An Investigation of an Example-Based Method for Crowd Simulations

STEFAN KAALEN

DAVID MALMSTRÖM
Figure 1: Real-life crowd movements is recorded (left), these movements are used to create a database of examples (middle). This database is then used to create completely new crowd simulations (right).

Abstract

The problem of simulating a crowd to see how it would behave in certain situations or just to create a realistic-looking scene to be used in a movie or video game is important and complex, and there are many different methods to solve it. This project is primarily an investigation of the example-based crowd simulation method described in the article "Crowds by Example" by Lerner et al. [11]. In the article, traced video footage of crowds are used to create a data set. The simulation program continuously finds situations in the data set that resembles the current situations in the simulation and updates the simulation thereby. We implemented this for around 10 agents using Unity 3D. Example-based crowd simulations does not only, like some other types of crowd simulation methods (for example ORCA [1]), take collision avoidance into account but also the more complex ways the human mind thinks and therefore does not always behave as one would predict. The main conclusion is that this method of simulating crowds has the potential to create more realistic simulations than other forms of crowd simulations. The downsides are that the time the program spends creating simulations can quickly get very high and to make realistic simulations a lot of video footage must be filmed and then traced.
Sammanfattning

Problemet att simulera en folksamling för att se hur den skulle bete sig i olika situationer eller bara för att skapa scen som ser realistisk ut för att använda i en film eller ett datorspel är viktigt och komplext, och det finns många sätt att lösa det på. Det här projektet är primärt en utredning av den form av exempelbaserad simulering av folksamlingar som beskrivs i artikeln "Crowds by Example" av Lerner et al. [11]. I artikeln används filmade och sedan spårade videoklipp av folksamlingar för att skapa en databas. Simulationsprogrammet hittar kontinuerligt situationer i databasen som liknar de nuvarande situationerna i simuleringen och uppdaterar utefter dessa simuleringen. Vi implementerade detta för ungefär 10 agenter i Unity 3D. Exempelbaserade simuleringar av folksamlingar undviker inte bara, som andra typer av folksamlingssimuleringar (till exempel ORCA [1]), kollisioner, utan tar även hänsyn till de mer komplexa sätten som människor och den mänskliga hjärnan beter sig på, beteenden som inte i vanliga fall är så lätt att förutsäga. Den huvudsakliga slutsatsen av det här projektet är att den här simuleringsmetoden har potential att skapa mer realistiska simuleringar än andra metoder för simulering av folksamlingar. Nackdelarna är dock att tiden som programmet spenderar på att skapa simuleringar snabbt kan bli väldigt lång och att mycket video måste filmas och sparas för att realistiska simuleringar ska kunna skapas.
# Contents

## 1 Introduction

1.1 Background ................................................................. 1

1.2 Research Purpose .......................................................... 2

1.3 Methodology ................................................................. 2

1.4 Scope and Limitations ...................................................... 3

1.5 Outline ................................................................. 3

## 2 Previous Work

2.1 Methods for Multi-Agent Navigation .................................... 4

2.1.1 Steering Methods ......................................................... 4

2.1.2 RVO ................................................................. 4

2.1.3 ORCA ................................................................. 5

2.1.4 Social Forces ........................................................... 5

2.1.5 Example-Based Crowd Simulation ................................... 6

2.2 Evaluation Methods ......................................................... 6

2.2.1 Perceptual Evaluation ................................................. 6

2.2.2 Steerbench .............................................................. 6

2.2.3 Entropy-Based Evaluation ............................................ 7

2.2.4 Evaluation Based on Path Patterns .................................. 7

2.2.5 Data-Driven Evaluation .............................................. 8

## 3 Implementation

3.1 Pseudo Code of Simulations ............................................... 9

3.2 Setting up Database of Examples ....................................... 11

3.2.1 Saving Data ............................................................ 11

3.2.2 Influence Functions .................................................... 12

3.3 New Simulations Using Examples ...................................... 14

3.3.1 Update Trajectory ....................................................... 14

3.3.2 Matching Function ..................................................... 15

3.3.3 Collision Handling ..................................................... 16

3.3.4 Approximate nearest neighbour search ............................. 16

## 4 Evaluation

4.1 Database Setup Scenario ............................................... 18

4.2 Test Cases ................................................................. 18

4.3 Results ................................................................. 19

4.3.1 Observations .......................................................... 19

4.3.2 Performance .......................................................... 20

4.4 Evaluation of Results .................................................... 21

4.5 Analysis ................................................................. 22
5 Conclusion

5.1 Summary ................................................................. 25

5.2 Future Work ............................................................ 25

6 Acknowledgements ...................................................... 27
1 Introduction

Crowd simulation is the process of simulating a crowd, or more specifically, simulating the behaviour of an amount of entities, often called agents, moving in the proximate environment of each other. Often the agents in the simulations are to resemble humans, but nothing is preventing the crowd from being composed of something else, for example animals or robots.

1.1 Background

Crowd simulation can be, and are being, applied to a variety of places in today’s society. Perhaps the most prominent is the movie and video game industries, where using simulations of a crowd can save a lot of money and time compared to hiring people to make up the crowd. There are many companies specialised in precisely this. Two examples are Golaem\(^1\) which has worked for major TV productions such as *Game of Thrones* and *Breaking Bad* and Massive software\(^2\) which provides a software package specialised in simulating crowds for films and TV and has been used in (among other) the films *Lord of the Rings* and *Avatar*. Figure 2 shows two examples where simulated crowds is used.

![Figure 2: Examples where Golaem (a) and Massive Software (b) is used. The pictures are taken from the two software’s respective web pages.](http://www.legion.com/news/legion-launches-3d-crowd-animation-service)

But there are of course many other areas in which crowd simulations can be useful. Building designers can use crowd simulations to be able to present their building concepts where the potential clients can actually see crowds moving in the buildings. One company which provides a service for doing this is Legion\(^5\). An important thing that crowd simulations also could be used for is to prepare for emergency situations, for example one might want to use simulated crowds to find out what is the most efficient way to evacuate a building in case of a fire.

Another application for crowd simulations is in the field of coordinating control systems of robots, or swarm robotics \([14, 13, 19]\). Crowd simulations can for example be used to simulate swarms of unmanned aerial vehicles (UAVs) and then use these simulations to steer actual UAVs. There are like every kind of crowd simulations different ways of doing this, you can choose to simulate each UAV as its own

---

\(^{1}\) http://golaem.com/

\(^{2}\) http://www.massivesoftware.com/


entity but you can also choose to simulate the swarm as one entity. de Vries and Subbarao presents a method for steering a swarm as one entity in [21].

The goals of crowd simulations differ depending on the application. In movies and video games the goal is most likely to simulate a crowd that has the most natural look possible, the creators wants the audiences to perceive the simulations as realistic. Meanwhile, in a simulation of crowds during a catastrophe, the goal is rather to simulate a crowd that acts as close to reality as possible, this to be able to prepare for when a real catastrophe occurs. At a first glance you may think that these two goals are the same thing, but that does not have to be the case, the human mind is complex and diverse and therefore people in real events may not act in a way that other people see as rational and thereby realistic. This means that a crowd is not necessarily realistic just because it looks realistic and vice versa.

There are many different ways of simulating a crowd, for example using steering methods [18], RVO/ORCA [2] [1], social forces [9] or example-based crowd simulations [11]. There are also methods based on the similarities, as described in [10], between moving crowds and the physical models of a flowing continuum. By the use of these similarities physical-based particle models as in [7] can be created. To evaluate whether a simulated crowd is “good” or not, there are several types of evaluation techniques, for example evaluation using the SteerBench [20] benchmarking suite, entropy-based evaluation [6], evaluation based on finding latent path patterns [22], or data-driven evaluation [12]. Different evaluation methods may differ in whether they optimise for, for example more realistic, realistic-looking or effective crowds simulations. Therefore certain evaluation methods may be better than others when it comes to creating crowd simulations for certain purposes.

1.2 Research Purpose

The goal of this project is to re-implement and investigate the example-based crowd simulation method presented in “Crowds by Example” by Lerner et al. [11]. Furthermore this paper will seek to answer the questions “what is the advantages and disadvantages of example-based crowd simulations compared to other crowd simulation methods?” and “what is the possible future usage of example-based crowd simulation?”.

1.3 Methodology

The methodology while re-implementing [11] was to try to follow the article as closely as possible, but there were some important implementation details missing. To handle this, we contacted the author to get some clarifications, but for the most part we came up with solutions on our own, meaning that our version of the method differs a bit from the original author’s in some ways.

Our implementation of the crowd simulation method was made using C# in Unity 3D[6], which is a game engine used to create video games. Unity 3D made it relatively easy for us to implement the crowd simulations to a visual environment without having to focus too much on the graphical details, and without

it the project would probably have taken a lot longer time to complete.

The example data from real-life video footage used in this project was taken from the “Zara Data Set”[7] that was used in [11], which in our case yielded around 4000 example situations to our completed example database, details of this can be seen in section [5.2].

1.4 Scope and Limitations

The example-based method of simulating crowds discussed in this paper works best on sparser crowds, and it is not optimal to use in situations with a large number of agents due to the performance going down rapidly. The methods primary feature is that it can display many naturally occurring and complex behaviours of crowds, for example smooth collision avoidance and group behaviour, without explicitly taking them into account when creating a simulation. Since the method is based on earlier recorded examples of real-world situations it can be argued that one limitation of the method is that it will have trouble simulating crowds in scenarios which are not similar to any other previously observed real-world scenario.

It should be noted that this paper is written with a consumer PC in mind when it comes to performance and all the simulations and results mentioned was performed on a desktop with an Intel Core i5 6600K CPU.

1.5 Outline

In this paper we are going to start off with a previous work section (section [2]) where we present some of the different methods for creating and evaluating crowd simulations. We will then move over to the implementation section (section [3]) in which we will describe in detail what we have done and how we have managed to make example-based crowd simulations. This will be done with a combination of pseudo code and text. After the implementation section there is an evaluation section (section [4]), in this section we will present and evaluate our results, and we will also analyse both our implementation of the example-based simulation and the example-based simulations in general. Finally there is the conclusions section (section [5]), this section consists of a summary of what we have done in this project followed by some future work potential building on our paper.

https://graphics.cs.ucy.ac.cy/research/downloads/crowd-data
2 Previous Work

Here will follow a short introduction to some of the different present methods for creating and evaluating crowd simulations.

2.1 Methods for Multi-Agent Navigation

There are many methods for simulating crowds and we will in this section present a few of them.

2.1.1 Steering Methods

![Figure 3: Illustrations of the three steering behaviours described in [18]: separation (left), cohesion (middle) and alignment (right). [From [18]]](image)

One way of simulating a crowd is to define certain parameters controlling an agent’s steering based on the positions and velocities of other simulated agents. Reynolds described in [18] a method of simulating autonomous multi-agent navigation called "Boids" using three parameters, called "steering behaviours": separation, cohesion and alignment, which are used to control each agent. Separation is used to regulate the distance between agents, cohesion is used to regulate the agents ability to approach and form groups with other agents and alignment is used to regulate the agents tendency to align themselves with the rest of the agents. An illustration of these 3 parameters is shown in Figure 3. Only the nearest vicinity of the agent is influencing the agents steering, and different combinations of separation, alignment and cohesion will result in a different future behaviour of the crowd. This method is often used to simulate coordinated animal motion seen in for example bird flocks or fish schools.

2.1.2 RVO

![Figure 4: Number of steps needed to avoid a head-on collision between two agents using VO (left) compared to RVO (right). [From [2]]](image)
RVO (Reciprocal Velocity Obstacles)[2] is a way of implementing real-time multi-agent navigation in which each agent navigates without explicit communication with other agents. RVO is based on an earlier type of motion planning method called Vertical Obstacles (VO) described in [5] by Fiorini and Shillert where avoiding manoeuvres are selected for agents to avoid obstacles. The difference between RVO and VO is that in RVO it is taken to account that the other agents in the scenario will make a similar collision-avoidance reasoning as the agent in question, resulting in better performance, as shown in figure 4.

2.1.3 ORCA

Figure 5: Snapshots of an ORCA simulation in action. The scenario has as can be seen a complex layout and a large number of agents. [From [1]]

A problem that exists when using RVO is that RVO can not generally ensure collision avoidance. To ensure that a scenario plays out collision free when using RVO some specific conditions must be filled. This is what makes ORCA (Optimal Reciprocal Collision Avoidance), as proposed in the article by van der Berg et al. [1], generally more reliable than RVO when it comes to collision avoidance in crowds. ORCA guarantees that the multi-agent navigation stays collision free, even though the agents in ORCA like agents in RVO have no explicit communication with each other. ORCA is a state of the art of crowd simulation at the moment, and can simulate very large number of agents without trouble, as can be seen in figure 5.

2.1.4 Social Forces

Another approach of constructing a simulated crowd is given in [9] by Helbing and Molnár. Here it is proposed that pedestrian behaviour can be modelled using “social forces”, a form of measure of each individual’s internal motivations to perform certain movements. There are several terms presented that decides a pedestrian’s motion. The first one quantifies the acceleration toward the desired velocity, the second one reflects an individuals distance to other persons and static geometry and the third term models the attractive behaviours of crowds. By combining these a probable motion of individuals in a crowd can be determined and these can be used to simulate the future movement of the individuals in the crowd.
2.1.5 Example-Based Crowd Simulation

The crowd simulating technique that will be investigated and re-implemented in this paper is the example-based crowd simulation described in [11] by Lerner et al. This method uses traced video footage of a real-life crowd to generate a database of “examples”, where an example consists of a specific situation in the crowd logged over about 1.6 seconds, including a “subject” person along with the other people being close enough to probably have influenced this person's actions. This means that an example in the database can describe the crowd’s behaviour in that particular moment, from the perspective of the subject person.

For the simulation of a synthesised crowd, this database is then searched through every time an agent is in need of a new trajectory, and through a similarity function an example which matches the current situation best is found. The trajectory of the subject in this example is given to the agent in the simulation, provided that no collision with other agents will occur, and the next simulated agent in need of a trajectory can be treated. Due to the search through the database this kind of simulation can be quite demanding for even a medium amount of agents, but things can be done to improve on this a bit.

2.2 Evaluation Methods

Following is a selection of methods that can be used to evaluate different aspects of a crowd. They are all appropriate for different kinds purposes and have various types of advantages and disadvantages.

2.2.1 Perceptual Evaluation

This is perhaps the most obvious, simple (in theory) and in some aspects important way of evaluating crowd simulations. It is easy just to gather a group of people to look at a simulation and instruct them to grade its “look-and-feel” on some scale. This is also the most important metric to optimise for many applications where the goal is just to display the visuals to some kind of audience. Aside from overall smoothness and effectiveness of the crowd, which is important in most evaluation methods, other factors plays a large part in perceptual evaluation, for example the orientations of the agents, as mentioned in [17] by Peters et al. While it may give a good looking behaviour perfect for films or video games, perceptual evaluation may not be that practical for analysing what could happen in for instance an emergency evacuation, where crowds can behave in a very different way from other types of situations, as mentioned in [8] by Helbig et al. The big problem though is that this is a very subjective evaluation technique, and the results for the same crowd simulation can vary a lot depending on what the people that watched it believes is realistic.

2.2.2 Steerbench

SteerBench is presented in [20] as a benchmarking suite for evaluating steering behaviour. It scores the performances of algorithms for simulating crowds by simulating crowds in several different test scenarios with the aim of giving an objective assessment of the movements. A disadvantage here is of course that there is a finite set of scenarios that are being investigated, which may not cover the situation that the crowd simulation is planned to simulate at all. It does however allow the user to add new test cases and to choose which ones if any of the existing test cases to use. This evaluation method will present an answer
to which simulation algorithm is the most effective, which could possibly coincide with which simulation algorithm is most realistic looking. It will however not answer which simulation is more realistic.

2.2.3 Entropy-Based Evaluation

Figure 6: Three different kinds of simulations (b-d) compared to a similar real-world data scenario (a). A lower entropy metric indicates a higher similarity. The red rings are there to help point out the similarities and differences between the pictures. [From [6]]

An entropy-based evaluation is proposed by Guy et al. [6]. The method is described as an information-theoretic method to measure the similarities between observed real-world data and the movements of the agents in a simulated crowd using Bayesian inference and a maximum likelihood estimator. More specifically this evaluation method presents a way of quantifying a simulation method’s ability to show the collective behaviour that is shown in the real-world data. This gives a metric for a specific simulation method that can be evaluated with respect to several factors and compared to other simulation methods, as shown in figure 6. Since this evaluation method scores a simulation algorithm by its ability to show the collective behaviour that is shown by real crowds, it mainly seeks to answer the question of how realistic a simulation method is.

2.2.4 Evaluation Based on Path Patterns

Figure 7: The three left pictures show the trajectories resulting from three path planning algorithms. The right three pictures show the resulting top path patterns in each corresponding situation. [From [22]]

Wang et al. introduces in [22] a method for evaluation based on finding latent path patterns in both real and simulated data using non-parametric Bayesian inference. A simulated crowd’s path pattern’s fidelity to a real crowd’s path patterns can then be analysed, allowing for comparison with other simulation techniques. Much like entropy-based evaluation, the path patterns technique seeks to quantify how realistic crowd simulation algorithms are, rather than how realistic looking they are, by comparing the simulations to real data. It does however differ from entropy-based evaluation in the way that entropy-based evaluation
studies a simulation’s collective behaviour while this evaluation method tries to find more latent patterns in real crowds to compare with. An example of the path patterns created from a number of trajectories is shown in figure 7.

2.2.5 Data-Driven Evaluation

[12] by Lerner et al. is a paper that describes a data-driven evaluation of crowd simulations. This method evaluates the grade of realism in a crowd simulation by algorithmically comparing it to real life examples - video footage of a real crowd where the movements of individuals have been traced and recorded. For each decision of every agent in the simulation a state-action pair will be stored which holds information about what decision is made and what influences the agent in this decision making. These state-action pairs are then compared to state-action pairs of real crowds to find a similarity value. This method does like entropy- and path pattern based evaluation, evaluate the realism rather than how realistic looking the simulation is.
3 Implementation

To create and visualise our simulations we have been using Unity 3D, which is a game engine used to create video games. We have chosen to divide the process of creating example-based simulations into two steps:

- Use the sparse crowd footage to build up a database
- Create new simulations using our database

The first step to do is to use our sparse crowd footage to create a database of examples, more about this is explained in section 3.2. This database will then, in the second step, be used to create new example-based simulations. The pseudo code of this is displayed in section 3.1 and the details of how this is done is then further explained in section 3.3.

3.1 Pseudo Code of Simulations

To give an overview of our implementation method we first present a pseudo code of the algorithms used. For more detailed explanation of each step, see the later subsections in this section.

https://unity3d.com/
Algorithm 1 This is the main function of our program. Config. stands for configuration.

1:   def main{
2:       read in database of examples
3:       place out wanted number of agents in Unity, give them start positions and start velocities
4:       while (currentFrame < total numbers of frames wanted){
5:           for (every agent in our simulation){
6:               assignTrajectory(this agent)}
7:       }
8:   }
9:   def assignTrajectory(q) {
10:       if (current trajectory for q is not finished) {
11:           if (passed frames since last trajectory update ≥ 15 frames) {
12:               use matching function to compare q’s config. with its config. at last trajectory update
13:               if (matching value < predefined cutoff matching value) {
14:                   updateTrajectory(q)
15:               }
16:           }else{
17:               updateTrajectory(q)
18:           }
19:       }
Algorithm 1 shows the pseudo code of how our main function works, this function will be run at the start of our simulation. As can be seen in algorithm 1, every agent will at every frame go through the function "assignTrajectory". The function "assignTrajectory" will find out whether an agent needs a new trajectory, if this is the case the agent will go through the function called "updateTrajectory".
Algorithm 2 This will be done if a certain agent \( q \) needs a new trajectory. Config. stands for configuration.

```python
1: def updateTrajectory(q) {
2:     calculate all influence values working on agent q
3:     if (all influence values are below predefined cutoff value){
4:         give agent q a trajectory from random example without influences over cutoff
5:     } else{
6:         get matching value between current config. and the config. of all examples in database
7:         for (all examples sorted after falling matching values) {
8:             if (that example’s trajectory will not send agent q towards a collision) {
9:                 assign the first \((matchingvalue) \cdot 40\) frames of that trajectory as the trajectory
10:                of agent q and break loop
11:             }
12:         }
13:         if (no new trajectory was assigned) {
14:             for (every example in the database) {
15:                 calculate affinity value between q and the example using the affinity function
16:                 for (every example in database sorted by ascending affinity values){
17:                     if (that example’s trajectory will not send agent q towards a collision) {
18:                         assign that trajectory as the trajectory of agent q and break loop
19:                     }
20:                 }
21:             }
22:         }
23:     }
```

Algorithm 2 shows the pseudo code of how the function "updateTrajectory" works. As mentioned above, this function will be used if the agent \( q \) needs an updated trajectory. This function will try to give agent \( q \) the trajectory of the example with the highest matching value. If the trajectories of all the examples with a good matching value would send agent \( q \) to a collision this function gives agent \( q \) the trajectory of an example with a high affinity value which does not lead agent \( q \) into a collision.

### 3.2 Setting up Database of Examples

As mentioned in the short summary of the simulation method in section 2.1.5 to make example-based simulations we first need to set up a database of examples. This database will be based on the text-files\(^9\) describing crowd movement that were used in [11] by Lerner et al. These are text files containing the position over some frames for each person when they are in frame during the recorded videos.

#### 3.2.1 Saving Data

Every example in the example database has a “subject” agent whose trajectory is what can be called the main information of the example, the information which will be passed on to the agent in the simulation in need of a new trajectory. As can be seen in figure 8 each example contains an example number, a list of FrameData for each frame in the example, and a list of InfluenceValues containing information about how the subject agent is influenced by the other agents in the scene during this time window. An example

:\[\text{https://graphics.cs.ucy.ac.cy/research/downloads/crowd-data}\]
is defined to go over 40 frames, which is with 25 frames per second about 1.6 seconds, which means that each example stores 40 different FrameData in the frames list. Each FrameData contains a frame number (from 0 - 39), an AgentData object representing the example subject and a list of AgentData representing any other agents that might have influenced the subject agent during the time of the example. An agent trajectory is saved by saving down the position, direction and speed of the agent for each frame in an AgentData object (also containing an agent number to keep track of what agent it belongs to). The position is saved in a local coordinate system, unique for each example, with the origin at the subject agent’s position at frame 0 and the y-axis along the direction of the subject agent at frame 0. The direction saved in AgentData is given in radians counter-clockwise from the local x-axis.

To save the finished database we chose to serialise (saving down the whole object) the ExampleContainer object using Protocol Buffers (a .proto file) which saves the information much faster and using less memory than other similar file types like Extensible Markup Language files (.xml).

Figure 8: Schematics of how the examples is saved. An ExampleContainer is a list of examples (ExampleDatas) and is what makes up the whole database. An ExampleData contains an integer identifying that example, all influence values working on the subject agent, the trajectory of the subject agent and all influencing agents during for the 40 frames.

### 3.2.2 Influence Functions

There are many factors which could influence a person’s movement, some of these factors are other persons, houses and the topology of the scene. In [11] consideration has only been taken of two factors, other people in the near vicinity and static geometry such as buildings and cars. Since the description of the treatment of the static geometry is very vague in the original paper we used other agents as the only influencing factors in our implementation. The information about how much an agent is influenced by a certain influencing factor (agent) is be quantified by the use of influence functions.
In [11] a form of these influence functions were proposed, which we will use for our implementation of example-based crowd simulations. In this section we will denote the subject agent of an example as agent $i$ and its influence factors will be denoted agents $j$. The velocity of agent $i$ as well as the positions of agent $i$ and an influencing agent $j$ needs to be considered to account for how much agent $j$ is influencing agent $i$. Equations [11][13] shows the influencing functions we used.

$$\text{Inf}_{PE}(j, t) = \exp\left(-0.5 \frac{d_1^2}{v}\right)$$ \hspace{1cm} (1)  

$$\text{Inf}_{FE}(j, t) = \exp\left(-0.5 \frac{d_2^2}{v}\right)$$ \hspace{1cm} (2)  

$$\text{Inf}_{BE}(j, t) = \exp\left(-0.5 \frac{d_2^2}{1/2v}\right)$$ \hspace{1cm} (3)

$v$ is the speed of agent $i$, $d_1$ is the distance in the $x$-direction between $i$ and $j$ and $d_2$ is the corresponding distance in the $y$-direction. The $y$-direction is the direction aligned with the velocity of agent $i$ and the $x$-direction is the direction perpendicular to the $y$-direction in the plane. It is important here to note that the distances $d_1$ and $d_2$ and the speed $v$ were scaled manually to get appropriate values on the influence values (equation[5]). The normalised influence function showing how much agent $i$ is influenced by agent $j$ is now given by:

$$\text{Inf}_{E}(j) = \frac{\max_t \text{Inf}_{E}(j, t)}{\sum_k (\max_t \text{Inf}_{E}(k, t))}$$ \hspace{1cm} (4)

where $t$ is the 1.6 second time window the example considers and where

$\text{Inf}_{E}(j, t) = \text{Inf}_{PE}(j, t) \cdot \begin{cases} 
\text{Inf}_{FE}(j, t) & j \text{ in front of } i \\
\text{Inf}_{BE}(j, t) & j \text{ behind } i
\end{cases}$ \hspace{1cm} (5)

Inf$_E(j)$ is what we call an influence value and will later be used to create new simulations from the example database. If the influence value is below a predefined cutoff value of 0.3 we say that influencing agent does not affect the path decisions of the subject agent. This influence agent will then not be saved into the example.

To account for the velocity of agent $j$ without explicitly using it in the influence functions and thereby complicating them, all positions of $j$ during a preset time window of 1.6 seconds will be considered. The position during this time window giving the highest influence value will be the position used in the influence function, as seen in equation[4]. How this will account for the velocity of agent $j$ can be seen in figure[9].
Figure 9: How positions over several frames affect the influence values. The two influencing agents, left (purple) and right (orange), have corresponding starting positions but the different velocities will give the purple agent a higher influencing value. [From [11]]

It is here important to note that the speed of $i$ and its position that is compared to the position of $j$ is the speed and position at the same frame as the compared position of $j$. If we consider Figure 9 the green subject agent has changed positions over all frames shown but the positions of the influencing agents that can be seen are always relative positions to the position of the subject agent at the same time. In other words the coordinate system shown in Figure 3 is a coordinate system always moving and rotating with the green subject agent.

3.3 New Simulations Using Examples

Using the example database, new crowd simulations can be made by simply adding a desirable number of agents with desirable start position and start velocity. All of these agents will then be checked every frame, every 40 milliseconds, to see if the agent in question needs an updated trajectory. The first thing to do here is to construct a query which reflects the current configuration around the agent. There are two points that determine whether an agent needs an updated trajectory:

- Is the current trajectory of the agent exhausted?
- Has the surrounding configuration changed so much since the last time it got assigned a trajectory that a new trajectory needs to be assigned?

If the answer is yes to any of these two questions the agent will be assigned a new trajectory, algorithm 2 and section 3.3.1 shows how this is done.

3.3.1 Update Trajectory

The first point checks whether there is any future trajectory for the agent and if no such trajectory exists a trajectory needs to be assigned for the agent to keep moving. A walking step is approximately 15 frames (0.6 seconds with 25 frames per seconds) long so therefore to make the agent move as smoothly as possible the second point is not checked for the first 15 frames after an agent is given a trajectory. After
the 15 frames however the second point will be checked every frame to make sure that an agent will not stay on a trajectory that is not realistic with regard to its current surroundings. The similarity between the current configuration and the configuration the last time a trajectory was assigned will be computed using the matching function presented in section 3.3.2 and in this case the former configuration will be treated in the same way as an example in the function. If the matching value is above 0.8 the surrounding configuration has not changed much and the agent can keep moving on its current trajectory.

If the agent in question needs to get a new trajectory the first thing to check is if it has any surrounding agents with influence value over the predefined cutoff value of 0.3. If that is not the case the agent go through every example in the database that does not have any influence values over 0.3. Of these examples the agent is given a trajectory at random from the ones with roughly the same speed as the agent. If an agent needs a new trajectory and has at least one surrounding agent with an influence value above 0.3 there are three points an example must fulfil for the agent to use its trajectory:

- The example’s trajectory can not lead to a collision
- The matching value between the example and the current configuration must be positive
- The matching value between the example and the current configuration must be as high as possible

So the trajectory that the agent in question will use is the trajectory of the example that fulfils the two first points and which has the highest matching value \( Sim(Q, E) \). The matching value \( Sim(Q, E) \) is described in section 3.3.2 in equation [6]. The agent will not be given the whole 40 frames long trajectory from the example but will only be given the first \((Sim(Q, E) \cdot 40)\) frames of the example’s trajectory.

### 3.3.2 Matching Function

The matching value is given by the matching function, \( Sim(Q, E) \), where

\[
Sim(Q, E) = S(Q, E) \cdot \left( \sum_{k \in Q} M_E(k) - Um(Q, E) \right).
\]  

(6)

Here, \( S(Q, E) \) is a function with the purpose to assure a smooth transition in terms of speed between the new trajectory and the trajectory which was used until now, and \( S(Q, E) \) is given by

\[
S(Q, E) = \exp \left( \frac{(v_e - v_q)^2}{1/v_q} \right).
\]  

(7)

\( v_e \) is here the speed of the subject agent in the example at the first frame of the example and \( v_q \) is the current speed of the agent which needs a new trajectory. \( Um(Q, E) \) in equation [6] is a penalty value lowering the matching value if not all influence factors of the agent in need of a trajectory are matched to the influence factors in the example and vice versa, given by

\[
Um(Q, E) = \frac{1}{2} \left( \sum_{k \in Q_{unmatched}} \text{Inf}_Q(k)^2 + \sum_{j \in E_{unmatched}} \text{Inf}_E(j)^2 \right).
\]  

(8)

\( M_E(k) \) gives an affinity value between an influencing factor, \( k \), influencing the agent in need of a trajectory and an influencing factor, \( j \), in the example:

\[
M_E(k) = \frac{\text{Inf}_Q(k) + \text{avg}(\text{Inf}_E(j))}{2} \text{Aff}(k, j).
\]  

(9)
avg(Inf_E(j)) divides Inf_E(j), the influence value of agent j, with the number of times that j was matched to an influence factor, k. Aff(k, j) is known as the affinity function and quantifies the affinity between an influence factor, k, and an influence factor, j, and is given by

\[
\text{Aff}(k, j) = \frac{\text{avg}(\text{SimVal}_{t'}(k, j))}{t'}.
\] (10)

The influencing factor j which gives the highest value of Aff(k, j) is the j that will be matched to that specific k in ME(k):

\[
j = \arg \max_{i \in E} \text{Aff}(k, i).
\] (11)

Aff(k, j) is given by the average of SimVal_t'(k, j) during the time window of the example (40 frames), and SimVal_t'(k, j) is given by

\[
\text{SimVal}_{t'}(k, j) = \exp \left( -\frac{(x_k - x_j)^2}{1/v_q} - \frac{(y_k - y_j)^2}{1/v_q} - \frac{\theta^2}{1/v_q} \right),
\] (12)

where the three Gaussians represents the similarity in position, direction and speed (in order). \(\theta\) is here the smallest positive angle between the directions of agent j and agent k. Since the query is not 40 frames long, the configuration of the missing frames is linearly extrapolated using the latest points of the query.

### 3.3.3 Collision Handling

Ideally, if our database was infinite in size there would always be an example which gives a high matching value and does not lead to a collision. In our case the database is however limited in its numbers of examples and situations can therefore occur when all examples with a positive matching function leads to a collision. When this occurs another method of choosing a trajectory for the query must be used. The affinity function is then used for all examples but with the input agents being the agent in need of a new trajectory and the subject agent of the example instead of using the influence factors. This will quantify the similarity between the last used trajectory by the agent in question and the trajectory of the example. The trajectory in the examples that gives the highest affinity value without leading to a collision is the trajectory that will be given to the agent in question.

### 3.3.4 Approximate nearest neighbour search

It should be noted that the content of this section is not included in the psuedo code as it is not part of the main algorithms of the crowd simulation method.

A problem with this method of simulation is that it takes a significant number of calculations to calculate the matching value between a query and an example. Because of this fact the time spent to create a simulation may become a problem for even rather short real video footage (and therefore small database). In [11] using an approximate nearest neighbour search is proposed so that instead of comparing a query to all examples in a database of in our case thousands of examples you can reduce the number of comparisons to around a hundred.

What we did is that when we read in the examples from the database, which only has to be done once in the beginning of a simulation, we took out 60 examples at random and called these our “bucket leaders”.
We then created a list called a “bucket” for each one of the bucket leaders. We then looped through all the rest of the examples and used the matching function described earlier in this section to compare each example with the bucket leaders and placed the example in the bucket with the best matching bucket leader. Thus, examples that are similar tend to go into the same buckets.

When we then needed to give a trajectory to a agent in our simulation we matched the agent in need of a trajectory and it’s configuration to all bucket leaders. From the bucket with the highest matching bucket leader, 40 examples had their matching value computed (compared to the simulated agent and it’s surrounding configuration). The best matching example was tested for collision, followed by the next best matching example if the first would yield a collision and so on. If none of these examples had a trajectory that did not lead to collisions and had a positive matching function, the normal collision avoidance method described in section 3.3.3 is used.
4 Evaluation

In this section an evaluation of how well our test scene works as a test case for our simulations is presented, this is done by comparing its characteristics to the characteristics of the video footage. After this the results of our simulations are presented and discussed, followed by a more general evaluation of example-based crowd simulations and its advantages and disadvantages.

4.1 Database Setup Scenario

The database was set up using the same corpus used in [11], more specifically the “Zara Data Set”[10]. This is a data set from a real video of a sparse crowd outside a clothing store. The real video footage is around 13 minutes long, and our way of creating the example database described in section 3.2 generated roughly 4000 examples using the data set.

4.2 Test Cases

The Unity 3D assets for the environment and the agents of our test cases were provided to us and their appearances can be seen in figure 10a and figure 10b. Our typical and most used test case is when we divide the agents into two groups, place them on two parallel lines and let the two halves of the agents walk towards each other, as seen in figure 10a and 10b.

![Figure 10: The starting positions of the agents in the typical test case](https://graphics.cs.ucy.ac.cy/research/downloads/crowd-data)

This type of scenario is effective for testing since it immediately forces a large number of conflicting paths for the agents, which can then be investigated.

The scene used for testing can be of great importance, a crowd’s behaviour might look very natural in one environment but very strange in another. A database made up from footage of a heavily trafficked pedestrian street during the morning rush will probably make unrealistic simulations of a beach at summertime. Our main test scene is the KTH entrance, which as can be seen in figure 11 has many similarities with the real-life place the video footage was recorded in. This is since both are (in this perspective) unlimited horizontally and bounded by walls and a road vertically. A difference between them is the entrance of
the Zara store in the real-life footage, where we can see people walk into during the videos, which does not exist in the KTH scene. This might lead to agents tending to try to walk towards and through walls in the simulation. Conversely, the KTH scene has a large opening in the middle where we know from experience that most of the people in real life would walk, where very few people in the simulation walks since the recorded footage is missing such a thing.

Figure 11: Two scenes in the projects, one real and one virtual. a) is taken from the video of the “Crowds by Example” data set vide [1].

Overall though the KTH environment is similar enough to the scene of the real footage to be good for simulations using our database, which is a big part of the reasons why the simulation looks realistic in its setting. However, if we were to use recorded footage of a completely different place with completely different conditions for our database the simulations would probably not look nearly as good, with agents behaving very differently from what we might expect in the environment, for example turning gradually all the time, in case the recorded footage was of a narrow curving walkway.

4.3 Results

We will in this section start out with presenting some observations we have seen when we have been running our simulations. We will after that present some information about the performance of our program.

4.3.1 Observations

A collision-free example-based crowd simulation was successfully created, and we note that it in our view looks realistic except for the fact that the agents perform their turns and speed changes abruptly and not spread out over a short period of time like it would in reality. An example of an encounter of two groups of agents can be seen in figure [12]. The simulation can in real-time handle around 10 agents without any bigger troubles (at 25 frames per second) on a laptop, however more agents will result in lesser frame rates and stuttering. Sometimes but rarely, in situations when there are many agents close to each other, the program will have trouble finding an example in the example database with a trajectory that does not lead to a collision which at the moment is unhandled since the situation is not mentioned or treated in [11].

The simulation that we created as of now does not take static geometry into account, which means that
the agents will walk straight through any environmental objects in the unity scene, for example the KTH building as seen in figure 11. In the simulations we can see many of the common behaviours of a real crowd, seen in the video footage, for example walking in pairs/groups, smooth collision avoidance and slowing down/stoping unpredictably.

We also note that the agents in the simulations looks like they have a predetermined goal in that they in most cases just make small changes in directions to avoid collisions and therefore mostly keep walking in roughly the same direction during the simulation even though they according to our implementation never had been given an objective to move towards.

Figure 12: 7 agents with conflicting paths meet and avoid each other, pictures are in order 1-6.

4.3.2 Performance

A graph displaying the performance of the real time simulation in terms of frames per seconds as a function of the number of agents in frame is shown in figure 13. In the figure the average frames per second is shown as well as the lowest recorded frame rate drop. The reason for the lowest frame rate drops being included is that the simulations sometimes has parts in them where there are more conflicting paths than otherwise. This results in an uneven work load, so the lowest frame rate drop can be seen as a metric for stuttering and lag spikes. We see in figure 13 that the performance of the simulation is very heavily dependant on the number of agents, the frames per second decrease from acceptable to very low as the number of agents go from 10 to 20 agents.

The performance seen in figure 13 is of the simulation program including the approximate nearest-neighbour search described in section 3.3.4. Using Unity 3D’s performance profiler we see that the absolute majority of the computations while running the program is computing the value described in equation 12.

The scene used for these tests are of the type shown in figure 10b and 10c, where half of the agents in the scene are moving straight towards the other half, resulting in a lot of conflicting paths. The computer running these tests had an Intel Core i5 6600K CPU and an Nvidia Geforce GTX 1070 GPU, although
it should be noted that the graphics rendering was never a strain on the simulation performance, and we consider the results to be rendering-independent.

Figure 13: A graph of the real time simulation’s performance in scenes like those in figures 10b and 10c where half of the agents in the scene walk towards the other half.

4.4 Evaluation of Results

To make sure the agents in our simulations change direction and speed smoothly, we would need to make a change in our database so that the agents in the examples do not change velocity abruptly. As mentioned in section 4.3 the program can in some situations not find any paths in the database that will not lead to a collision of agents. This is a problem that can always be solved by having a database that is bigger, however if no more data exists this problem could be solved by adding a final way of collision avoidance that occurs when the path of no example leads out of a collision. One way this collision avoiding solution could work is that the program tries to give the agent in question a path from another predefined list of paths that goes in many different directions. If none of these paths would make the agents avoid collision either the agent in question could simply be told to stand still for 40 frames. This way the agent tries a lot of very different ways of avoiding but if no one works, which will likely occur if the agent is completely surrounded, the agent simply stand still and let the surrounding agents move so that the agent in question may have a way out 40 frames later during the next check. This will probably not look so natural or realistic in all cases, and could probably sometimes induce a deadlock among the agents.

The reason why most of the agents in our simulation look like they have a predefined goal that they move towards, even though this was not explicitly implemented in the simulation program, is because of how the persons in our real video footage move. Most persons in the video move in roughly the same direction (horizontally). Because path segments of these persons are the paths that is given to the agents
in our simulations, our agents will continue in roughly the same direction as they are given at the beginning. Even if walking straight forward leads to a collision our agents will in most cases turn just enough to avoid collision and not more, since most persons in the video footage use the least amount of effort to avoid collisions.

The fact that we have not taken static geometry into account in our simulations is something that could affect the realism of our simulations in more ways than the agents simply walking through walls. In the video footage used to set up our database people does consider the static geometry of the setting (for example cars or walls) and does in some cases let it affect their trajectories. In our examples created by the video footage however we only consider other agents and not static geometry as influencing factors. This makes the trajectories of some of our examples unrealistic since that trajectory would only be realistic if the static geometry is taken into account. This can make some of our simulated agents move in a way that is seemingly unrealistic in its setting.

As can be seen in the results section and figure 13 the simulation is heavily dependant on the number of agents. This is because for every extra agent another search through the database has to be made almost every frame. More agents in the same area, as in our testing situations, also result in a higher number of conflicting paths. This leads to trouble finding a suitable path for the agents, since a collision is always to be avoided, and sometimes in the worst case to not finding a path at all in the example database. It should be noted that the test case used to get the results in figure 13 (two groups of agents walking straight towards each other) is a very specific and quite unnatural scenario primarily designed for forcing many conflicting paths, which means that in a normal sparse crowd scenario the frame rates might be slightly higher. Despite of this we still observe that the number of agents this kind of example-based crowd simulation is capable of handling in real time is probably too low for most applications. A solution until more processing power is available in computers is of course always to remove the real time aspect and just save down the positions of the agents and the replay it later, which would allow for much larger simulations.

As mentioned in section 4.3.2 the bottleneck of our simulation is equation 12. To calculate the matching value between a query and an example every influence factor in the query needs to be compared to every influence factor in the example. When this comparison needs to be done for every example in the database for every agent in the simulation and for almost every frame the number of calculations needed per second gets very high. If one would want to simulate a lot of agents using the method presented here, one would need to reduce the number of calculations needed for every agent at every second. One way of doing this is by using the approximate nearest neighbour search presented in section 3.3.4.

4.5 Analysis

There is a number of methods for simulating crowds that only take smooth collision avoidance into account, one of which is the ORCA method presented in section 2.1.3. These methods can make simulations where crowds move smooth and looks good, the disadvantage of these simulation methods is however that though they may look good they do not take into account the unpredictability and diversity of the human mind. Humans will not always act in the most logical and predictable way and because of this fact these
methods for simulation crowds will not make realistic simulations even though they might look good. This is where we believe the method of simulating crowds implemented in this paper is vastly superior. Since our simulations are based on how humans actually act in crowd situations, they do not only have smooth collision avoidance but also are very realistic if we only have a big enough database of situations similar to the simulated ones. One example of this is the observed group behaviour mentioned in section 4.3.1 which is not something that was explicitly programmed into our simulations (a research area in itself [16]) but came along due to it being inherent in the examples we used.

By doing this research we have showed how real footage of crowds can be used to create realistic crowd simulations. However it should be considered that the fact that we have based our notion of “what is a realistic behaviour” on only few minutes of footage is problematic. Some human behaviours in crowds that in fact are realistic and even common will never show themselves in our simulations just because no one in the video footage has that particular behaviour. In the same way some persons in our video footage may be acting in a very unusual way and since the video footage is relatively short our simulations may showcase this as an often concurring behaviour despite it being very uncommon in real life. To create real lifelike simulations that can show the great diversity of the human behaviour a lot more footage would have to be used as a base for the data-driven simulations. If we suppose for once that the video footage and therefore the database were infinite in size the simulations should be able to be showing off all the diversities of the human mind.

Even if a very large amount of video footage is used to create the database, this method of simulating crowds does however still have a major disadvantage, which we will show here using an example. Let us say we have collected a lot of video footage from the same central street during the morning rush and use this to build up a large database of examples. We will most likely be able to make very realistic simulations of crowds on that street during the morning rush. If we do however try to use this database to simulate crowd movements in the events of a tsunami we will end up with a simulation of people acting like it is a normal Monday morning even though they are seconds from getting hit by the tsunami. What we want tell using this rather extreme example is that video footage of a certain situation can only be used to simulate crowds during roughly the same conditions. This means that you can not just build up a big enough database and then use it for every possible simulation, you must rather think about exactly during what situation you wish your simulation to occur and then try to find video footage of very similar situations to use in your database.

Another limitation that we have noticed during our implementation of example-based simulations is the fact that the way of calculating the matching value to compare an example with a query seems rather arbitrary. Maybe there is more important information that we have not collected from the footage besides speed, direction and position and maybe we have not used the information in the best possible way in our comparison method. This applies for the influence functions as well, as mentioned in section 3.2.2 the scaling factors in the influence function equations [13] are fine-tuned manually, which of course is a source of error and can probably be more optimised. There are possibly also some more interesting influencing information that are unaccounted for in our influence values. It could for example be whether
the influencing agent is talking loudly or not.

There are some things that can be done to improve our simulations. One of these things is the issue that our program for simulations does not take the static geometry into account, which means that our agents can currently move through static obstacles like walls. One other issue that can make our simulations less realistic comes from the approximate nearest neighbour search we use to find which examples a query should be compared with. Using our approximate nearest neighbour search does not assure that a query always gets the most appropriate trajectory from the database, but only a trajectory that is fairly similar.

The examples in our database has been made from hand-traced information from the video footage. This does create a problem regarding the time aspect, this since the time it takes to track all persons movements in a video is rather high. This fact limits the amount of traced footage that can be collected for a simulation. This is a dilemma that in the future can be solved by letting a computer trace peoples movements in videos instead of doing it by hand.
5 Conclusion

A general summary of our results and analysis are presented. After this possible future work around the area of this report are discussed.

5.1 Summary

We mostly succeeded in re-implementing the example-based crowd simulation described in [11], and we saw a number of real crowd behaviours appear in our simulations even though they were not explicitly taken account for at all during our work. The performance of the simulation method is heavily dependant on the number of agents, which makes it most applicable for simulation sparse crowds.

Conclusively we feel that the example-based crowd simulation technique definitely has potential in creating reliable and realistic simulations, but it is fairly highly reliant on a large (the larger the better) database that fits the specific scenario that one wants to create a simulation for.

5.2 Future Work

A potential future research question that could be interesting to look into is the question of realistic versus realistic looking simulations, where one could investigate the correlation between what is realistic and what is perceived as realistic. An interesting investigation would be to find out more exactly in what aspects the perception of realistic movements differs from real movements. This information could possibly be used to answer why the perception of reality is not always the same as what is truly real. Earlier studies in this research are has been made in [15], which tries to decide if a crowd behaviour using a social forces model is more realistic or not, and [16], which studies what makes a group formation in a crowd perceptually plausible. There is also [4] and [3] which investigate how position, orientation and camera viewpoint affects the plausibility of pedestrian formations.

There is also some interesting future work mentioned by Lerner et. al. in [11], which applies to our work as well. The agents in this simulation method are, even though they sometimes may look like it, not goal oriented. A future project could be to try to incorporate a goal oriented behaviour for agents in example-based simulations. One could do this by adding a negative penalty value to the matching value of examples whose trajectories move the simulated agent to much away from the goal of the agent. Other possible future work that is mentioned in [11] is that, as mentioned in section 4.5, the matching and influence functions are not optimal, and work could done in trying to optimise these functions.

There are plenty of ways of evaluating different aspects of simulated crowds, a few of which are mentioned in section 2.2. In [12] the data-driven evaluation method mentioned in section 2.2.5 is used to evaluate example-based simulations. In future works it would be possible to evaluate the example-based simulation method using other types of evaluation methods to see how it performs. One could also investigate which evaluation methods are best to use depending on the goal of the simulation.

One more thing to look into in the future regarding example-based simulations could be to see in what ways one could make the simulations faster and therefore make simulations work for larger crowds. As
mentioned in this article we use an approximate nearest neighbour search to make the simulations faster, however there might be a better way to solve this problem.
6  Acknowledgements

We want to thank Alon Lerner for helping us out by answering questions and giving us further information about how to implement the example-based crowd simulation method. We would also like to thank Jack Shabo for providing us with the assets for human-like agents used in the earlier part of our project and the KTH environment assets used in the final version. A thanks goes out to Halit Anil Dönmez for providing us with the human-like agents used in our final version. Finally our special thanks goes out to our supervisor Christopher Peters for giving us a lot of inspiration and guidance during the whole project.
References


