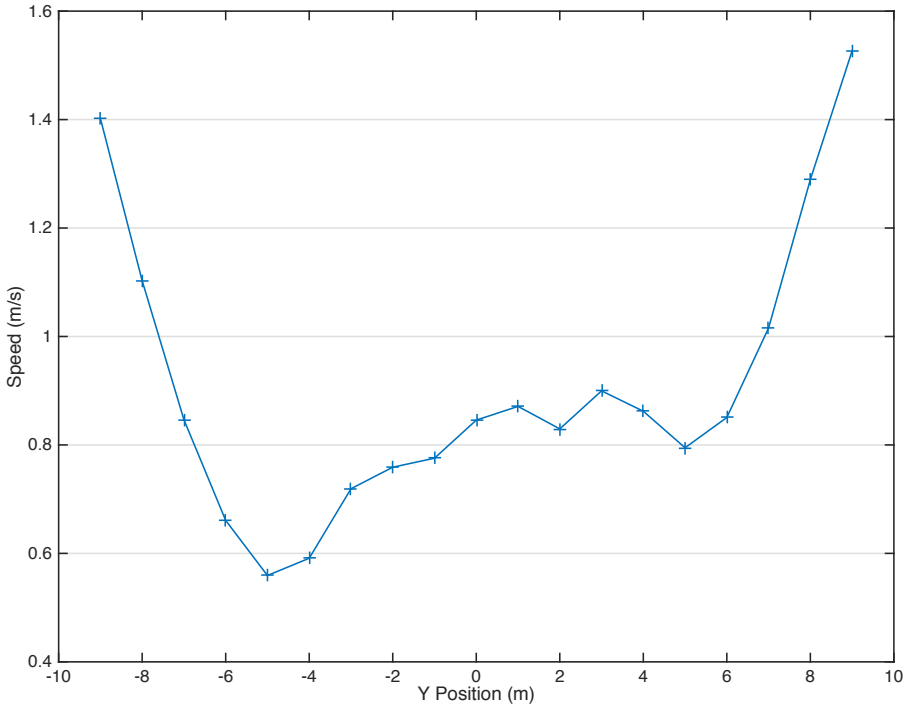


Figure 4.7: *Edge effect*: average speed distribution along Y position from $40th$ to $90th$ step, where X position $\in [-30, -20]m$.

of "streets" is placed horizontally or vertically on the ground, and "buildings" are placed at the middle areas of streets as obstacles. Some large buildings are able to block the streets. The size of each grid cell is $10m$, and the width of each street is $1.6m$. Agents are randomly placed on the streets inside the scenario, and their goals are randomly set on the boundary of the scenario.

As mentioned in 3.4, a dynamic edge weight is assigned to each graph edge in the global planner to estimate the effort needed to traverse the edge at each step of simulation. Results turn out that this strategy outperforms the naive way of using euclidean distance in high density conditions. Estimated effort represented by edge weight help the following agents to walk around existed congestions. Agents tend to choose longer, but less occupied path to save effort and move more efficiently. As a result, more available space in the scenario is utilized. This is supported by the scores in table 4.2 where PLE generates much lower *average effort* and *average time* than RVO does, yet higher *average length*. Figure 4.8 shows average effort for both PLE and RVO with respect to the agent number in the test case. the RVO method suffers severe growth in average effort as agent number increases, while PLE scales well as a result of local congestion avoidance.

To demonstrate the difference more intuitively, figure 4.8 presents a series of screenshots at different stages of simulation. Notice the congestion areas marked by red boxes in both methods. Using the RVO method, every agent keep moving

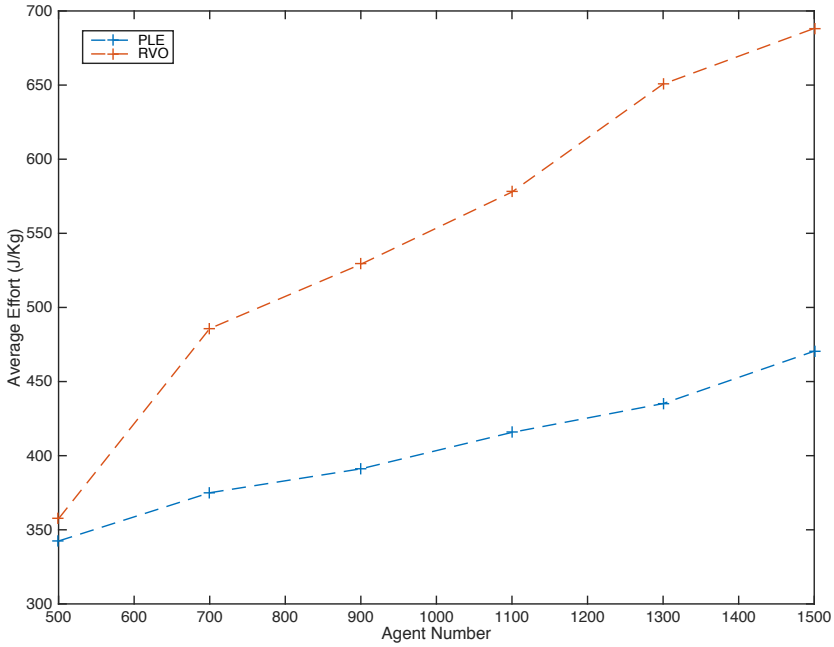


Figure 4.8: Average Effort in *Downtown*.

along the shortest distance path even if it is occupied by other agents, which further worsens the condition. There are still a large number of agents stuck in the middle after 5 minutes of simulation. In contrast, PLE directs agents to longer but unobstructed paths, so that agents reach their goals more efficiently.

In the simulation, the finger effect has also been observed. This is another emergent phenomenon confirmed by G.K.Still from video footage [15]. The*finger effect* describes the phenomenon that when two high density crowds moving in opposite direction encounter each other, people in the same crowd tend to form in line to pass those from another crowd so that the crowd flows look like interlaced fingers. The phenomenon matches the PLE intuitively as following behind others costs less effort than pushing through the crowd oneself. Figure 4.9 illustrates such phenomenon in the simulation. It is easy to see that agents do show propensity to form lines.

4.5 Showcases

Finally, the implementation is used to animate virtual characters in scenarios to better demonstrate its capability in real-time graphical applications. The following showcases make use of the game engine Unity[6], and the implementation is plugged into the engine in the form of dynamic library. The virtual KTH scene, human

[6] `http://unity3d.com`

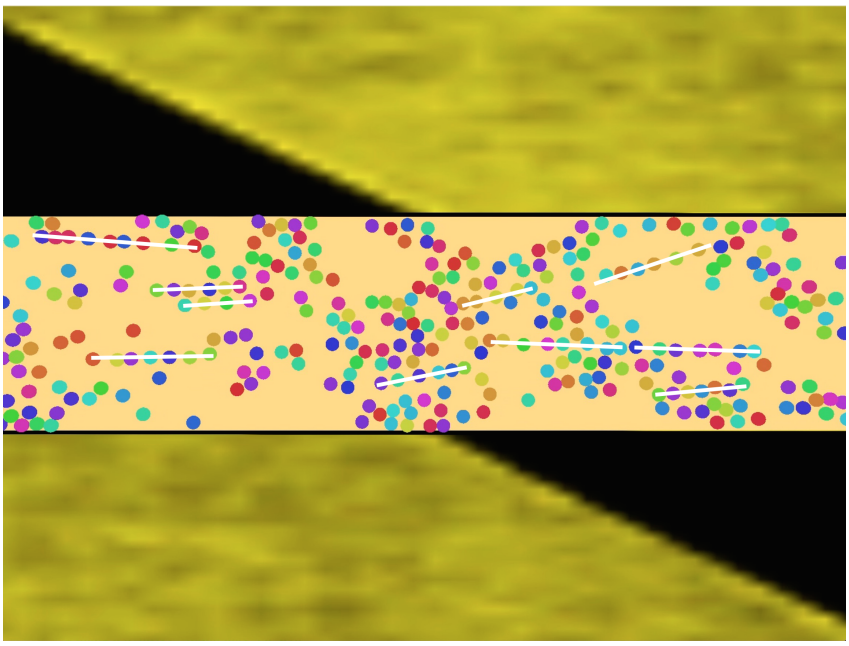
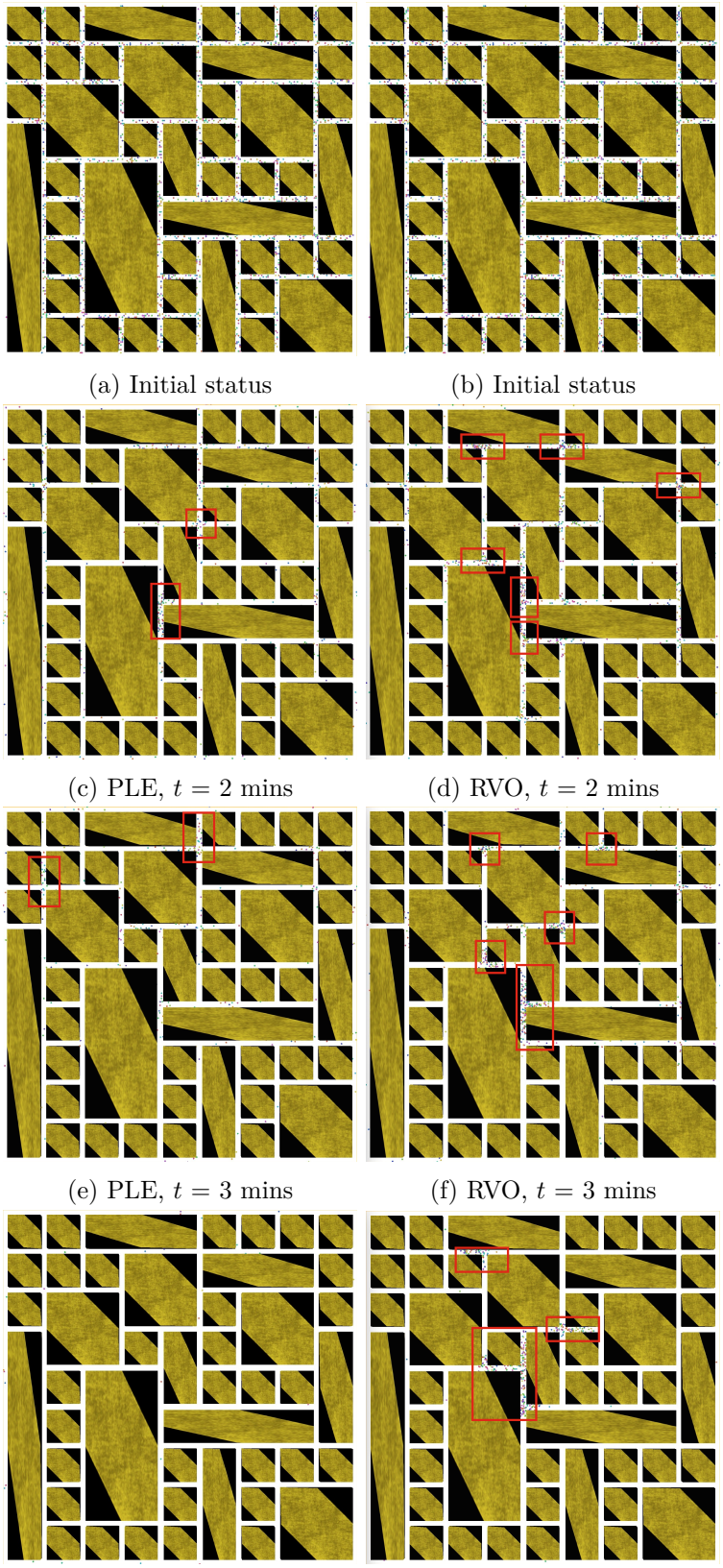


Figure 4.9: *Finger effect*. Agents tend to form into lines (marked by white segments).

characters, and animation are courtesy of students from the DH2320 and DH2323 courses at KTH[7].

[7] http://www.csc.kth.se/~chpeters/projects.html



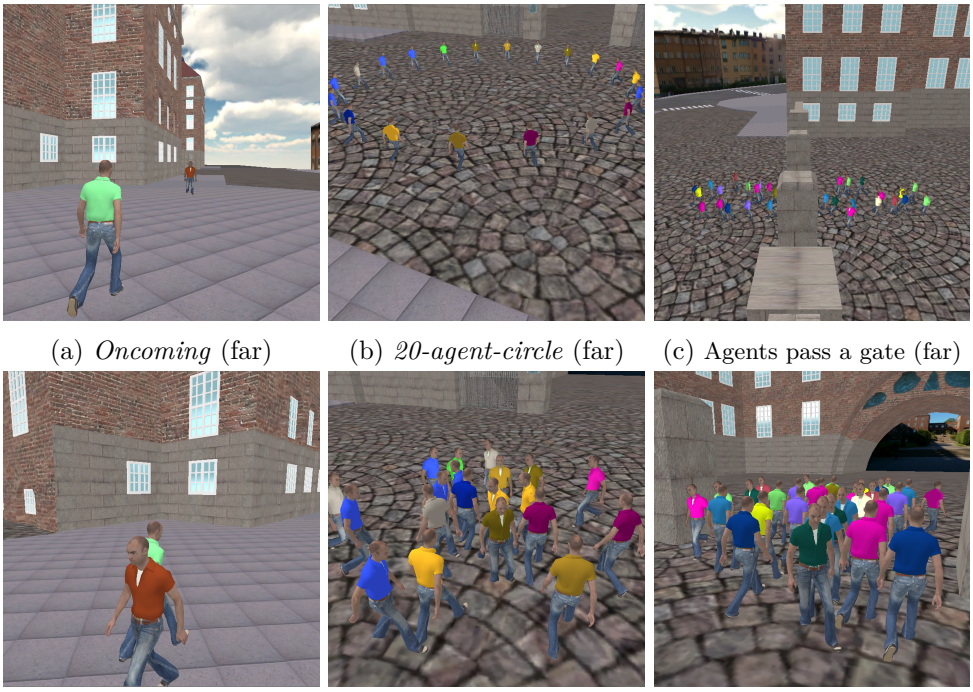
(a) Initial status (b) Initial status

(c) PLE, $t = 2$ mins (d) RVO, $t = 2$ mins

(e) PLE, $t = 3$ mins (f) RVO, $t = 3$ mins

(g) PLE, $t = 5$ mins (h) RVO, $t = 5$ mins

Figure 4.10: Screenshots of simulating *Downtown*. Red boxes mark congestion areas.



(a) *Oncoming* (far) (b) *20-agent-circle* (far) (c) Agents pass a gate (far)

(d) *Oncoming* (near) (e) *20-agent-circle* (near) (f) Agents pass a gate (near)

Figure 4.11: Demonstration of the PLE method implemented in the 3D scenario featuring a virtual model of KTH. Figures (a) - (f) demonstrate a variety of cases from Section 4.5.

Chapter 5

Discussion and Conclusion

In general, the agent behavior generated by the PLE method could be considered as visually satisfactory (figure 4.3, 4.4, 4.6, 4.9). An agent moves in smooth trajectory at most time, except when it reaches one intermediate waypoint and switch its current goal to next waypoint. This could be fixed by path smoothing techniques. In most cases, agents succeed in avoiding collisions with other agents and obstacles gently , though exceptions exist when the density is high and there is not enough space for collision avoidance (table 4.2 and figure 4.5). The implementation is capable of simulating thousands of agents in real-time (table 4.1).

5.1 Relation between PLE and RVO

The results in table 4.2 reflect the similarity of PLE and RVO. Although the two methods share the same local avoidance module, unlike PLE, RVO updates agent velocity by choosing the nearest feasible velocity to the ideal velocity. Nevertheless, even if RVO does not involve with optimization of effort function, it still generates similar behavior to PLE. The only reasonable explanation seems to be that the strategy of RVO is an implicit approximation of the least-effort approach. Therefore, it might be a good idea to combine the usage of two methods, since one produces more accurate results at higher computation cost, while the other produces coarser results but costs less.

On the other hand, due to the update of dynamic edge weight, two methods behave very differently when navigating agents through complex environment with the help of global planner. As shown in section 4.4.5, in high density cases, PLE agents have much stronger ability to avoid congestions by choosing longer but unblocked paths. However, it should be noticed that the dynamic weight assignment strategy is not reliable enough in sparse cases. When agent density is low, it is hardly likely to form local congestions, and most agents are able to move quickly. For an agent navigating in the environment, the estimated effort of an edge only indicates current situation, which will largely likely not remain when the agent reaches that edge. The small number of samples may also cause agent jittering between two similar

paths, resulting in unnecessary effort. In addition, the dynamic weight assignment strategy increases the computation amount as it requires frequent path re-planning. Better results may be achieved by switching between the two weight assignment strategy based on environment.

5.2 Limitations

There are several limitations of the method used in this project, and they are described here from detailed ones to broad ones. The first limitation comes from the linear simplification of constraints from static obstacles on an agent's velocity space (3.2). For an agent A and an obstacle O, $VO^{\tau}_{A|O}$ is calculated as Eq. 3.8, and any velocity outside $VO^{\tau}_{A|O}$ is safe to select. However, rather than selecting velocity from the complement, a sufficient but not necessary linear constraint is applied because it is necessary to keep the constrained velocity space convex in order to utilize the geometric method in section 3.3. This results in the artifact that agents behave too reservedly when encountering obstacles.

Second, it is hard to determine the key parameter of the method. The prediction interval τ is both essential for PLE and RVO, since it greatly affects the "cautiousness" of agent behavior and computational efficiency. However, it is hard to tell which specific τ value produces the best performance in different scenarios. Besides trial-and-error, there is no general rule of thumb or automatic decision process.

The third limitation lies in agent model. Although the circle model significantly simplifies the computation, it remains unchanged in all time, which is far from realism. In real world, it is common for people to adjust posture to move more efficiently. For example, people walk sideways to pass narrow lanes, or squeeze through crowds to reach the other side. The volume and surface area of human body can both change for efficient movement, yet this is not considered in the method of this project.

Fourth, the PLE method does not provide support for heterogeneous modeling of agent behavior. All virtual agents follow the same principle, while in real world different personalities lead to various behaviors. For example, aggressive people tend to overtake slower people, ignore others and focus on achieving their goals, while conservative people would rather wait for others to pass first. Even the same person may have multiple behavior patterns that can be switched based on external situation. For example, a person is expected to flee away in a panic when there is danger nearby, and calm down after getting back to safe area. In addition, interpersonal relationship also influences people's behavior. it is common for people with close relationship to form smaller groups inside crowd flows. Unfortunately, the PLE method does not consider any of these aspects. Although different prediction intervals affects agents' behavior (section 4.4.2), this is an unintended by-product, and the effect is much limited. On the contrary, the social force model could be used to simulate panic crowd [4]. Other methods exist to model diverse personalities, such as the HiDAC [6] which extends the social force model by introducing per-

sonality variables as well as psychological and geometrical rules. Guy *et al.* model the personalities of different agents in a simulated crowd by a perceptually driven formulation [13].

5.3 Future Work

Due to the amount of time, there remain lots of further work that could be done in future. So far only one effort function for normal walking has been developed. It is natural to introduce more effort functions suitable for different moving patterns, such as turning, and different external conditions. The extension of the PLE method is expected to yield more diverse behaviors.

Regarding to the evaluation and analysis of the method, although the basic characteristics such as collision avoidance, smooth trajectory, as well as several emergent phenomena have been observed in simulation, it is still hard to tell precisely whether the simulated behavior is plausible compared to real-world behavior. It is desirable to compare the simulated crowd with real crowd based on statistical methods. The *Entropy Metric* [19] may serve as an appropriate candidate. In addition, it would be interesting to compare the PLE method with more other techniques and study their differences.

5.4 Conclusion

PLE is able to simulates a large number of smooth agent trajectories in real time. Compared with the simulation approach that does not take into account the optimization of effort function (RVO), PLE agent spends less effort and time to reach goal. By choosing different prediction interval value, agent behavior varies from more aggressive one to more reserved one. The method is able to help agents to avoid future collision. Moreover, several emergent phenomena such as *edge effect*, *finger effect* and jamming can be observed in the simulation. With dynamic weight assignment strategy in the global planner, PLE agents are better at avoiding congestions in high density scenarios.

Regarding back to the problem statement and hypothesis of this thesis, it is now possible to conclude that the simulation approach based on the PLE produces good results for modeling individual behaviors in crowds. Although strict physiological and sociological validation is out of the scope of this thesis, inferring from the result(section 4.4), it is convincing to confirm that the PLE at least serves as a key guideline of human behavior.

Bibliography

[1] Stephen J Guy, Jatin Chhugani, Sean Curtis, Pradeep Dubey, Ming Lin, and Dinesh Manocha. Pledestrians: a least-effort approach to crowd simulation. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 119–128. Eurographics Association, 2010.

[2] Shawn Singh, Mubbasir Kapadia, Petros Faloutsos, and Glenn Reinman. Steerbench: a benchmark suite for evaluating steering behaviors. *Computer Animation and Virtual Worlds*, 20(5-6):533–548, 2009.

[3] Dirk Helbing and Peter Molnar. Social force model for pedestrian dynamics. *Physical review E*, 51(5):4282, 1995.

[4] Dirk Helbing, Illés Farkas, and Tamas Vicsek. Simulating dynamical features of escape panic. *Nature*, 407(6803):487–490, 2000.

[5] Adriana Braun, Soraia Raupp Musse, Luiz Paulo Luna de Oliveira, and Bardo EJ Bodmann. Modeling individual behaviors in crowd simulation. In *Computer Animation and Social Agents, 2003. 16th International Conference on*, pages 143–148. IEEE, 2003.

[6] Nuria Pelechano, Jan M Allbeck, and Norman I Badler. Controlling individual agents in high-density crowd simulation. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 99–108. Eurographics Association, 2007.

[7] Craig W Reynolds. Flocks, herds and schools: A distributed behavioral model. *ACM Siggraph Computer Graphics*, 21(4):25–34, 1987.

[8] Adrien Treuille, Seth Cooper, and Zoran Popović. Continuum crowds. *ACM Transactions on Graphics (TOG)*, 25(3):1160–1168, 2006.

[9] Rahul Narain, Abhinav Golas, Sean Curtis, and Ming C Lin. Aggregate dynamics for dense crowd simulation. In *ACM Transactions on Graphics (TOG)*, volume 28, page 122. ACM, 2009.

[10] Carsten Burstedde, Kai Klauck, Andreas Schadschneider, and Johannes Zittartz. Simulation of pedestrian dynamics using a two-dimensional cellular automaton. *Physica A: Statistical Mechanics and its Applications*, 295(3):507–525, 2001.

[11] Kazuhiro Yamamoto, Satoshi Kokubo, and Katsuhiro Nishinari. Simulation for pedestrian dynamics by real-coded cellular automata (rca). *Physica A: Statistical Mechanics and its Applications*, 379(2):654–660, 2007.

[12] Funda Durupinar, Jan Allbeck, Nuria Pelechano, and Norman Badler. Creating crowd variation with the ocean personality model. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 3*, pages 1217–1220. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[13] Stephen J Guy, Sujeong Kim, Ming C Lin, and Dinesh Manocha. Simulating heterogeneous crowd behaviors using personality trait theory. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 43–52. ACM, 2011.

[14] LF Henderson. The statistics of crowd fluids. *Nature*, 229:381–383, 1971.

[15] G Keith Still. *Crowd dynamics*. PhD thesis, University of Warwick, 2000.

[16] George Kingsley Zipf. *Human behavior and the principle of least effort*. addison-wesley press, 1949.

[17] V.T. Inman, H.J. Ralston, F. Todd, and J.C. Lieberman. *Human walking*. Williams & Wilkins, 1981.

[18] Nuria Pelechano, Catherine Stocker, Jan Allbeck, and Norman Badler. Being a part of the crowd: towards validating vr crowds using presence. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 1*, pages 136–142. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[19] Stephen J Guy, Jur van den Berg, Wenxi Liu, Rynson Lau, Ming C Lin, and Dinesh Manocha. A statistical similarity measure for aggregate crowd dynamics. *ACM Transactions on Graphics (TOG)*, 31(6):190, 2012.

[20] Rachel McDonnell, Michéal Larkin, Simon Dobbyn, Steven Collins, and Carol O'Sullivan. Clone attack! perception of crowd variety. *ACM Transactions on Graphics (TOG)*, 27(3):26, 2008.

[21] Rachel McDonnell, Michéal Larkin, Benjamín Hernández, Isaac Rudomin, and Carol O'Sullivan. Eye-catching crowds: saliency based selective variation. In *ACM Transactions on Graphics (TOG)*, volume 28, page 55. ACM, 2009.

[22] Jur Van Den Berg, Stephen J Guy, Ming Lin, and Dinesh Manocha. Reciprocal n-body collision avoidance. In *Robotics research*, pages 3–19. Springer, 2011.

[23] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer Science & Business Media, 2008.

[24] Joseph O'Rourke. *Computational geometry in C*. Cambridge university press, 1998.